Le rendu expressif, un ensemble d'outils et techniques pour la communication visuelle

Joëlle Thollot

▶ **To cite this version:**

Joëlle Thollot. Le rendu expressif, un ensemble d'outils et techniques pour la communication visuelle. Interface homme-machine [cs.HC]. Institut National Polytechnique de Grenoble - INPG, 2008. ⟨tel-00383924⟩

HAL Id: tel-00383924

https://theses.hal.science/tel-00383924v1

Submitted on 13 May 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

HAL Authorization

Habilitation à diriger des recherches

Le rendu expressif
Un ensemble d'outils et techniques pour la communication visuelle

Joëlle Thollot

7 novembre 2008

Table des matières

1 Introduction **5**
- 1.1 Comment en suis-je arrivée là ? . . . 5
- 1.2 Qu'est-ce-qui fait une « bonne » image ? . . . 7
- 1.3 Organisation du document . . . 9

2 Outils de création d'images **11**
- 2.1 Simulation de média traditionnels . . . 11
- 2.2 Dessin vectoriel . . . 13
  - 2.2.1 Les courbes de diffusion : une nouvelle primitive de dessin vectoriel . . . 13
  - 2.2.2 Dessin de motifs par l'exemple . . . 15
- 2.3 Contrôler le rendu par une image . . . 16

3 Contrôle du détail visuel **19**
- 3.1 Détail et regroupement . . . 19
- 3.2 Détail et modèle de la vision humaine . . . 21
- 3.3 Ce qui est important n'est pas du détail . . . 22
  - 3.3.1 Perception de la couleur . . . 22
  - 3.3.2 Perception de la forme . . . 23
  - 3.3.3 Perception des matériaux . . . 25

4 Cohérence temporelle **27**
- 4.1 Quel est le problème ? . . . 27
- 4.2 Papier dynamique . . . 28
- 4.3 Advection de texture . . . 29
- 4.4 Distribution de points . . . 30
- 4.5 Motifs 2D dynamiques . . . 31
- 4.6 Bilan . . . 32

5 Conclusion **35**

A Etat de l'art **41**

B Principales publications **73**

Chapitre 1

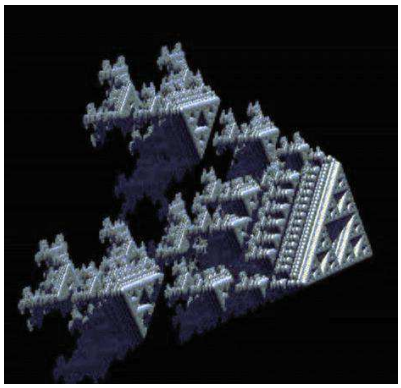
Introduction

Mes travaux de recherche, si l'on inclut mes travaux de thèse, ont porté sur différentes disciplines de l'informatique graphique. Néanmoins mon questionnement sous-jacent est resté le même et continue d'orienter mon projet de recherche actuel. Je le résumerai à l'étude de la communication visuelle et des outils informatiques permettant de l'améliorer. Cela implique en particulier d'être en lien avec les utilisateurs de tels outils, c'est-à-dire les personnes qui vont vouloir communiquer un message visuel et qui ne sont pas nécessairement des spécialistes de l'informatique. Cela implique aussi, au delà des aspects purement algorithmiques, de comprendre ce qui est important dans une image (nécessairement dépendant de l'application) pour concentrer les efforts sur ces points, que ce soit les temps de calcul ou la qualité visuelle.

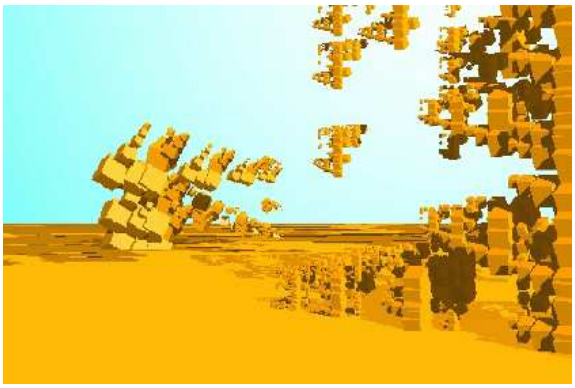
Ce thème fondamental s'est décliné en divers travaux (modélisation géométrique fractale, rendu temps réel de scènes complexes, rendu expressif) et diverses applications (arts graphiques, architecture, archéologie). J'ai choisi de ne présenter dans ce mémoire que mes travaux sur le rendu expressif car c'est l'objet de mes recherches actuelles et c'est le sujet qui m'est le plus personnel puisque je ne l'ai démarré qu'une fois en poste et suite à mes propres réflexions et contacts. Je commencerai néanmoins par un bref historique de mes travaux me permettant de motiver ma démarche et de citer les diverses personnes avec qui j'ai collaboré car ce sont en partie grâce à elles que ce mémoire existe aujourd'hui.

1.1 Comment en suis-je arrivée là ?

J'ai effectué ma thèse au LIGIM (maintenant LIRIS) à l'université Claude Bernard à Lyon sous la direction de Denis Vandorpe et Eric Tosan [Tho96]. Durant ma thèse j'ai travaillé sur la modélisation géométrique fractale. Notre but était de faciliter la création d'objets 3D ayant des propriétés fractales car ces objets sont en général difficiles à modéliser par les techniques standards. Nous avons pour cela défini des opérateurs permettant de composer des objets fractals simples pour obtenir des objets complexes ([TT93] [TT95] [Tho97] [TZTV97]). Ce prinipe est connu en modélisation géométrique classique sous le nom de CSG (Constructive Solid Geometry). J'ai défini de tels opérateurs basés sur des notion ensemblistes mais aussi sur des opérateurs spécifiques aux langages formels comme la concaténation ou le mélange (figure 1.1 à gauche). Un objet fractal peut en effet être vu comme un langage sur un alphabet constitué de transformations affines. Ces travaux ont été menés en étroite collaboration avec une artiste, Martine



Mélange de deux fractales



Voyage en Egypte ©Martine Rondet-Mignotte

FIG. 1.1 – Modélisation géométrique d'objets fractals.

Rondet-Mignotte, qui appliquait les méthodes proposées dans l'équipe pour sa propre production artistique (figure 1.1 à droite). Cette collaboration nous a permis de valider l'utilisabilité des outils de modélisation que nous développions.

Lors de mon arrivée dans l'équipe iMAGIS du laboratoire GRAVIR j'ai changé de thématique de recherche puisque les domaines de recherche privilégiés de l'équipe étaient le rendu ou l'animation. Je me suis naturellement tournée vers le rendu dont la vocation est la visualisation et donc, en quelque sorte, la fin de la chaîne de construction d'une image. Je me suis alors intéressée aux techniques permettant le rendu interactif de scènes complexes : niveaux de détails et visibilité [DDTP00]. La visualisation de scènes urbaines était une des applications de ces travaux et j'ai donc cherché à nouer des contacts avec des architectes. Cela a débouché sur une collaboration avec l'école d'architecture de Lyon (laboratoire ARIA) durant la quelle j'ai en particulier travaillé avec Xavier Marsault et Renato Saleri. Au cours de nos discussions, j'ai pris conscience du manque de visualisations adaptées à leur domaine. Toute une partie du travail de l'architecte débute par des dessins et croquis préparatoires. Suit une étape de modélisation avec des logiciels spécialisés, comme AutoCAD, puis une présentation du projet aux clients avec des visualisations 3D tirées du modèle géométrique mais aussi de nombreux dessins. Ensuite, même lors du raffinement du projet, l'utilisation de dessins reste présente pour donner une vision globale du projet.

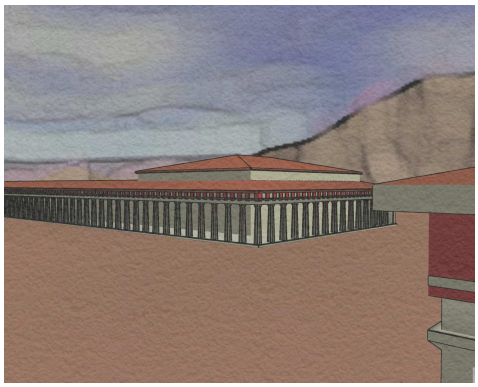
J'ai alors à nouveau ré-orienté ma recherche, tout en restant dans la problématique du rendu interactif, vers des techniques plus expressives que l'on appelle aussi non-photoréalistes. J'ai, en plus des architectes, pris contact avec des archéologues qui sont aussi de grands utilisateurs de dessins pour illustrer leurs recherches. Ces collaborations ont été formalisées dans le projet région DEREVE et l'ARC ARCHEOS. J'ai en particulier travaillé avec l'équipe Homerica de l'université Stendhal (Grenoble), et plus spécifiquement avec Françoise Létoublon et Isabelle Ratinaud sur un projet de visualisation du site d'Argos comme montré figure 1.2 ([RT03]). Un des bâtiments du site avait en effet deux interprétations contradictoires et la possibilité de montrer une restitution 3D du site avec une visualisation de type aquarelle permettait de souligner l'aspect hypothétique des choix d'architecture effectués (et accessoirement de ménager les susceptibilités des archéologues auteurs des deux interprétations).

Ces exemples reflètent bien pour moi la problématique de la communication visuelle qui ne se restreint pas à la production d'images réalistes alors que la majeure partie des travaux de

Le site d'Argos

Aquarelle réelle d'un monument

Animation d'une foule [HLTC03]



Rendu avec papier dynamique [CTP+03]

FIG. 1.2 – Restitution archéologique dans le cadre de l'ARC ARCHEOS.

recherche en synthèse d'images s'est portée sur la création d'images ressemblant à une photo. Ces premiers travaux ont marqué pour moi le début des travaux de recherche que je poursuis encore aujourd'hui et que je présente dans ce mémoire.

1.2 Qu'est-ce-qui fait une « bonne » image ?

De manière générale pour produire une image il faut un auteur et un outil. Il parait assez évident que pour faire une « bonne » image il faut un « bon » auteur et un « bon » outil. La recherche en informatique graphique vise à faire de bons outils mais il est crucial de garder à l'esprit que ces outils vont être utilisés par un auteur. L'informatique étant un outil d'automatisation de tâches, une question fondamentale dans la création d'images est ainsi de savoir ce qu'il est raisonnable d'automatiser et ce qui est du ressort de l'auteur, puis comment l'auteur va pouvoir aisément assurer sa part du travail. Cette question est très dépendante de l'application que l'on vise. Selon les métiers un certain nombre de règles pourront être utilisées pour automatiser certaines tâches. La liberté laissée à l'auteur dépendra aussi fortement de son but. Un graphiste souhaitera certainement plus de possibilités de contrôle qu'un archéologue. Le premier critère de qualité de nos approches sera donc la juste place du contrôle utilisateur.

Il est probablement vain d'espérer lister et formaliser de manière exhaustive les critères de qualité d'une image. On peut néanmoins dégager un certain nombre de principes qui sont autant

de guides pour la production d'outils de création d'images performants.

Lisibilité. Pour qu'une image communique efficacement un message il faut qu'elle soit lisible, autrement dit compréhensible. Pour cela, quelque soit la technique picturale utilisée et la scène représentée, les artistes ont souvent travaillé sur la gestion du détail. Un photographe va, au travers de la profondeur de champ, être en mesure de mettre en valeur certaines parties de la scène, un illustrateur va pouvoir choisir de détailler certaines parties de son dessin ou de varier l'intensité et la précision de son trait selon son but. Une partie de mes travaux s'est ainsi attachée à proposer des outils de création ou de traitement d'images permettant de contrôler le détail visuel.

Absence d'artefacts. Un critère important de qualité d'une image est l'absence d'artefacts visuels. Cette considération a toujours existé (par exemple le peintre cherche à éviter les craquelures lors du séchage de ses toiles) mais est particulièrement importante dans le cas de l'informatique graphique. En effet, la nature même du médium (une image discrète) est source naturelle d'artefacts. C'est pourquoi je me suis intéressée à des techniques de dessin vectoriel qui permettent de s'affranchir de la nature discrète de l'outil informatique. Les artefacts sont encore plus visibles lorsque l'on s'intéresse aux animations. Dans ce cas, des artefacts temporels s'ajoutent aux artefacts spatiaux pouvant amener une baisse radicale de la qualité du résultat. Je me suis donc intéressée dans de nombreux travaux à la question de la cohérence temporelle qui vise à limiter les artefacts temporels dans les films d'animation stylisés.

Esthétisme. Une bonne image doit être belle ! La beauté étant très subjective, il parait difficile de se baser sur un critère esthétique pour conditionner un outil. Il n'en reste pas moins qu'au cours de l'histoire de l'art les peintres et dessinateurs ont produit de magnifiques images avec les outils dont ils disposaient. Une partie de mes travaux a donc consisté à tenter de reproduire certains effets des média traditionnels tout en laissant un contrôle suffisant à l'utilisateur pour qu'il puisse créer l'image de son choix. Dans ce domaine le compromis entre automatique et manuel est fondamental. La collaboration avec des graphistes professionnels permet de vérifier la pertinence des choix effectués.

Figuration. Si l'on exclut l'art abstrait, une bonne image doit efficacement représenter la scène figurée. Une manière de simplifier le travail de l'auteur est alors simplement de lui permettre de créer ses images à partir d'une donnée existante (photographie, vidéo, scène 3D). Dans le cas des animations stylisées cela devient un gain de temps notable mais dans le cas général cela permet aussi à des non-spécialistes d'accéder plus facilement à la création artistique. Ainsi, mes travaux ne se sont pas spécialisés à un type de données. Ce critère est aussi en lien avec la gestion du détail. Si l'on propose un algorithme qui est capable d'enlever du détail, cet algorithme doit aussi préserver la forme générale de la scène représentée.

Une image est faite pour être vue. Le but d'une image est d'être montrée à un être humain. Cela nous donne un cadre supplémentaire pour guider nos outils : la perception humaine. D'un coté il est inutile de représenter des choses que l'oeil humain ne peut percevoir ; d'un autre coté il est important de concentrer nos efforts sur les éléments auxquels l'homme est sensible.

La prise en compte des connaissances que les scientifiques ont acquises dans ce domaine est ainsi une source d'information fructueuse pour la création d'images. Les travaux présentés ici utilisent en particulier des résultats sur la perception des couleurs, le regroupement perceptuel et la théorie des espaces d'échelle (*scale-space*).

1.3 Organisation du document

Dans la suite du document je présente les travaux que j'ai effectués depuis 2003. Ces travaux sont le résultat de plusieurs collaborations scientifiques. En premier lieu avec mes étudiants de thèse ou de master, ainsi qu'avec plusieurs collègues français et étrangers. Tous ces collaborateurs seront cités lors de la description des travaux correspondants.

Un état de l'art sur le rendu expressif est donné en annexe A sous la forme du chapitre sur le rendu expressif publié dans l'ouvrage "Informatique graphique et rendu" [BTT07] et écrit en collaboration avec Pascal Barla et Gwenola Thomas.

Je présenterai dans la suite mes contributions regroupées en trois grands thèmes : outils de création d'images ; simplification et abstraction ; cohérence temporelle. Une description détaillée sera donnée en annexe sous la forme d'une collection d'articles. Je conclurai enfin par des perspectives de recherche.

Chapitre 2

Outils de création d'images

Dans ce chapitre je présente des travaux cherchant à produire des outils de création d'image. Ces outils visent essentiellement les graphistes et une part importante du questionnement est de trouver les bons contrôles à laisser à l'utilisateur de tels outils. Je travaille depuis de nombreuses années avec des infographistes (Philippe Chaubaroux, Florent Moulin, Laurence Boissieux, Isabelle ?) afin de déterminer leurs besoins et les compromis acceptables pour eux. Il s'avère que les utilisateurs actuels des logiciels proposés sur le marché (que ce soit pour de la retouche photo, du dessin vectoriel, du traitement de vidéo ou de l'animation 3D) sont habitués à manipuler des interfaces extrêmement complexes et supportent de très longs temps de calculs sans sourciller. En revanche leur demande majeure est de pouvoir contrôler finement le résultat visuel final et de pouvoir créer des images en y mettant leur style personnel. Dans le même temps et pour des raisons historiques, une bonne partie des travaux de recherche en informatique graphique vise à accélérer les calculs et à augmenter l'automatisation.

Les travaux que je présente ici ont ainsi pour but essentiel de redonner du choix au graphiste. Bien évidemment nous chercherons tout de même à optimiser les temps de calculs car cela ne gâche rien.

2.1 Simulation de média traditionnels

Un gain précieux de l'outil informatique par rapport au dessin traditionnel est d'offrir la possibilité de produire des images à partir d'une donnée existante : photographie, vidéo, scène 3D. Une façon de donner du choix au graphiste est alors d'étendre les possibilités des outils de traitement en terme de richesse visuelle. En effet, la gamme des styles actuellement disponibles reste relativement réduite et varie selon le type d'entrée à traiter. Au niveau industriel, si l'on prend le cas des animations 3D ou des vidéos, les styles disponibles se résument grosso-modo à un style photo réaliste et un style dessin animé (contours et aplats de couleur). Une gamme plus importante est disponible pour le traitement de photographies, essentiellement sous forme de filtres d'images. En revanche de nombreux travaux académiques (voir l'annexe A) ont cherché à reproduire des média plus variés tels que le hachurage, la peinture, l'aquarelle, etc...

Pour que ces méthodes soient intéressantes pour le graphiste, il faut d'une part qu'elles lui laissent suffisamment de liberté et d'autre part qu'elles offrent un réel bénéfice par rapport au dessin traditionnel. Pour cela il nous semble important que les méthodes que nous proposons

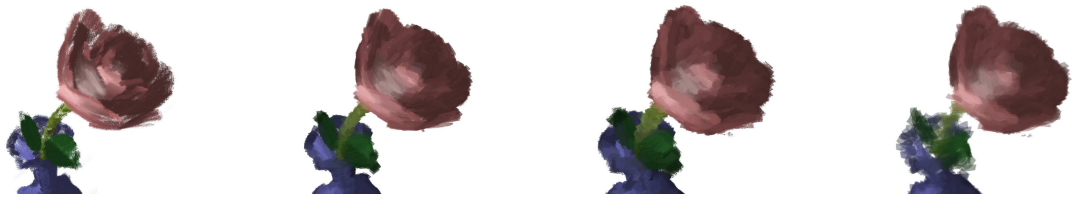


FIG. 2.1 – Modèle à base de marques : un même objet 3D rendu dans quatre styles différents.

FIG. 2.2 – Modèle à base de textures : rendu aquarelle obtenu à partir d'une photographie (gauche) ou de modèles 3D (milieu et droite).

ne soient pas spécifiques à un seul type d'entrée. Nous cherchons ainsi à être capable de traiter aussi bien des images que des animations (vidéos ou 3D).

Nous avons étudié deux modèles informatiques pour reproduire les effets de média traditionnels : un modèle à base marques et un modèle à base de textures. Le modèle à base de marques est défini par l'analogie avec la peinture dans laquelle une image est constituée d'un ensemble de coups de pinceaux. Le modèle à base de texture représente une image sous une forme plus continue. Les effets du média sont simulés par une composition de textures et de filtres. Ce type de représentation est parfaitement adaptée à des média comme l'aquarelle où il n'y a pas de coups de pinceaux distincts mais plutôt des variations de fréquence caractéristiques du médium.

Le modèle à base de marque (figure 2.1) a été étudié par David Vanderhaghe dans le cadre de sa thèse. La problématique principale de ces travaux est de donner au graphiste la possibilité de définir son style. David a proposé un modèle de style et une implémentation concrète de ce modèle [VBTS07b]. Ce principe a été validé par la réalisation d'un court métrage [MBV+07] en collaboration avec deux infographistes 3D : Florent Moulin et Laurence Boissieux.

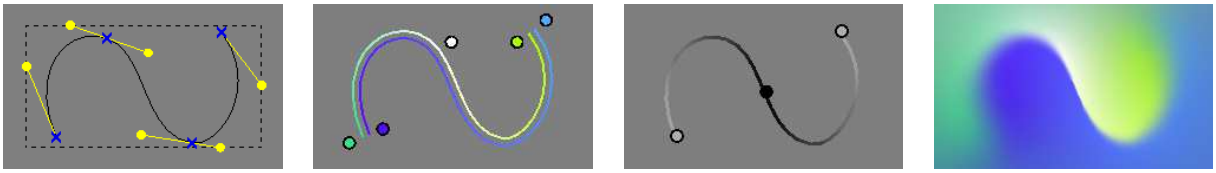
Le modèle à base de texture (figure 2.2) a été étudié par Adrien Bousseau dans le cadre de sa thèse en spécialisant l'étude pour l'aquarelle. La problématique principale était la définition des effets caractéristiques de l'aquarelle et leur modélisation sous forme de textures et de traitements d'images. Adrien a proposé un pipeline de traitement d'image paramétrable pour l'aquarelle [BKTS06]. Ces travaux ont donné lieu à une collaboration avec David Salesin (Adobe research lab). Adrien a produit un plugin photoshop et un plugin After Efffect durant un séjour chez Adobe Seattle. L'approche a été validée par un deuxième court métrage réalisé par Laurence Boissieux.

Pour ces deux modèles une question spécifique se pose lors du traitement d'animations : la gestion des artefacts temporels. Nous y reviendrons plus en détail au chapitre 4.

2.2 Dessin vectoriel

Parmi les outils informatiques pour la création d'image, le dessin vectoriel offre de nombreux avantages. Les primitives de dessin sont représentées par un modèle paramétrique et sont ainsi facilement éditables et indépendantes de la résolution de l'image finale. Les outils comme Flash ou Illustrator en sont les exemples les plus connus. Ils sont principalement utilisés pour produire des illustrations ou des courts métrages d'animation 2D (la paramétrisation des primitives permettant des techniques de morphing bien plus efficaces que sur des images pixelliques). Toujours dans le but de donner au graphiste des outils plus performants nous nous sommes intéressés à deux problèmes dans le cadre du dessin vectoriel. Tout d'abord nous avons proposé une nouvelle primitive de dessin permettant en particulier le dessin de dégradés complexes de manière intuitive. Deuxièmement nous nous sommes intéressés à la question de la création de motifs à partir d'un exemple donné par l'utilisateur. Le point commun de ces deux approches est d'une part de faciliter le travail du graphiste et d'autre part de s'inspirer de travaux liés à la perception humaine.

2.2.1 Les courbes de diffusion : une nouvelle primitive de dessin vectoriel



(a) (b) (c) (d)

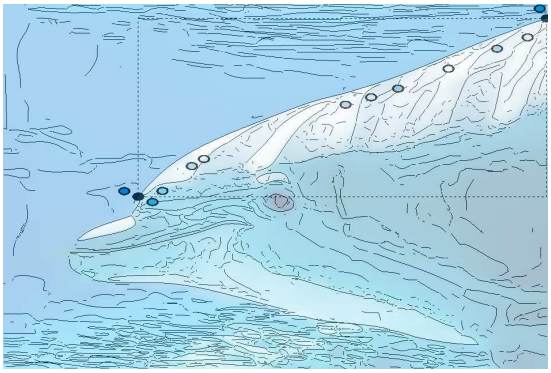
FIG. 2.3 – Une courbe de diffusion est composée de (a) une courbe de Bézier, (b) des contraintes de couleurs arbitrairement positionnées de chaque coté de la courbe puis interpolées, (c) des contraintes de flou interpolées linéairement le long de la courbe. (d) L'image finale est obtenue par diffusion et floutage.

Ce travail a été effectué par Alexandrina Orzan et Adrien Bousseau dans le cadre de leur thèse et en collaboration avec Holger Winnemoller (Adobe), Pascal Barla (IPARLA - INRIA) et David Salesin (Adobe). En nous appuyant sur des travaux en vision montrant que l'information contenue dans une image peut être concentrée sur les arêtes de l'image, nous avons proposé une nouvelle primitive de dessin vectoriel constituée d'une courbe de Bézier au long de laquelle sont définies (aussi de manière vectorielle) une couleur gauche, une couleur droite et un niveau de flou (voir figure 2.3). L'image finale est ensuite construite par diffusion des couleurs et du flou. Ce modèle très simple permet de dessiner de manière intuitive des dégradés complexes mais nous avons aussi proposé une méthode d'extraction automatique qui construit l'ensemble des courbes de diffusion approximant au mieux une image donnée (voir figure 2.4).

Cet outil a été validé par Laurence Boissieux et Philippe Chaubaroux par la production de divers dessins vectoriels (voir figure 2.4) et a été l'objet d'une publication à Siggraph 2008 [OBW+08].

Conversion automatique

Image originale



Courbes automatiquement extraites

Image vectorisée résultat

Différence entre les deux images

Dessins effectués par des artistes

©Laurence Boissieux

©Philippe Chaubaroux

FIG. 2.4 – Courbes de diffusion.

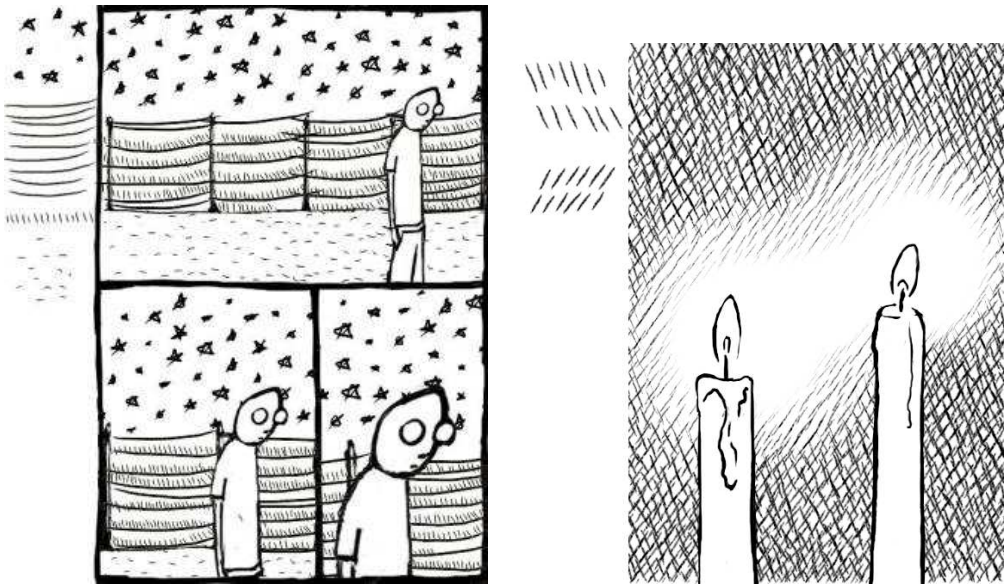


FIG. 2.5 – Méthode paramétrique de synthèse de points et hachures.

2.2.2 Dessin de motifs par l'exemple

Ce travail a été effectué par Pascal Barla dans le cadre de sa thèse et en collaboration avec Simon Breslaw et Lee Markosian (University of Michigan). L'objectif était de fournir aux graphistes un outil leur permettant de produire des motifs vectoriels. Par motif nous entendons un ensemble d'éléments vectoriels répartis le long d'un chemin 1D ou dans une région 2D selon une distribution similaire sur toute la forme. Nous avons choisi une approche "par l'exemple" car cela permet au graphiste de se libérer des contraintes liées à la description de son style. Il donne un exemple en dessinant quelques éléments du motif et le système informatique se charge de reproduire ce motif sur une plus grande région. Toute la difficulté de ce type d'approche est d'être capable de capturer les caractéristiques importantes du style et d'éviter des répétitions visibles dans l'image finale. Nous avons proposé deux approches inspirées de la synthèse de texture.

La première (voir figure 2.5) est une approche paramétrique où l'on analyse les statistiques de l'exemple et où l'on recrée un motif similaire ayant les mêmes statistiques. Cela suppose de définir un modèle paramétrique de ce type de motifs. Nous avons basé ce modèle sur la théorie du regroupement perceptuel. Cette théorie nous indique en effet comment le système visuel humain groupe des éléments similaires répétitifs en donnant divers critères (continuation, parallélisme, proximité). Cependant ce modèle ne décrit correctement que des éléments simples (points, segments).

Pour être en mesure de prendre en compte des éléments plus complexes, nous avons proposé une autre méthode cette fois non paramétrique (voir figure 2.6). Ce type d'approche consiste à placer des éléments un par un en cherchant localement dans l'exemple des voisinages similaires à celui où l'on veut ajouter un nouvel élément. Il n'y a ainsi plus besoin d'un modèle complet du motif mais il faut définir ce qu'est un voisinage et quelle est la mesure de similarité entre deux voisinage. Une fois encore la théorie du regroupement perceptuel nous a fourni des indices de similarité. Cette approche permet de synthétiser des motifs constitués d'éléments quelconques

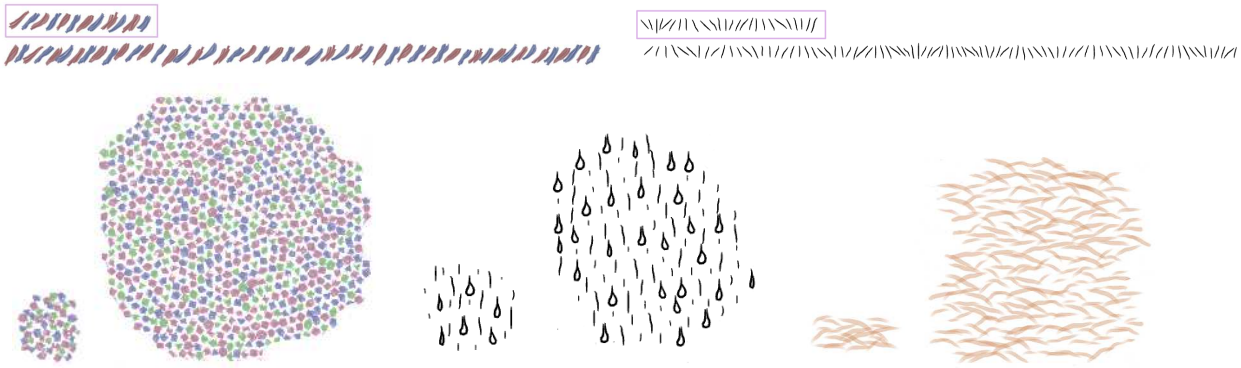


FIG. 2.6 – Méthode non paramétrique de synthèse de motifs 1D et 2D.

mais restreint la distribution à être quasi uniforme. Cette méthode a été publiée à Eurographics 2006 [BBT+06].

Ce domaine reste de mon point de vue très ouvert. Des travaux récents de collègues japonais ont étendu notre deuxième approche mais il reste encore beaucoup de travail à faire pour être en mesure d'offrir des outils pouvant couvrir la gamme des motifs que les graphistes souhaitent pouvoir dessiner.

2.3 Contrôler le rendu par une image

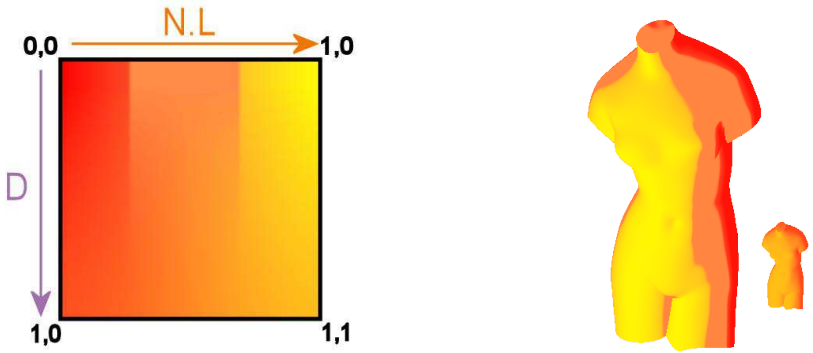


FIG. 2.7 – Xtoon : (à gauche) une texture 2d est utilisée pour controler l'éclairage ($N.L$) en fonction d'un paramètre de détail D. Par exemple (à droite) D peut être la profondeur depuis la caméra.

Les idées simples étant parfois les meilleures, je termine ce chapitre par un travail effectué par Pascal Barla dans le cadre de sa thèse et en collaboration avec Simon Breslaw et Lee Markosian (University of Michigan) qui illustre l'importance du choix de l'outil de contrôle laissé au graphiste. Dans ce projet Pascal a proposé de contrôler le rendu d'un objet 3D grâce à une image 2D, telle que montrée figure 2.7. Cette image est une interface extrêmement intuitive pour corréler 2 paramètres (correspondant aux deux axes de l'image).

C'est un principe très général et Pascal l'a mis en oeuvre dans le cas du contrôle de l'éclairage d'un objet. L'axe x de l'image correspond à l'illumination de l'objet (produit scalaire entre

FIG. 2.8 – Xtoon : différents rendus obtenus.

la normale et la direction de la lumière) et la couleur du pixel de l'image donne la couleur du point 3D sous l'illumination correspondante. L'axe y permet alors de faire varier cette couleur selon un autre paramètre. Ce peut être la profondeur ce qui permet par exemple de désaturer ou de simplifier la couleur en fonction de la distance au point de vue. Ce peut être la distance à un point donné dans la scène pour permettre une mise en valeur d'une certaine zone de la scène. Mais l'on peut imaginer n'importe quel autre couplage.

Cette technique est codable très facilement en utilisant le GPU et offre ainsi un contrôle temps-réel permettant une large gamme de rendus (voir figure 2.8). Le graphiste peut aisément créer sa propre image 2D pour contrôler l'effet qu'il souhaite obtenir.

Cette technique a été implémentée dans un plugin de rendu aquarelle développé par David Lanier (société dl3D) dans le cadre d'un contrat de recherche avec Studio Brocéliande (un studio de production de films d'animations situé à Paris) et publiée à NPAR 2006 [BTM06].

Chapitre 3

Contrôle du détail visuel

Ce chapitre présente des travaux visant à contrôler le détail visuel dans une image. Cette thématique est souvent décrite comme de l'abstraction visuelle utilisée ici au sens simplification de la donnée d'entrée. Cette notion est commune à tous les arts graphiques car c'est une manière d'améliorer la lisibilité d'une image et de mettre en valeur les éléments clés (fonction du message que veut transmettre le créateur de l'image).

Dans ce contexte, abstraire revient ainsi à enlever du détail. Cependant, les critères permettant de choisir ce qui peut être considéré comme du détail sont mal définis car la notion de détail est mal définie. Cette notion est en partie sémantique : c'est l'application visée ou le graphiste qui fixe ce qui est important. Pour une autre part, le détail est lié à la perception humaine. Intuitivement, ce qui ne se perçoit pas ou peu est du détail.

Dans ces travaux nous nous sommes ainsi posé la question de savoir ce qu'est du détail pour diverses applications. Il me parait encore aujourd'hui impossible de donner une définition de cette notion dans un cas général. En revanche nous avons exploré un certain nombre d'outils mathématiques et de théories de la perception pour asseoir nos divers travaux. Nous avons aussi proposé une méthode plus tournée vers le graphiste, lui permettant de d'inclure sa propre sémantique dans les critères de simplification.

3.1 Détail et regroupement

FIG. 3.1 – Intérêt de la segmentation pour l'aquarelle. A gauche l'image originale, au milieu la version aquarelle sans segmentation, à droite la version aquarelle après segmentation.

Une première manière de définir ce qu'est du détail est de considérer que si l'on peut grouper

FIG. 3.2 – Simplification de dessins au trait.

deux éléments d'une image en un seul sans changer la sémantique de l'image alors la précision apportée par ces deux éléments était superflue. Par conséquent une manière de simplifier une image est de regrouper ce qui est similaire pour le remplacer par un unique représentant. Toute la question est alors de définir ce que veut dire similaire.

Une application directe de ce principe est la segmentation d'image (voir figure 3.1). Dans ce cas deux pixels sont similaires s'ils sont proches spatialement et ont des couleurs similaires. Nous avons ainsi utilisé une segmentation basée sur un mean-shift dans le cas de l'aquarelle pour créer des zones de couleur uniformes dans l'image sources [BKTS06].

Nous avons d'autre part appliqué ce principe au dessin au trait vectorisé. Le but était, partant d'un dessin au trait noir et blanc constitué de lignes vectorielles munies d'une épaisseur, de le simplifier en un ensemble plus petit de lignes. Ce travail a été effectué par Pascal Barla dans le cadre de sa thèse [BTS05a, BTS05b]. Pascal a proposé un algorithme glouton de regroupement progressif des lignes à une échelle choisie par l'utilisateur. Le critère de similarité des lignes est

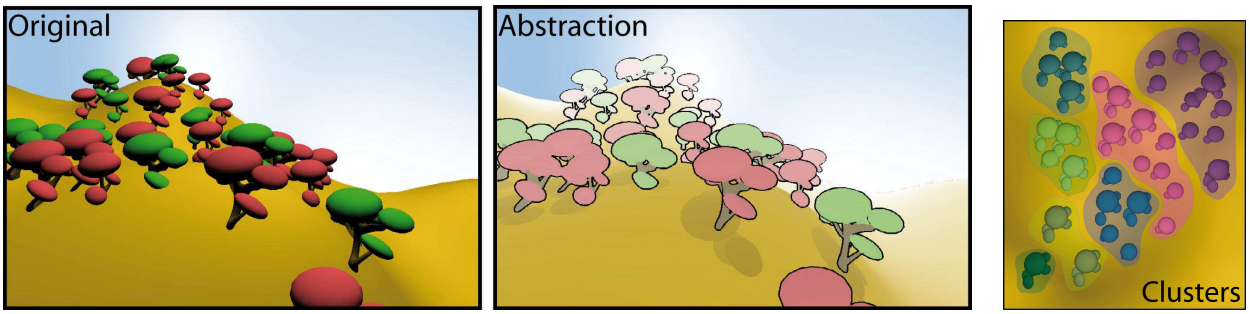


FIG. 3.3 – Groupement 3D permettant de produire une stylisation d'une scène 3D.

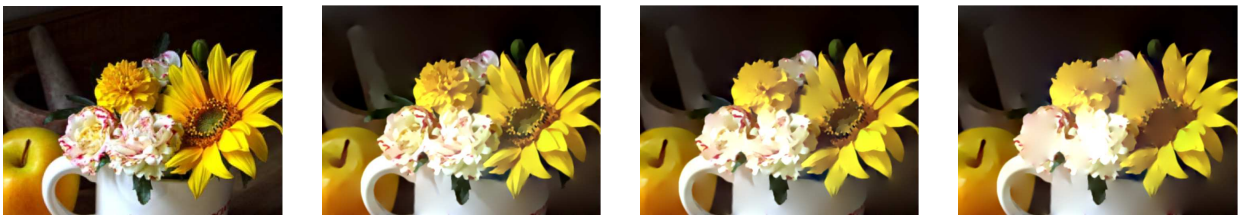
donné par la théorie du regroupement perceptuel auquel on peut ajouter des critères de couleur. Une fois le groupement obtenu chaque groupe de ligne peut être redessiné selon une stratégie choisie par l'utilisateur et dépendante de l'application : ne garder qu'une seule ligne du groupe, construire la ligne médiane, etc. La figure 3.2 montre deux exemples de simplification avec deux stratégies de dessin différentes : en haut la ligne médiane de chaque groupe est dessinée ; en bas seule la ligne la plus longue de chaque groupe est conservée. Cette deuxième stratégie fonctionne très bien pour des applications de type niveaux de détail.

Enfin, nous nous sommes intéressés au regroupement dans le cas d'une scène 3D dynamique. Ce travail a été effectué par Hedlena Bezerra dans le cadre de sa thèse en collaboration avec Elmar Eiseman et Xavier Décoret tous deux membres de l'équipe. L'idée ici est de grouper les objets 3D selon un critère de similarité choisi par l'utilisateur comme une combinaison des informations disponibles dans scène (voir figure 3.3). Le fait que la scène soit dynamique imposait de plus une contrainte temps-réel. Hedlena et Elmar ont proposé un algorithme de regroupement basé sur un mean shift 3D d'un ensemble de points représentatifs de la scène. Le mean-shift étant un algorithme qui peut travailler en n'importe quelle dimension, le regroupement peut s'effectuer sur un ensemble d'informations choisies par l'utilisateur. Par exemple on peut vouloir grouper en fonction de la position des points en projection perspective et de leur couleur. Une extension du mean-shit a aussi été proposée pour ajouter des paramètres accessibles à l'utilisateur lui permettant de préciser sa stratégie de regroupement ; typiquement en donnant une importance différente selon les objets de la scène. Ces travaux ont été publiés à NPAR 2008 [BEDT08].

3.2 Détail et modèle de la vision humaine

Une autre façon de voir le problème est de considérer que le détail est ce qui se perçoit moins bien. Divers modèles et théories du fonctionnement du système visuel humain peuvent ainsi nous donner des critères de définition du détail.

En particulier nous avons utilisé la théorie des espaces d'échelles (*scale space*) pour proposer un algorithme de manipulation de photographies. Ce travail a été effectué par Alexandrina Orzan et Adrien Bousseau dans le cadre de leur thèse en collaboration avec Pascal Barla (IPARLA - INRIA) [OBBT07b, OBBT07a]. Les espaces d'échelles gaussiens offrent un modèle de la perception humaine bas-niveau qui a de solides fondations mathématiques. Intuitivement, ce modèle consiste en une pile d'images convoluées par un filtre gaussien de noyau de plus en



Divers niveaux de détails construits à partir de l'image de gauche.

Combinaion de divers niveaux au sein d'une même image (ici l'abeille est choisie comme importante)

Stylisations appuyées sur les niveaux de détail obtenus

FIG. 3.4 – Utilisation des espaces d'échelle pour détecter les éléments importants d'une image.

plus large. L'image d'origine, d'échelle minimum, perd ainsi progressivement du détail au fur et à mesure que l'on monte dans les échelles. Pour cette raison, les arêtes de l'image disparaissent aussi progressivement et la théorie des espaces d'échelle garantit qu'une même arête ne peut réapparaître dans les échelles supérieures. La perception humaine nous dit alors que plus une arête persiste dans l'espace d'échelle, plus elle est importante. Ce critère nous permet ainsi de savoir quelles arêtes sont du détail à une échelle donnée. En s'appuyant sur cette théorie Alexandrina et Adrien ont pu proposer une mesure de l'importance des arêtes d'une image. Ils ont montré diverses applications selon que l'on enlève les arêtes de faible importance, que l'on les stylise ou que l'on les déforme (voir figure 3.4).

3.3 Ce qui est important n'est pas du détail

Il est parfois plus facile de savoir ce qui est important que de définir ce qui est du détail. En effet, sous cette forme là, la question revient à chercher quels sont les indices visuels pertinents pour représenter l'information de départ. Nous avons travaillé sur cette question en variant les applications et les angles d'approche. C'est un domaine difficile car les connaissances en sciences cognitives et modèles de la perception haut niveau sont encore très partielles. Les travaux présentés dans cette section sont ainsi moins aboutis que mes autres travaux mais ils me semblent néanmoins prometteurs et fondamentaux si l'on veut comprendre comment représenter au mieux une information visuelle.

3.3.1 Perception de la couleur

Nous avons travaillé sur la conversion d'images couleur en niveau de gris (voir figure 3.5). Ce travail a été effectué par Kaleigh Smith (MPI Informatik, Saarbrücken) durant un stage ef-

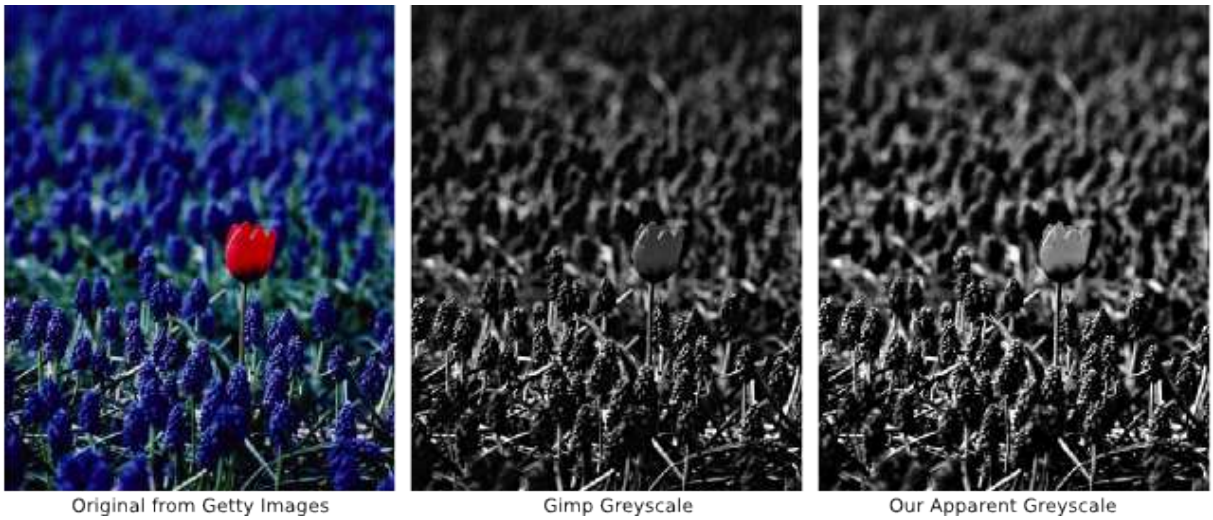


FIG. 3.5 – Conversion d'une image couleur (gauche) en niveau de gris (droite) et comparaison avec la version de gimp (milieu).

fectué sous ma direction et Pierre-Edouard Landes (membre de l'équipe) en collaboration avec Karol Myszkowski (MPI Informatik, Saarbrücken) et publié à Eurgraphics 2008 [SLTM08]. Lorsque l'on convertit une image couleur en niveau de gris on perd de l'information. Pour obtenir une bonne conversion la question est alors de savoir comment limiter la perte et réintroduire l'information perdue. Pour cela il faut définir quelle est l'information importante. Kaleigh et Pierre-Edouard ont proposé une approche en deux temps. Tout d'abord, en se basant sur un modèle validé par des expériences en perception, ils ont proposé une conversion de l'information de couleur vers un niveau de gris qui minimise la perte. En particulier ce modèle prend en compte l'effet Helmholtz-Kohlrausch qui permet de différencier des couleurs iso-luminantes. Dans un second temps ils ont proposé un algorithme pour réintroduire l'information importante : le contraste au niveau des discontinuités dans l'image pour l'information chromatique. Cette deuxième étape est contrôlable par l'utilisateur qui peut ainsi fixer son propre compromis entre fidélité à l'original et discrimination de l'information. Kaleigh a implémenté cette méthode sous forme d'un plugin Gimp disponible en ligne.

3.3.2 Perception de la forme

Dans le cas du dessin au trait, nous nous sommes penchés sur le problème de la représentation de la forme. La question est ici de déterminer quelles sont les lignes qui décrivent au mieux la forme d'un objet. De nombreuses équipes travaillent sur cette question et toute une série de type de lignes ont été proposées dans la littérature : contours, crêtes et vallées, contours suggestifs, crêtes dans la direction du point de vue, etc. Les expérimentations montrent qu'il n'y a pas un seul type de ligne qui donne une représentation visuelle correcte de la forme (variant entre trop de lignes ou pas assez). Il est donc probable qu'il faille trouver soit une combinaison de ces différents types, soit une nouvelle définition du problème. Nous avons travaillé avec l'équipe IPARLA (INRIA Bordeaux) et le laboratoire de science cognitive de Bordeaux sur cette question dans le cadre de l'ARC MIRO. Gwenola Thomas (IPARLA) et moi-même avons encadré

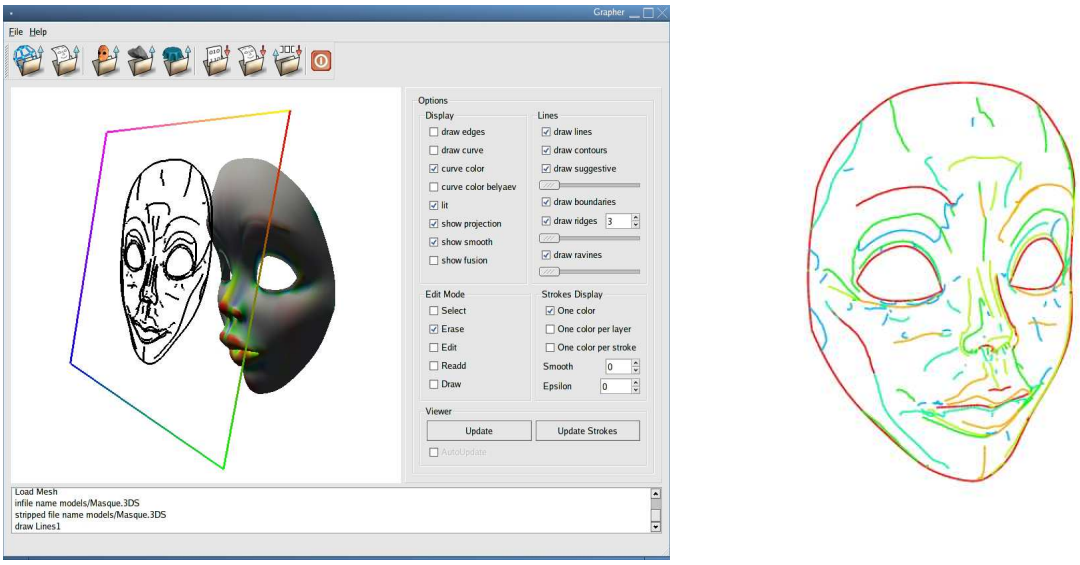


FIG. 3.6 – Expérience utilisateur pour qualifier les lignes caractéritiques d'un dessin. L'interface est montrée à gauche et le résultat cumulé de plusieurs sujets est montré à droite : le taux de sélection varie du vert (aucun utilisateur) au rouge (tous les utilisateurs).

deux étudiants de master, David Vanderhaeghe à Grenoble et Audrey Legeai à Bordeaux, sur deux approches orthogonales du problème.

David a proposé une plateforme interactive pour permettre la mise en oeuvre d'une étude utilisateur visant à déterminer quelles lignes conserve un utilisateur pour représenter un objet 3D (voir figure 3.6). Le système présentait à l'utilisateur un modèle 3D sous une vue fixée et rendu de manière classique (Phong shading). L'utilisateur pouvait afficher un ensemble de lignes calculées par le système (contours, crêtes et vallées et contours suggestifs). Il devait ensuite effacer les lignes lui semblant superflues jusqu'à obtenir un dessin le satisfaisant. Les résultats obtenus ont montré que les contours étaient indispensables pour représenter correctement la forme. En revanche ils n'ont pas permis de conclure sur une comparaison ou combinaison entre les crête et vallées et les contours suggestifs. Ce résultat est en partie du au fait que ces deux types de lignes vivent dans les lieux de courbure maximale et sont donc souvent redondants. Dans ces cas-là il est impossible de savoir quelle ligne particulière l'utilisateur a voulu conserver.

Audrey a proposé une approche pour sélectionner automatiquement les lignes pertinentes parmi le même ensemble de ligne que David. Elle s'est appuyée pour cela sur un modèle de la vision humaine qui définit les lieux de discontinuité perçus dans une image en niveau de gris. Ce modèle s'appelle une carte de brillance et est basé sur le fonctionnement des cellules situées sur la rétine qui mesurent des différences de luminosité entre le centre et le pourtour de la cellule à différentes échelles. Grâce à cette carte de brillance calculée sur une vue d'un modèle 3D, on peut seuiller automatiquement les divers algorithmes de calcul de lignes (voir figure 3.7). Audrey a étendu cette approche a des vues multiples permettant ainsi de tourner autour d'un objet 3D rendu au dessin au trait.

Ces travaux sont pour moi un premier pas mais de nombreuses pistes restent à explorer. La compréhension du mécanisme permettant au cerveau humain d'interpréter facilement un dessin est un champ de recherche actif des sciences cognitives et plusieurs équipes dans le monde

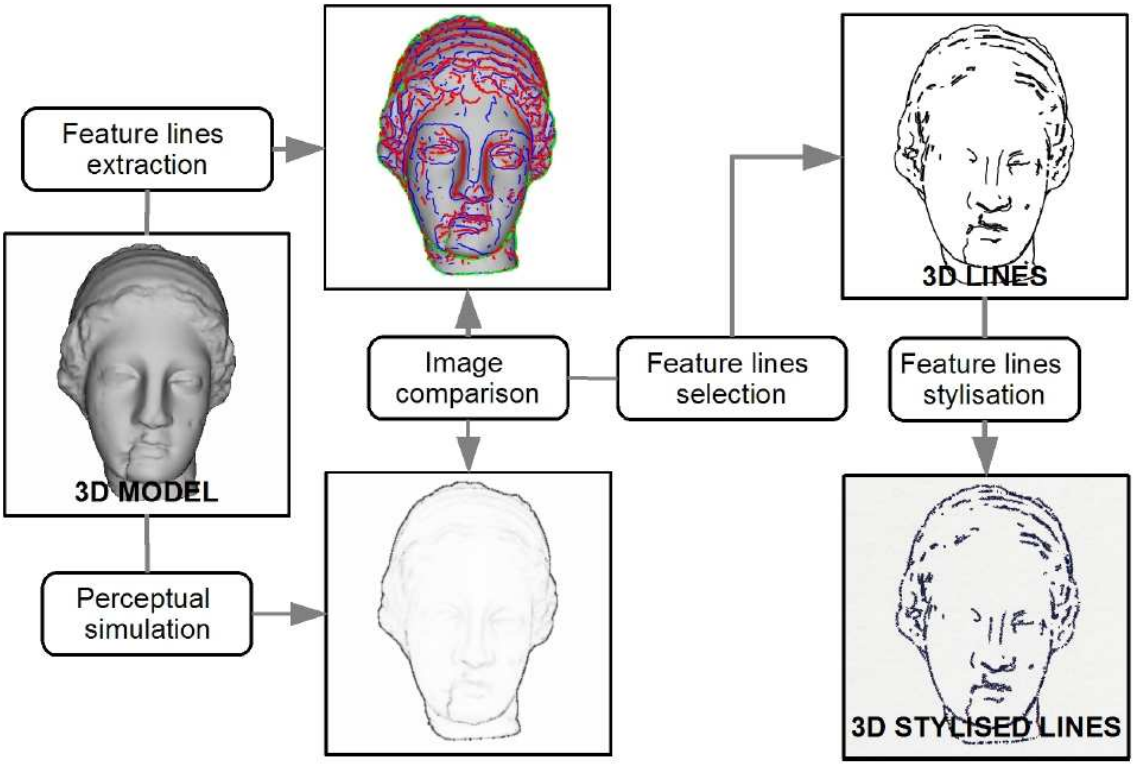


FIG. 3.7 – Pipeline d'extraction de lignes caractéristiques.

travaillent sur cette question du point de vue informatique.

3.3.3 Perception des matériaux

Enfin, nous nous sommes intéressés à la perception des matériaux. Ce sujet a été étudié en vision humaine et appliqué coté informatique à la reconstruction d'objets à partir des déformations de sa texture. Mais peu de travaux se sont posés la question de la stylisation des matériaux. La question est ici de savoir quels sont les indices visuels qui permettent de reconnaître tel ou tel type de matériaux. C'est une question complexe car un matériau est défini par un ensemble de propriétés qui vont de sa réaction à l'éclairage (est-il diffus ou brillant), sa surface (est-il plus ou moins rugueux) et ses motifs (veines, craquelures, etc). C'est cet ensemble d'indices qui font que l'on reconnaît un matériau dans une image.

Nous avons travaillé sur le cas de l'illustration par hachures et pointillage (*stippling*). Cette recherche a été menée par Kaleigh Smith (MPI Informatik, Saarbrücken) durant un stage effectué sous ma direction en collaboration avec Pascal Barla et Cyril Soler (membres de l'équipe) et Thomas Luft et Oliver Deussen (université de Constance). Nous avons défini les éléments caractéristiques d'un matériau en nous fondant sur les travaux existants en vision humaine puis pour chacun de ces indices nous avons proposé des solutions pour les extraire automatiquement d'un modèle 3D muni d'une fonction de texture bidirectionnelle (*BTF ou bidirectional texture function*) et les représenter visuellement dans le style que nous visions (voir figure 3.8). Nous avons ensuite mené une étude utilisateur pour valider cette approche en demandant d'ordonner

FIG. 3.8 – Représentation de matériaux dans un style illustration. En haut les textures sont représentées, en bas la propriété de “brillance” est illustrée.

des séries de sphères texturées réalistes et illustrées selon les éléments caractéristiques que nous avions choisis.

Les résultats obtenus montrent que notre approche donne des résultats comparables au cas réaliste mais nous a aussi montré que l’humain n’était pas très efficace pour faire ce type de classement. Il s’avère ainsi que notre perception des matériaux à partir d’une image est en fait peu précise quelle que soit le type de visualisation. D’autre part les résultats visuels obtenus n’atteignent pas la qualité des algorithmes semi-automatiques (où c’est l’utilisateur qui choisit lui-même sa texture d’illustration). Nous continuons donc à travailler sur ce sujet afin d’obtenir une représentation automatique de qualité équivalente aux méthodes semi-automatiques disponibles actuellement.

Chapitre 4

Cohérence temporelle

La stylisation est un procédé qui vise à représenter une scène en s'éloignant plus ou moins de la donnée photométrique de départ (ce que l'on verrait si l'on prenait une photo de la scène). Dès lors que l'on veut produire une animation stylisée à partir d'une animation donnée sous forme d'une vidéo ou d'une animation 3D se pose la question des artefacts visuels engendrés par cette stylisation. Ces artefacts sont de plusieurs types mais trouvent tous leur origine dans le fait que la stylisation d'une animation doit satisfaire des contraintes intrinsèquement contradictoires en terme de mouvement et caractéristiques 2D.

4.1 Quel est le problème ?

Les caractéristiques du médium utilisé pour styliser la scène sont définies en 2D (ce peut être une fréquence, une distribution, une taille de coups de pinceaux, une épaisseur de trait, etc.) mais ce médium doit évoluer en accord avec la scène représentée. Or le mouvement 3D de la scène crée un mouvement 2D qui, si le médium le suit simplement, va induire des trous ou des concentrations de médium dans l'image. Il est donc impossible du point de vue théorique de satisfaire les deux contraintes : conserver des caractéristiques 2D fixes et suivre le mouvement 3D tout au long de l'animation. Pour pallier à ça, on va alors proposer des algorithmes qui approximent ces deux contraintes au prix de certains artefacts : écart par rapport au mouvement, ajout/suppression de médium qui peut provoquer des discontinuités dans l'animation, écart par rapport aux caractéristiques du médium qui peut provoquer du flou ou des oscillations plus ou moins continues, induction de mouvements fantômes (qui n'existent pas dans l'animation de départ), etc.

A cela peuvent s'ajouter des contraintes supplémentaires. On peut vouloir que les caractéristiques du médium varient par objet de la scène ou par zone dans l'image. C'est le cas par exemple pour les hachures où la densité de hachure doit varier en fonction du ton (donc dans l'espace image) mais l'orientation des hachures peut varier en fonction de la courbure des objets (donc dans l'espace objet). Ces variations peuvent induire des discontinuités temporelles supplémentaires qu'il faut lisser au cours de l'animation.

Enfin, le mouvement que le médium est supposé suivre n'est pas nécessairement le mouvement 3D des objets. Par exemple si une scène est fixe mais l'éclairage dynamique, on peut vouloir que le médium suive le mouvement d'un reflet spéculaire sur un objet plutôt que rester

fixe sur l'objet en changeant uniquement ses attributs (couleur, épaisseur par exemple). Il n'est donc pas évident de définir la métaphore de mouvement adaptée au but recherché par l'auteur de l'animation.

Ainsi nous avons proposé plusieurs approches pour améliorer la cohérence temporelle de la stylisation qui chacune offre un compromis différent entre les divers artefacts. Je les décris ci-dessous et je terminerai par un bilan sur ces méthodes.

4.2 Papier dynamique

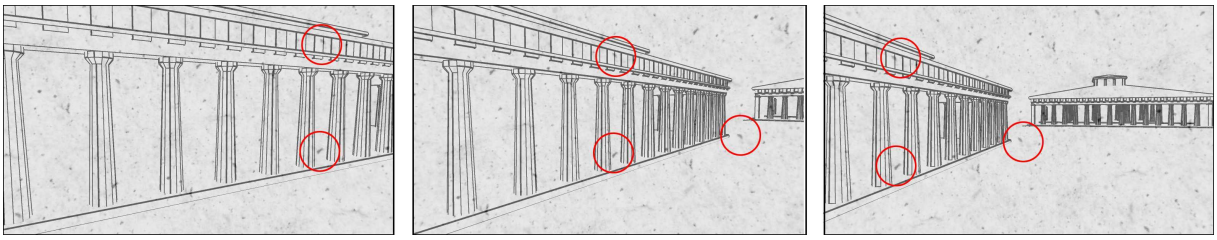


FIG. 4.1 – Papier dynamique : le grain du papier suit les traits du dessin au cours de la promenade.

Le premier travail que nous avons effectué sur ce sujet a porté sur le cas du papier sur lequel l'image est supposée être dessinée. Ce travail a été effectué par Matthieu Cunzi lors de son stage de master en collaboration avec Sylvain Paris, Gilles Debunne et Jean-Dominique Gascuel (membres de l'équipe) et Frédo Durand (MIT). Nous avons proposé une méthode permettant d'animer le papier en accord avec les mouvements de la caméra (voir figure 4.1). Ce choix de métaphore correspond aux promenades virtuelles sur un terrain plat (ville, site archéologique). Le mouvement du promeneur est décomposé en deux parties : avancer/reculer, tourner la tête. Pour ces deux mouvements nous avons proposé une métaphore d'évolution du papier. Lorsque le promeneur avance ou recule le papier subit un zoom. Afin de conserver des caractéristiques constantes en espace image nous avons proposé un algorithme de zoom infini en composant diverses échelles du même papier. Lorsque le promeneur bouge la tête, le papier se déplace dans le sens opposé. Nous avons montré que son mouvement est similaire à celui d'une sphère englobant la caméra qui roule sans glisser sur le plan du papier.

Ces deux métaphores permettent une évolution cohérente du papier tout au long de la promenade et offrent ainsi l'illusion d'avoir à chaque instant une scène dessinée sur un papier. Le dessin se fait simplement en compositant le rendu de la scène avec l'image de papier. Cette méthode a ainsi été utilisée par un élève de Philippe Chaubaroux à SupCréa (école de réalisateur 3D) pour son projet de fin d'année en compositant le papier dynamique à un rendu obtenu avec 3DStudio et a été publiée à Graphics Interface 2003 [CTP+03].

Cette méthode présente bien sur des artefacts. Tout d'abord, le zoom infini étant produit par le mélange de plusieurs échelles d'une même texture, le résultat visuel n'est pas exactement équivalent à la texture de départ. Ce n'est pas perceptible dans le cas de motifs à hautes fréquences comme pour le grain du papier mais devient visible pour des motifs plus structurés. Le mélange va alors créer des motifs (des fréquences et des couleurs) qui n'étaient pas présents dans la texture de départ.

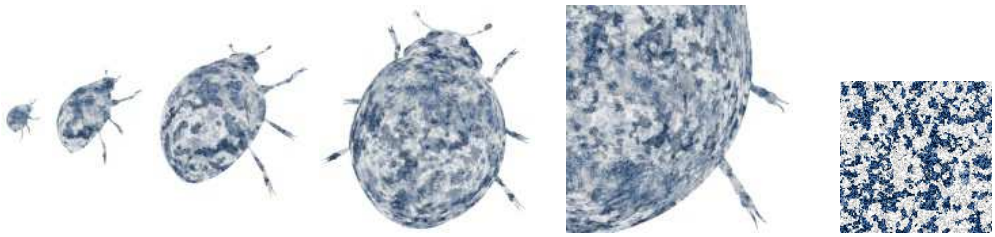


FIG. 4.2 – Zoom infini tridimensionnel (original 2D à droite).

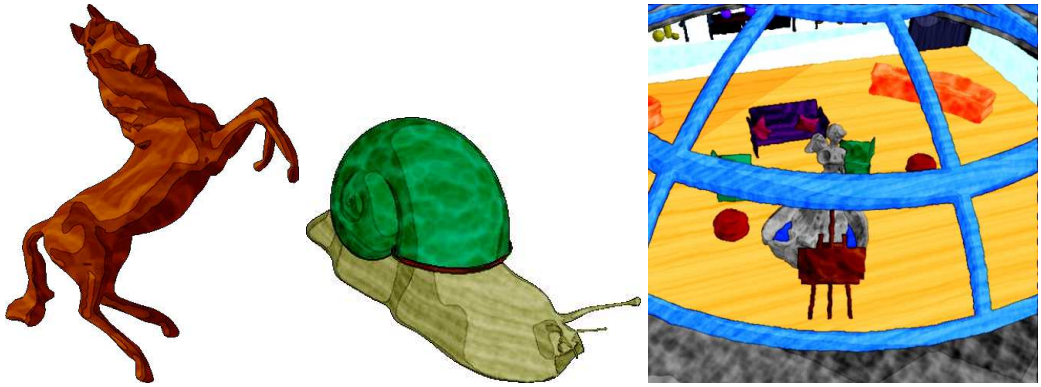


FIG. 4.3 – Rendu aquarelle avec textures volumiques.

D'autre part, le mouvement 3D réel de la scène lors d'un déplacement n'est réductible à un zoom uniforme. Il y a donc un écart entre le mouvement du papier et celui de la scène en fonction de la profondeur pour laquelle le zoom a été calculé. Cela se perçoit comme un glissement de certaines parties de la scène sous le papier. Pour supprimer cet artefact nous avons proposé de considérer une texture 3D au lieu d'une texture 2D. Pierre Bénard dans le cadre de son master et Adrien Bousseau dans le cadre de sa thèse ont travaillé sur une extension du zoom infini aux textures volumiques. L'intérêt est de pouvoir associer une texture par objet qui va, par le même mécanisme de mélange, maintenir une apparence fixe en 2D lorsque l'objet se rapproche ou s'éloigne de la caméra (voir figure 4.2). Une première version de ce travail a été implémentée dans un plugin de rendu aquarelle développé par David Lanier (société dl3D) dans le cadre d'un contrat de recherche avec Studio Brocéliande (voir figure **??**). En revanche les limitations liées au mélange persistent et Pierre va étudier, dans le cadre de sa thèse, des alternatives pour permettre de prendre en compte une plus large gamme de textures.

4.3 Advection de texture

Toujours dans le cas de textures peu structurées nous avons proposé une méthode alternative pour la stylisation de vidéo. Ce travail a été effectué par Adrien Bousseau dans le cadre de sa thèse à l'occasion d'un séjour chez Adobe à Seattle en collaboration avec David Salesin (Adobe) et Fabrice Neyret (membre de l'équipe). Le but était d'obtenir un style aquarelle à partir d'une vidéo. Dans ce cas la seule information de mouvement dont nous disposons est le flot optique.

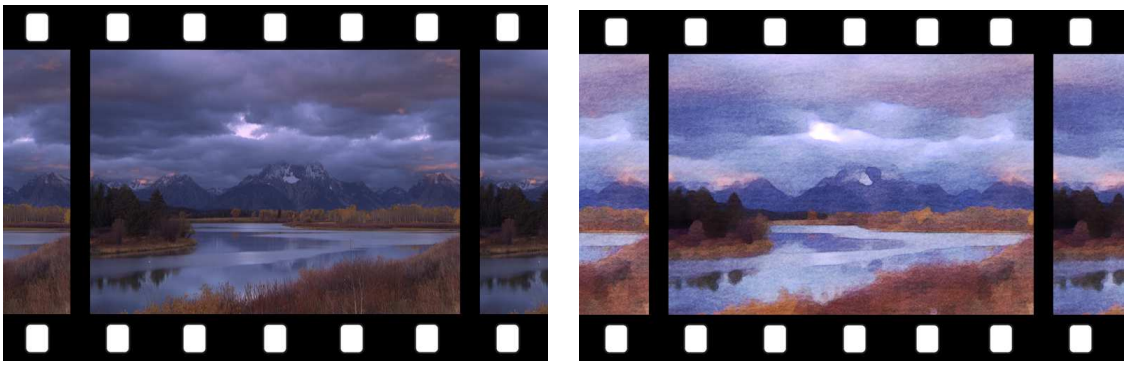


FIG. 4.4 – Stylisation de vidéo dans un style aquarelle.

Les méthodes décrites à la section précédente sont donc plus applicables. Adrien a alors proposé de faire évoluer une texture aquarelle couvrant le plan image en accord avec le flot optique. Cette évolution correspond à de l'advection de texture selon un champ de vecteur. Il est bien évident que la texture se déforme au fur et à mesure que la vidéo avance et perd en quelques dizaines de frames ses caractéristiques de départ. Adrien a proposé une méthode permettant de limiter ces artefacts en revenant périodiquement à la texture d'origine et en combinant plusieurs advections : dans le sens de lecture et dans le sens inverse ainsi qu'avec des cycles différents. Pour chaque pixel, sa couleur sera prise dans la texture advectée la moins déformée autour de ce pixel.

Cette méthode produit des résultats très convainquants pour des textures peu structurées au prix d'une légère baisse de netteté due au fait que les couleurs des pixels sont tirées de diverses textures. Cette perte de netteté est cependant moins notable que dans le cas du papier dynamique où l'on effectuait par pixel un mélange de plusieurs images. Cette technique a été implémentée par Adrien comme un plugin After Effect et validée par un court métrage réalisé par Laurence Boissieux. Elle a fait l'objet d'une publication à Siggraph 2007 [BNTS07].

Cette approche ne convient par contre pas aux textures structurées pour lesquelles la déformation du motif est visible après quelques frames.

4.4 Distribution de points

David Vanderhaeghe a travaillé dans le cadre de sa thèse sur la cohérence d'une distribution de points. Ce travail a été mené en collaboration avec Pascal Barla et François Sillion (membres de l'équipe). Comme décrit au chapitre 2, un modèle de simulation de médium est le rendu à base de marques. Le principe est d'attacher des marques à des points distribués dans la scène. Les points évoluent selon le mouvement de la scène et les marques sont dessinées en 2D centrées sur la projection 2D des points. David a proposé un algorithme permettant de distribuer ces points de sorte à satisfaire les deux contraintes : avoir une distribution de type bruit bleu dans l'image et suivre un mouvement défini par l'animation (mouvement des objets pour une animation 3D ou flot optique pour une vidéo). Pour cela il propose de distribuer les points en espace image en assurant un critère de Poisson ; puis de les déplacer selon le mouvement demandé (soit en suivant directement le flot optique, soit en les projetant sur les objets de la scène pour les reprojeter en 2D à l'image suivante) ; et de corriger la distribution résultat en enlevant

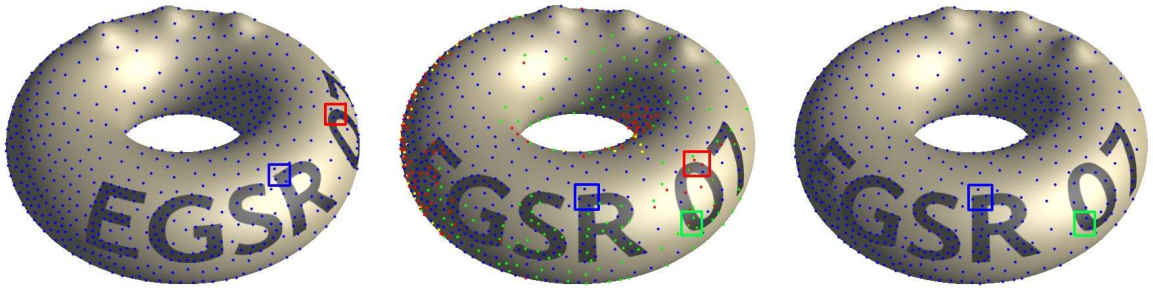


FIG. 4.5 – Distribution de points temporellement cohérente. (Gauche) Les points sont distribués en espace image. Ici on utilise la luminance pour guider la densité. Les points sont ensuite projetés sur l'objet 3D. (Milieu) L'objet est déplacé et les points reprojetés en espace image. Les points rouges sont supprimés, les points bleus sont conservés et les points verts sont ajoutés. Les points jaunes ne eux sont plus visibles. (Droite) Résultat finale de la distribution.

et ajoutant des points en 2D de manière à conserver le critère de Poisson (voir figure 4.5).

Cette approche s'intègre parfaitement dans les systèmes existants de rendu à base de marque et assure un suivi exact du mouvement. La contrepartie est l'apparition et disparition des points qui peuvent créer des discontinuités dans l'animation (*popping*). Cependant les points étant suivis au cours de l'animation, on peut lisser ces événements dans le temps lors du rendu final en faisant évoluer les marques continûment, soit par transparence, soit par grossissement.

Cette méthode a été validée par le court métrage Dumb Luck [MBV+07] réalisé par Florent Moulin et Laurence Boissieux et publisée à EGSR 2007 citeVBTS07a.

Contrairement aux deux méthodes décrites précédemment, cette approche va être très bien adaptées aux motifs contenant des éléments reconnaissables individuellement : points, hachures, coups de pinceaux. Elle est en revanche peu efficace pour des textures de type papier ou aquarelle pour lesquelles il faudrait autant de points que de grains de papier ou marques de pigments.

4.5 Motifs 2D dynamiques

Nous avons vu que nous pouvions proposer des approches pour améliorer la cohérence temporelle dans le cas de motifs non structurées, que ce soit des textures à hautes fréquences (papier, aquarelle) ou des textures possédant des éléments distincts distribués selon un bruit bleu (peinture, pointillage). Nous avons travaillé sur une dernière méthode visant les motifs structurés (croisillons, trames, ...). Ce travail a été effectué par Pascal Barla dans le cadre de sa thèse et en collaboration avec Simon Breslaw et Lee Markosian (university of Michigan). Le but est cette fois de faire évoluer un motif en 2D qui suit le mouvement la scène sans perdre sa structure (voir figure 4.6). Cette contrainte forte interdit les déformations non rigides du motif. Nous avons ainsi proposé une méthode qui approxime le mouvement de la scène 3D par une similitude 2D qui est alors appliquée au motif. Cette transformation du motif ne permet pas de suivre exactement le mouvement de la scène sous-jacente et nous laissons le choix à l'utilisateur de placer plusieurs morceaux de motifs sur la scène (usuellement un par objet) pour améliorer l'approximation. Cette approche a été publiée à Siggraph 2007 [BSM+07].

Cette solution respecte la structure du motif au prix d'un écart avec le mouvement à suivre. Une fois de plus, cela se perçoit comme un glissement du motif sur l'objet. En revanche cette

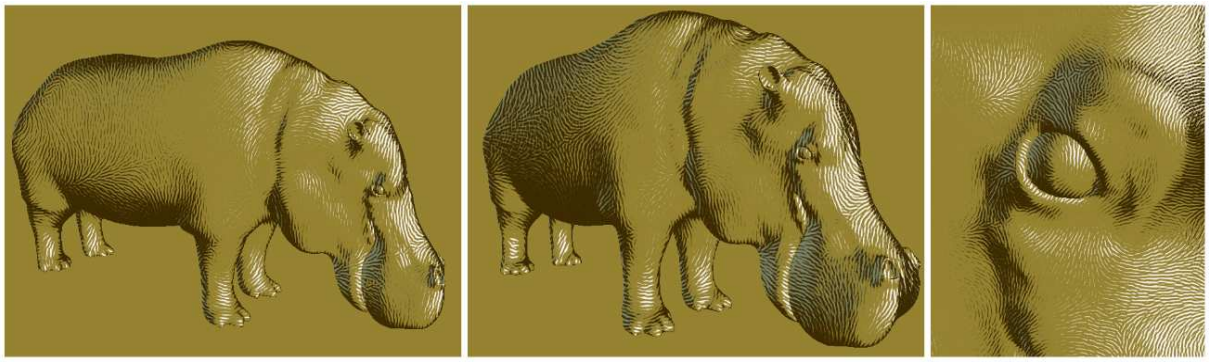


FIG. 4.6 – Motif 2D dynamique : le motif se déforme rigidement pour approximer le mouvement 3d de l'objet.

solution est extrêmement simple à mettre en oeuvre et fonctionne ainsi en temps réel.

4.6 Bilan

Nous venons de voir un ensemble de solutions au problème de la stylisation d'animations. Chacune propose un compromis différent entre toutes les contraintes induites par la stylisation.

Représentation du médium : Selon les approches, une fidélité plus ou moins grande au médium choisi va être obtenue. L'écart avec le motif visé va s'exprimer en terme de variation de fréquence, de déformation ou de perte de structure. Chaque méthode va ainsi être adaptée à un type de médium particulier selon le type d'écart effectué. Par exemple, une perte de structure est invisble sur un motif totalement irrégulier. De nombreuses classifications ont été proposées pour définir les types de textures. Ici nous en distinguons trois : les matières granuleuses ou quasi stochastiques (papier, pigments d'aquarelles), les motifs irréguliers où chaque élément est perçu individuellement (coups de pinceaux) et les motifs réguliers ou quasi réguliers qui ont une structure fixe (trames, croisillons).

Représentation du mouvement 3D : Selon les méthodes le mouvement 3D représenté va être plus ou moins exact. L'écart entre le mouvement 3D et celui représenté crée une impression visuelle de glissement du médium sur la scène. Le cas extrême est appelé "rideau de douche" : quand le médium est fixe en espace image, la scène semble glisser derrière une plaque de verre texturée.

Continuité temporelle : C'est un des problèmes principaux que l'on entend par cohérence temporelle. Diverses types de discontinuités peuvent être crées selon le type d'approche. Le cas le plus évident est l'apparition brutale d'un élément de dessin (*popping*) mais des discontinuités plus subtiles peuvent apparaitres comme des variations de netteté ou de contraste (typiquement liées au mélange de plusieurs textures).

Le tableau suivant résume les propriétés de chacune des méthodes.

	Médium	Mouvement 3D	Continuité	Problème
Papier 2D dynamique	Granuleux	Approché zoom global	OK	Mélange d'octaves
Texture 3D dynamique	Granuleux	Exact par sommet	OK	Mélange d'octaves Déformation perspective
Advection de texture	Granuleux	Exact par pixel	Variation de netteté	Besoin de l'animation complète
Distribution de points	Irrégulier	Exact par point d'ancrage	*Poppings*	Gestion des apparitions/disparitions
Motif 2D dynamique	Tous	Approché transformation rigide	OK	Glissements

Chapitre 5

Conclusion

Le domaine du rendu expressif est très jeune. Il reste par conséquent de nombreuses questions ouvertes. Pour ma part, quatre points fondamentaux vont très probablement occuper mes prochaines années de recherche.

Ma première ambition est d'avancer sur la compréhension de ce qui constitue le sens d'une image. Pour cela il me parait nécessaire d'aller plus loin dans la description des indices visuels qui permettent à l'humain de comprendre ce qui est représenté dans une image (dessin ou photographie). Cela peut conduire à des problèmes d'informatique graphique très concrets comme la représentation efficace de la forme ou des matériaux présentés section 3.3. Cela peut aussi donner lieu à des recherches sur la décomposition d'images en ses principaux constituants (couleur, texture, ombres, contours, ...). Dans tous les cas, la perception humaine doit rester un des guides majeurs pour ce type des travaux.

Un deuxième champs de recherche qui reste à approfondir est la question de la cohérence temporelle lors de la stylisation d'animations. Ce mémoire présente plusieurs approches mais la validation des résultats et la formalisation du problème reste incomplète. Il est impossible de comparer les résultats obtenus avec une référence réelle puisqu'il est impossible de faire une animation cohérente à la main. En revanche nous devons être capable de mesurer et qualifier les artefacts visuels créés par les différentes approches possibles. C'est en tout cas l'un des buts que je me fixe à moyen terme.

A plus long terme, une autre question, probablement plus difficile, me semble intéressante. C'est la question du style en tant quel. Si nous étions capables de définir des modèles de styles suffisamment généraux, nous pourrions alors aller beaucoup plus loin qu'aujourd'hui dans les méthodes de dessin par l'exemple. Ce champ de recherche a de très nombreuses applications dans la production audiovisuelle mais la notion de style étant extrêmement complexe il me parait raisonnable d'avancer progressivement sur les cas particuliers (comme nous avons commencé à le faire sur les motifs ou la peinture) avant d'espérer obtenir des modèles génériques.

Enfin, le thème de l'abstraction reste un des fondements de la stylisation puisque le processus même de la production d'une représentation visuelle est la production d'une abstraction de la scène représentée. Ce terme est un mot qui couvre de nombreux sens. Nous n'avons pour l'instant approché l'abstraction que sous l'angle de la simplification. Il me paraîtrait intéressant d'ouvrir nos approches dans la direction de la généralisation ou factorisation. Ce sens-là du mot abstraction pose la question de ce qui est commun aux diverses instances d'un même objet. Lors de la production d'un dessin, c'est souvent les caractéristiques communes qui font qu'un

objet est reconnaissable. Il me semble que cette vision des choses pourrait ouvrir la voie à de nouveaux algorithmes, typiquement pour le dessin au trait. Cette question rejoint bien sur la question des indices visuels pertinents pour représenter un objet donné.

Liste complète de mes publications

[BBMT06] Pascal Barla, Simon Breslav, Lee Markosian, and Joëlle Thollot. Interactive hatching and stippling by example. Technical Report inria-00255958, INRIA, 2006.

[BBT+06] Pascal Barla, Simon Breslav, Joëlle Thollot, François Sillion, and Lee Markosian. Stroke pattern analysis and synthesis. *Computer Graphics Forum (Proc. of Eurographics 2006)*, 25(3) :663–671, 2006.

[BEDT08] Hedlena Bezerra, Elmar Eisenman, Xavier Décoret, and Joëlle Thollot. 3d dynamic grouping for guided stylization. In *International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, pages 89–95, jun 2008.

[BKTS06] Adrien Bousseau, Matthew Kaplan, Joëlle Thollot, and François Sillion. Interactive watercolor rendering with temporal coherence and abstraction. In *International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, pages 141–149. ACM, 2006.

[BNTS07] Adrien Bousseau, Fabrice Neyret, Joëlle Thollot, and David Salesin. Video watercolorization using bidirectional texture advection. *ACM Transaction on Graphics (Proceedings of SIGGRAPH 2007)*, 26(3) :104.1–104.7, 2007.

[BSM+07] Simon Breslav, Karol Szerszen, Lee Markosian, Pascal Barla, and Joëlle Thollot. Dynamic 2d patterns for shading 3d scenes. *ACM Transaction on Graphics (Proceedings of SIGGRAPH 2007)*, 26(3) :20.1–20.5, 2007.

[BTM06] Pascal Barla, Joëlle Thollot, and Lee Markosian. X-toon : An extended toon shader. In *International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, pages 127–132. ACM, 2006.

[BTS04] Pascal Barla, Joëlle Thollot, and François Sillion. Simplification et abstraction de dessins au trait. In *AFIG '04 (Actes des 17èmes journées de l'AFIG)*, Nov 2004.

[BTS05a] Pascal Barla, Joëlle Thollot, and François Sillion. Geometric clustering for line drawing simplification. In *Siggraph technical sketch : SIGGRAPH'2005*. ACM, 2005.

[BTS05b] Pascal Barla, Joëlle Thollot, and François Sillion. Geometric clustering for line drawing simplification. In *Proceedings of the Eurographics Symposium on Rendering*, pages 183–192, 2005.

[BTT07] Pascal Barla, Joëlle Thollot, and Gwenola Thomas. Rendu expressif. In D. Bechmann B. Péroche, editor, *Informatique graphique et rendu*, Traitement du signal et de l'image, chapter 11. Hermes - Lavoisier, 2007.

[CHNT00] Marie-Paule Cani, Stefanie Hahmann, Fabrice Neyret, and Joëlle Thollot, editors. *Actes des Journées Française d'Informatique Graphique*, 2000.

[CHT01] Marie-Paule Cani, Stefanie Hahmann, and Joëlle Thollot. Numéro spécial de la revue de cfao et d'infographie : Mondes virtuels représentation et simulation, 2001.

[CTP+03] Matthieu Cunzi, Joëlle Thollot, Sylvain Paris, Gilles Debunne, Jean-Dominique Gascuel, and Frédo Durand. Dynamic canvas for immersive non-photorealistic walkthroughs. In *Proc. Graphics Interface*, pages 121–130. A K Peters, LTD., june 2003.

[DDTP00] Frédo Durand, George Drettakis, Joëlle Thollot, and Claude Puech. Conservative visibility preprocessing using extended projections. In Kurt Akeley, editor, *27th annual conference on Computer graphics and interactive techniques, SIGGRAPH 2000, July, 2000*, Annual Conference Series, pages 239–248, New Orleans, Louisiana, Etats-Unis, July 2000. ACM SIGGRAPH, ACM Press / /Addison-Wesley.

[HLTC03] Laure Heïgéas, Annie Luciani, Joëlle Thollot, and Nicolas Castagné. A physically-based particle model of emergent crowd behaviors. In *International Conference Graphicon 2003, September, 2003*, pages 1–9, Moscou, Russie, September 2003.

[LNHT04] Sylvain Lefebvre, Fabrice Neyret, Samuel Hornus, and Joëlle Thollot. Mobinet : a pedagogic platform for computer science, maths and physics (how to make students love maths by programming video games). In *Eurographics - Education*. Eurographics, august 2004. Grenoble.

[LTT06] Audrey Legeai, Gwenola Thomas, and Joëlle Thollot. Non-photorealistic line-based rendering : A perceptual approach npar poster session, june 2006.

[MBV+07] Florent Moulin, Laurence Boissieux, David Vanderhaeghe, Joëlle Thollot, and Pascal Barla. Dumb luck. Eurographics Animation Theatre, sep 2007.

[OBBT07a] Alexandrina Orzan, Adrien Bousseau, Pascal Barla, and Joëlle Thollot. Multi-scale shape manipulations of photographs - Siggraph poster. SIGGRAPH, 2007.

[OBBT07b] Alexandrina Orzan, Adrien Bousseau, Pascal Barla, and Joëlle Thollot. Structure-preserving manipulation of photographs. In *International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, pages 103–110, aug 2007.

[OBW+08] Alexandrina Orzan, Adrien Bousseau, Holger Winnemöller, Pascal Barla, Joëlle Thollot, and David Salesin. Diffusion curves : A vector representation for smooth-shaded images. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2008)*, 27(3) :92.1–92.8, 2008.

[RT03] Isabelle Ratinaud and Joëlle Thollot. Virtual immersion in the ancient greek city of argos. In *International Congress of Classical Archaeology*, 2003.

[SLTM08] Kaleigh Smith, Pierre-Edouard Landes, Joëlle Thollot, and Karol Myszkowski. Apparent greyscale : A simple and fast conversion to perceptually accurate images and video. *Computer Graphics Forum (Proceedings of Eurographics 2008)*, 27(2) :193–200, apr 2008. url : http ://www.mpi-inf.mpg.de/resources/ApparentGreyscale/.

[Tho96] Joëlle Thollot. *Extension du modèle IFS pour une géométrie fractale constructive*. PhD thesis, Université Claude Bernard Lyon 1, 1996.

[Tho97] Joëlle Thollot. Set manipulations of fractal objects using matrices of IFS. In Ehoud Ahronovitz and Christophe Fiorio, editors, *7th International Workshop on Discrete Geometry for Computer Imagery, DGCI'97, September, 1997*, volume 1347 of *Lecture Notes in Computer Sciences*, pages 223–234, Montpellier, France, 1997. Springer.

[Tho04] Joëlle Thollot. Numéro spécial de la revue i-mag sur la réalité virtuelle. Magazine de l'AAE ENSIMAG, 2004.

[TT93] Joëlle Thollot and Eric Tosan. Construction of fractales using formal languages and matrices of attractors. In Harold P. Santos, editor, *Compugraphics*, pages 74–81, 1993.

[TT95] Joëlle Thollot and Eric Tosan. Constructive fractal geometry : constructive approach to fractal modeling using language operations. In *Graphics Interface (GI'95) Proceedings*, 1995.

[TZTV97] Joëlle Thollot, Chems Eddine Zair, Eric Tosan, and Denis Vandorpe. Modeling fractal shapes using generalizations of ifs techniques. In Claude Tricot Jacques Lévy Véhel, Evelyne Lutton, editor, *Fractals in Engineering*. Springer, 1997.

[VBT+07] Romain Vergne, Adrien Bousseau, Joëlle Thollot, David Vanderhaeghe, Pascal Barla, and Xavier Granier. Utilisation du rendu expressif pour l'illustration et l'exploration de données archéologiques. In *Virtual Retrospect 2007 : Archéologie et Réalité Virtuelle*. Ausonius, 2007.

[VBTS06a] David Vanderhaeghe, Pascal Barla, Joëlle Thollot, and François Sillion. A dynamic drawing algorithm for interactive painterly rendering. In *Siggraph technical sketch : SIGGRAPH'2006*. ACM, aug 2006.

[VBTS06b] David Vanderhaeghe, Pascal Barla, Joëlle Thollot, and François Sillion. Dynamic painting of animated 3d scenes. NPAR Poster Session, jun 2006.

[VBTS06c] David Vanderhaeghe, Pascal Barla, Joëlle Thollot, and François Sillion. Un système interactif et controlable de peinture pour l'animation 3d. In *AFIG '06 (Actes des 19èmes journées de l'AFIG)*, nov 2006.

[VBTS07a] David Vanderhaeghe, Pascal Barla, Joëlle Thollot, and François Sillion. Dynamic point distribution for stroke-based rendering. In *Rendering Techniques 2007 (Proceedings of the Eurographics Symposium on Rendering)*, pages 139–146, 2007.

[VBTS07b] David Vanderhaeghe, Pascal Barla, Joëlle Thollot, and François Sillion. Rendu peinture, interactif et controlable, de scènes 3d. *Revue Electronique Francophone d'Informatique Graphique*, 1(1) :53–62, jan 2007.

Annexe A

Etat de l'art

Cette annexe reprend l'intégralité du chapitre sur le rendu expressif publié dans l'ouvrage "Informatique graphique et rendu" [BTT07] et écrit en collaboration avec Pascal Barla et Gwenola Thomas.

Chapitre 1

Rendu Expressif

1.1. Introduction

Le rendu expressif, aussi appelé non-photoréaliste (NPR en anglais) concerne l'ensemble des techniques de rendu qui ne s'intéressent pas à reproduire fidèlement les lois de la physique (i.e. matériaux réalistes et illumination globale). Cette définition reste vague et en pratique, la grande majorité des approches en rendu expressif imitent ou s'inspirent des média traditionnels : peinture à l'huile, dessin au trait, aquarelle, etc. Il serait cependant réducteur de considérer que le rendu expressif ne se limite qu'à simuler de tels média ; tout comme en peinture et dessin, il faut d'abord se poser la question de ce que l'image est sensée représenter. Pas seulement la nature du sujet, mais surtout les aspects qui doivent être mis en avant et ceux que l'on va préférer ignorer. Car après tout, c'est l'avantage des média traditionnels sur la photographie : pouvoir mettre en avant ou abstraire certains aspects.

Afin d'illustrer cela, on peut comparer deux types de représentations nettement différents : l'illustration scientifique et la peinture impressionniste. Avec le premier, c'est la forme de l'objet qui prime sur tout le reste. Souvent, l'artiste doit même supprimer certains défauts de l'objet pour représenter non pas le spécimen qui est sous ses yeux, mais un prototype qui représente une catégorie d'objets. Tandis qu'avec le second, c'est l'image rétinienne qui est mise en avant, la distribution des couleurs en particulier. Ici, l'artiste se concentre plus sur la retranscription de l'impression suscitée par la scène représentée que sur la forme précise des objets qui la composent (d'où le terme "impressionnisme").

Chapitre rédigé par Pascal BARLA, Joëlle THOLLOT et Gwenola THOMAS.

Le médium utilisé n'intervient qu'ensuite, mais il a toutefois un intérêt crucial : il impose des contraintes de par ses limitations, et peut aussi être choisi pour certaines propriétés qui permettent de mieux atteindre le but de la représentation. Par exemple, le dessin au trait ne peut représenter qu'une gamme de tons limitée (grace à des groupes de hachures par exemple) mais illustre clairement la forme des objets par une économie de moyens (quelques contours), et de ce fait convient parfaitement à de nombreuses illustrations scientifiques où la forme doit être mise en avant au détriment du matériau. Parfois, le choix du médium est aussi lié à des problèmes de reproduction ; pour reprendre l'exemple précédent, le dessin au trait a cet avantage qu'il est bien adapté à l'impression noir et blanc.

Le rendu expressif en informatique graphique a de nombreux points communs avec les techniques traditionelles ; il partage les mêmes problématiques de représentation, indépendantes du médium utilisé : quels sont les aspects importants de la scène à mettre en avant, et quels sont ceux que l'on va préférer abstraire, voire même omettre. Il a aussi intérêt, tout du moins dans un premier temps, à imiter ses prédécesseurs : quel type de "medium" est employé (qu'il reproduise ou seulement s'inspire d'un medium traditionnel), quel type d'outil est utilisé (qu'il soit simulé ou simplement suggéré), et comment introduire le style personnel de l'artiste ?

D'un autre coté, l'ordinateur apporte de nombreuses nouvelles possibilités : le rendu expressif peut assister l'artiste pour produire plus facilement et plus rapidement une image ou une animation par le biais de méthodes automatiques ou semi-automatiques (dessin par analogie) ; il permet de mieux contrôler l'évolution d'une représentation au cours d'une animation (on parle de cohérence temporelle) ; il facilite la reproduction d'une oeuvre en enregistrant la composition d'une image sous forme numérique ; il apporte la notion d'interactivité, quasi-absente des média traditionnels ; etc. Ces nombreux avantages font du rendu expressif un outil utile à de nombreux domaines d'applications tels que l'animation, l'illustration scientifique, l'archéologie, l'architecture, le dessin technique ou les jeux vidéos.

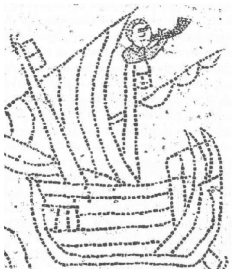
Nous présentons dans ce chapitre une vue d'ensemble de ce domaine. Le lecteur pourra compléter ce survol en se référant à deux livres de langue anglaise [GOO 01, STR 02] ainsi qu'à tous les articles que nous référons au cours du chapitre.

1.2. Les étapes du rendu expressif

Une manière d'étudier ce vaste domaine de recherche est de définir les divers éléments qui permettent de passer d'une scène 3D à un dessin ou peinture 2D. Comme proposé par John Willats et Frédo Durand [WIL 05, DUR 02], on peut décomposer un rendu en quatre étapes : le système spatial, l'extraction des primitives, le système d'attributs et le choix des marques.

Le système spatial est ce qui définit le changement d'espace. En général c'est une projection qui permet de passer du 3D au 2D. L'extraction des primitves fait correspondre les primitives de l'espace objet (points, courbes, surfaces, volumes) avec les primitives de l'espace image (points, lignes, régions). Le système d'attributs relie les propriétés visuelles telles que la couleur, la texture, l'épaisseur, etc... aux primitives de l'image. Finalement, le système de marques correspond au tracé physique des primitives à l'aide de marques dont l'aspect dépend des outils (e.g. pinceau, fusain), média (e.g. peinture à l'huile, aquarelle) et supports (e.g. toile, papier) employés. Le style d'un artiste est défini par la mise en oeuvre de chacun de ces quatre systèmes.

Pour fixer les idées, si l'on prend le cas ci-contre, le système de projection utilisé est ortographique ; les primitives extraites sont des lignes representant les silhouettes, les arêtes vives et des elements de matériaux ; les marques utilisées sont des pierres de mosaïque ; et pour dessiner chaque marque il faut choisir pour chaque ligne quelle est sa couleur, sa forme, son épaisseur, etc... Cela définira les attributs des marques.



Un style de dessin au trait est alors donné par le choix de la projection (spatial), le choix des éléments de la scène qui seront représentés avec les lignes (primitives), quelles marques choisit-on pour dessiner (marques) et quelles sont leurs propriétés (attributs). Nous détaillons dans le reste de ce chapitre les travaux existant pour chacuns de ces quatres sytèmes.

1.3. Système Spatial

La première étape d'un rendu expressif est de passer de l'espace 3D de la scène à l'espace 2D de l'image. La projection perspective est la plupart du temps utilisée, mais d'autres types de projection existent et sont employés à des fins artistiques ou didactiques en peinture et dessin. On peut les organiser en trois catégories : les variations sur les projections perspective et orthographique, les projections multi-perspectives et les projections topologiques.

Afin de définir les variations possibles sur les modèles de projection perspective et orthographique, nous reprenons l'approche développée par Levene dans sa thèse de Master [LEV 98]. Une projection est définie par trois caractéristiques : la forme de la surface de projection, assimilable à la combinaison objectif+film photographique dans un appareil photo ; les propriétés de convergence des lignes de vue, définies comme les lignes émanant de l'oeil dans la direction de la scène ; ainsi que la courbure des lignes de vue. Dans le cas d'une projection perspective classique, la surface de projection est plane, les lignes de vue convergent vers le centre de projection et sont rectilignes. En déformant la surface de projection, on peut obtenir une perspective curviligne. Des

peintures comme "Chambre a Arles" de Van Gogh emploient ce type de perspective curviligne, et certains objectifs de caméra dits "fish-eye" reproduisent cet effet. Si les lignes de vue sont parallèles (elles ne convergent ni ne divergent), la projection obtenue est oblique, et dans le cas spécifique où les lignes de vue sont perpendiculaires au plan image, on se retrouve dans le cas d'une projection orthographique. On retrouve ce type de projections dans l'art asiatique médiéval, mais aussi en illustration technique (vues de dessus, de face, de côté, etc). Si les lignes de fuite divergent, alors on obtient une perspective dite inversée, fréquemment utilisée dans les illustrations iconographiques Européennes du Moyen Âge. Enfin, la courbure des lignes de fuite peut également être modifiée dans un but artistique.

Certains peintres du XXième siècle comme Cezanne, de Chirico ou Escher ont également expérimenté la combinaison de multiples projections. Les travaux réalisés dans ce domaine en NPR se sont intéressés à combiner différentes vues généralement en perspective, d'où le nom de projections multi-perspectives [LEV 98, COL 04, YU 04, AGR 00]. La principale difficulté est de combiner les différentes projections ; il faut en particulier résoudre les problèmes de visibilité posés par des objets adjacents dans la scène mais projetés depuis différents points de vue. La solution généralement donnée est d'exprimer et combiner les différentes contraintes posées par chaque projection. L'approche de Coleman et al. [COL 04] a notamment été utilisée dans un court métrage appelé RYAN et a prouvé son efficacité dans ce contexte artistique (c.f. fig 1.1).

Enfin, les projections topologiques sont des transformations qui sont plus axées sur la structure d'un objet ou d'une scène afin d'en comprendre le sens ou le fonctionnement. Agrawala et al. ont notamment réalisé des travaux sur des systèmes de projection topologique, que ce soit pour la création de plans routiers [AGR 01], ou pour la création automatique de vues éclatées d'un objet et de plans d'assemblage [LI 04, AGR 03] (c.f. fig 1.1).

Figure 1.1. *Méthodes de projection. A gauche : combinaison de plusieurs perspectives [COL 04]. A droite : vue eclatée d'un objet [AGR 03].*

Un problème inverse

Un domaine connu sous le nom de modélisation à partir de croquis est également concerné par le passage d'un espace à un autre, mais cette fois-ci dans le sens inverse, d'un espace 2D vers un espace 3D. A partir d'un dessin en 2D réalisé par l'utilisateur, le système a pour but de reconstruire un objet 3D plausible. Toute la difficulté réside dans le fait que le problème est sous-contraint : plusieurs interprétations 3D sont possibles pour un même dessin en 2D. Le lecteur intéressé par ces techniques peut se reporter à [BOU 01, HAL 02].

1.4. Extraction des primitives

Les objets d'une scène 3D sont caractérisés par des primitives géométriques (points, courbes, surfaces, volumes) qui peuvent être représentées dans l'image par d'autres primitives (points, lignes, régions) une fois la projection effectuée.

Actuellement, l'essentiel des travaux d'extraction de primitives 2D portent sur les lignes. Nous ne détaillerons par conséquent pas le cas des points et régions. Les travaux concernant les points ont essentiellement visé le problème inverse de la modélisation à partir de dessin. Le cas des régions est traité en général par défaut : la primitive région est la projection de l'objet et ensuite c'est dans le travail de remplissage par des marques que se concentre le style. Quelques travaux basés sur de la segmentation d'image commencent à apparaître, notamment Wang et al [WAN 04b] explorent cette solution en traitant des vidéos pour produire un rendu de type dessin animé.

Lorsque l'on dessine un objet, il est fréquent de dessiner le trait qui sépare l'objet de son environnement, c'est-à-dire la silhouette. Cependant, dans un dessin au trait d'autres lignes sont ajoutées pour décrire la forme de l'objet. Plusieurs questions se posent alors : quelles sont les lignes importantes ? Comment les extraire ? Comment les rendre rapidement de manière interactive ? Nous donnons ici des éléments de réponse à ces questions en nous concentrant sur les lignes classiquement utilisées : silhouettes, contours suggestifs, crêtes et vallées. Ces lignes sont utilisées pour le rendu non-photoréaliste de surfaces *lisses*, c'est-à-dire sans angles vifs. Nous ne dressons pas une liste exhaustive de toutes les lignes qu'il est possible d'extraire d'une surface. Nous ne considérons pas par exemple les lignes de niveaux qui sont bien adaptées au rendu de terrain. Nous ne décrirons pas non plus dans le détail les lignes de courbures principales utilisées pour représenter les zones d'ombres sous forme de hachures [ZAN 04].

1.4.1. *Silhouettes*

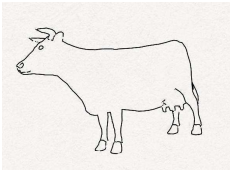
Dans cette partie nous donnons d'abord la définition des silhouettes puis nous décrivons des algorithmes d'extraction de ces silhouettes travaillant soit à la surface de

l'objet, soit dans l'espace image, soit dans les deux. Nous ferons ensuite une analyse critique comparée de ces algorithmes.

Définition

La silhouette est définie, pour un point de vue donné, comme l'ensemble des points d'une surface qui délimitent ses parties visibles de ses parties non visibles.

Sur cet exemple, la vache est dessinée uniquement par la représentation de la silhouette. Selon la définition d'Isenberg et al [ISE 03], on peut distinguer la silhouette externe et la silhouette interne. La silhouette externe (ou contour selon Isenberg) est la ligne qui sépare l'objet du fond. La silhouette interne sépare des sous-parties de l'objet ou un objet d'un autre. Sur l'exemple de la vache, les yeux et la délimitation des sabots font partie de la silhouette interne.



Formellement, la silhouette est constituée de l'ensemble des points où le vecteur de vue $\vec{v}$ est perpendiculaire à la normale à la surface $\vec{n}$. C'est-à-dire l'ensemble des points où le produit scalaire entre $\vec{v}$ et $\vec{n}$ est nul : $\vec{v}.\vec{n} = 0$.

Extraction

Dans le cas d'un modèle polygonal, la silhouette est composée des arêtes communes à une face visible et à une face invisible. Il est possible d'extraire la silhouette directement sur l'objet 3D ou dans l'espace image. Il existe aussi des méthodes mixtes. Pour une revue des méthodes de calcul des silhouettes, le lecteur peut se référer aux articles [HER 99, ISE 03].

Une méthode brutale pour l'extraction des silhouettes est de parcourir toutes les arêtes du maillage et de conserver celles qui sont connexes à un polygone visible et à un polygone invisible. Afin d'extraire plus rapidement les silhouettes, Markosian et al. [MAR 97] ont proposé une méthode stochastique (qui ne garantit pas de trouver les silhouettes dans leur intégralité). L'algorithme exploite la cohérence spatiale et temporelle des silhouettes. La recherche des segments candidats se fait donc dans le voisinage des segments déjà trouvés. Une fois les arêtes de silhouette identifiées à la surface de l'objet, il faut déterminer leur visibilité (voir Section 1.4.4.1).

On peut aussi détecter les silhouettes dans l'espace image en cherchant les discontinuités dans la carte des profondeurs [SAI 90].

Raskar et al [RAS 99] proposent une approche mixte qui procède en deux étapes : (i) Le z-buffer est d'abord initialisé avec la profondeur des faces visibles. (ii) Les faces arrière sont ensuite rendues en mode fil de fer. En utilisant l'égalité comme fonction de comparaison du z-buffer, les arêtes rendues sont celles qui partagent une face arrière et une face avant, c'est-à-dire les arêtes de la silhouette. Raskar a aussi

proposé un algorithme en une passe exploitant la carte graphique [RAS 01]. Cette version *hardware* ajoute des contours autour de chaque triangle. Les contours qui sont entre deux faces visibles disparaissent pendant le rendu. Il ne reste donc plus que le contour des silhouettes.

Analyse comparée des différentes techniques

La silhouette est dépendante du point de vue, *i.e.* elle doit être re-calculée chaque fois que la position ou la géométrie de l'objet est modifiée, et chaque fois que le point de vue change. Par conséquent, le rendu des silhouettes d'une scène animée peut s'avérer coûteux si les calculs mis en jeu ne sont pas optimisés. Les méthodes détectant les silhouettes dans l'espace image ou mixtes sont de bonnes candidates pour un rendu rapide des silhouettes. En effet, la complexité dépend uniquement de la résolution de l'image et non de la taille du maillage. L'inconvénient des méthodes travaillant dans l'espace image est que la silhouette est une matrice de pixel ; il est donc difficile de manipuler les lignes de la silhouette pour leur appliquer un style (par exemple varier l'épaisseur du trait au long de la ligne). Certains auteurs [CUR 98] ont proposé d'appliquer des traitements à l'image pour styliser les silhouettes. Malheureusement, le contrôle du rendu produit est pauvre et on utilisera plutôt des techniques travaillant à la surface de l'objet pour l'extraction de silhouettes que l'on souhaite styliser.

1.4.2. *Contours suggestifs*

Selon la définition de De Carlo et al. [DEC 03], les contours suggestifs sont constitués de l'ensemble des points de la surface qui : (1) appartiennent à la silhouette d'une vue voisine de la vue courante ; (2) ne sont pas en correspondance avec les points de la silhouette de la vue courante.

Plus formellement, les contours suggestifs sont les lieux où la courbure radiale (courbure normale dans la direction du vecteur de vue) s'annule et où la dérivée de cette courbure dans la direction du vecteur de vue est positive. La figure 1.2 illustre la localisation des contours suggestifs dans le voisinage de la silhouette et à l'endroit où la courbure radiale k_r s'annule. Le lecteur pourra se reporter à [DEC 03] pour une définition plus complète.

De Carlo et al. proposent deux méthodes d'extraction ; l'une travaille dans l'espace image et l'autre à la surface de l'objet. Les détails d'implémentation de ces méthodes sont discutés dans [DEC 04, DEC 03]. La figure 1.3 montre les contours (appelés silhouette selon la définition d'Isenberg donnée à la section 1.4.1 [1]) et les contours suggestifs extraits d'un objet 3D.

1. le terme contour est ambigü ; il désigne la silhouette selon De Carlo et al. [DEC 04] alors qu'il est la silhouette externe selon Isenberg [ISE 03].

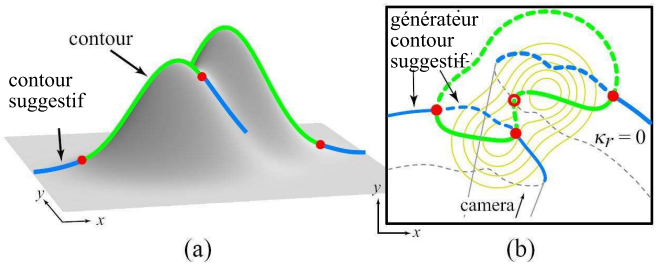


Figure 1.2. *(a) Les contours suggestifs (en gris foncé) étendent naturellement les contours (silhouette) de la surface (en gris clair). (b) Vue topographique des contours et contours suggestifs (lieux ou k_r s'annule) [DEC 03].*

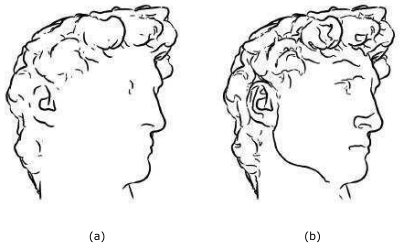


Figure 1.3. *(a) contours, (b) contours et contours suggestifs [DEC 03].*

Les contours suggestifs prolongent les silhouettes ce qui leur confère une certaine valeur esthétique. De plus, lorsque le point de vue change, les contours suggestifs anticipent l'apparition de silhouettes, ce qui adoucit les transitions entre les images. Les contours suggestifs sont liés à la courbure radiale, et donc au point de vue, ce qui implique que les contours suggestifs d'une scène animée doivent être re-calculés pour chaque changement de point de vue. Pour les séquences d'animation, ceci a deux conséquences : l'extraction des contours diminue les performances d'affichage ; les contours suggestifs se déplacent sur la surface quand la caméra évolue autour de l'objet. Cet effet peut susciter une gêne ; on a en effet le sentiment que les traits serpentent à la surface de l'objet.

1.4.3. *Crêtes et vallées*

Les crêtes et les vallées peuvent être appréhendées comme la généralisation des arêtes vives aux modèles lisses. La figure 1.4 illustre l'extraction de ces lignes qui sont complémentaires à la silhouette pour véhiculer la forme de l'objet.

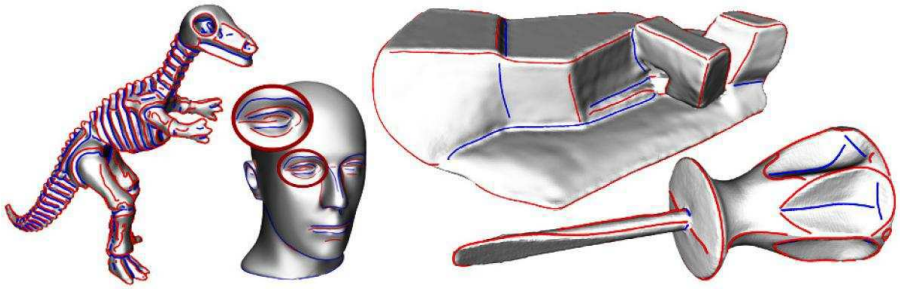


Figure 1.4. *Extraction de crêtes et vallées à la surface d'objets lisses [HIL 05].*

De façon formelle, elles sont définies comme les lieux où les valeurs de courbures principales atteignent des extrema locaux dans les directions principales. C'est-à-dire :

– pour les lignes de crêtes :

$$\frac{\partial \kappa_{max}}{\partial \vec{t}_{max}} = 0 \qquad et \qquad \frac{\partial^2 \kappa_{max}}{\partial \vec{t}_{max}{}^2} < 0 \qquad et \qquad |\kappa_{max}| > |\kappa_{min}|$$

– pour les lignes de vallées :

$$\frac{\partial \kappa_{min}}{\partial \vec{t}_{min}} = 0 \qquad et \qquad \frac{\partial^2 \kappa_{min}}{\partial \vec{t}_{min}{}^2} > 0 \qquad et \qquad |\kappa_{min}| > |\kappa_{max}|$$

où κ_{min} et κ_{max} sont respectivement les valeurs de courbures principales minimales et maximales et $\vec{t}_{min}$ et $\vec{t}_{max}$ les directions de courbures principales minimales et maximales.

Yoshizawa et al. [YOS 05] ont proposé un algorithme rapide pour la détection des crêtes et vallées à la surface maillée d'un objet. L'enjeu de ces algorithmes travaillant sur des modèles polygonaux est d'obtenir des lignes lisses malgré l'approximation du maillage. En effet, les valeurs des courbures principales sont évaluées à partir des variations de la normale, qui reflètent la géométrie d'un modèle sur un ensemble assez large de triangles. Hildebrandt et al. [HIL 05] proposent des méthodes de filtrage pour obtenir des lignes de meilleure qualité.

Les crêtes et les vallées sont des traits indépendants du point de vue. Ce sont donc des lignes intéressantes à conserver lorsqu'on souhaite manipuler un modèle 3D de manière interactive. Elles véhiculent une information forte sur la forme.

1.4.4. *Visibilité et cohérence temporelle*

1.4.4.1. *Visibilité*

Les problèmes de calcul de visibilité sont communs à toutes les familles de traits (crêtes et vallées, contours suggestifs, etc...). Une fois que ces traits ont été extraits à la surface de l'objet, on doit sélectionner pour le rendu le sous-ensemble visible de ces traits. Dans certains cas, on peut aussi vouloir rendre les traits cachés pour produire des effets de style. Les algorithmes que nous présentons ici ont été proposés dans le cadre du rendu des silhouettes mais s'appliquent aussi aux autres types de traits.

La méthode historique pour le calcul de visibilité est celle d'Appel [APP 67]. Markosian et al [MAR 97] ont adapté cette méthode dans le cadre du rendu NPR temps-réel ; l'information de visibilité extraite est très précise. A chaque arête $\mathcal{A}_i$ de la ligne de silhouette on associe une valeur QI (Quantitative Invisibility) représentant le *degré* d'invisibilité. Sachant un point de vue, QI correspond au nombre de faces séparant $\mathcal{A}_i$ du point de vue (QI est calculé par lancé de rayon). Une arête est donc visible lorsque QI est nul. L'intérêt de ce calcul de visibilité est que l'on peut appliquer des styles différents aux traits en fonction de la valeur de QI (rendre par exemple les traits cachés en pointillés). Cependant, cette information de visibilité quantitative n'est pas toujours nécessaire. Plus récemment, Northup et al [NOR 00] ont utilisé une méthode mixte et plus rapide pour la gestion de la visibilité. Toutes les facettes de l'objet 3D ainsi que les arêtes $\mathcal{A}_i$ de la silhouette sont rendues avec une couleur distincte dans un tampon d'identificateurs (ID-Buffer). Les arêtes visibles $\mathcal{A}'_i$ sont celles qui sont représentées par au moins un pixel dans le tampon d'identificateurs. Pour chaque $\mathcal{A}'_i$, on détermine précisément sa portion visible dans l'image. La dernière étape se déroule dans l'image et consiste à connecter les segments pour obtenir un chemin de qualité visuelle satisfaisante dans l'image.

1.4.4.2. *Cohérence temporelle*

La question de la cohérence temporelle se pose lorsque les lignes sont dépendantes du point de vue. Lors de changements de point de vue, les nouvelles lignes à afficher peuvent être très différentes des anciennes et des effets de scintillement nuisibles à la visualisation peuvent apparaître.

Pour le rendu des contours suggestifs, DeCarlo et al. [DEC 04] proposent d'anticiper l'apparition de nouveaux contours suggestifs dans l'image. Les contours sont rendus de manière plus ou moins prononcée en fonction de leur coïncidence avec la vue courante. Cette technique de fondu permet de limiter les effets de scintillement liés à l'apparition soudaine d'un nouveau contour.

En ce qui concerne les silhouettes, les rendus successifs sont souvent proches les uns des autres lorsque l'on manipule un objet. Les effets de discontinuité sont donc réduits. Cependant, lorsque la silhouette est dessinée à partir des lignes passant par les

arêtes du maillage, on peut observer de fortes discontinuités lorsque le maillage est grossier. Hertzmann et Zorin [HER 00] ont proposé une méthode permettant d'obtenir une silhouette plus précise. Grâce à leur méthode les transitions sont plus douces d'une image à l'autre au niveau du rendu de la silhouette. Pour trouver la silhouette, Hertzmann et Zorin proposent l'algorithme suivant : pour chaque face, si le signe du produit scalaire $\vec{v}.\vec{n}$ est différent sur les 2 sommets d'une arête, une interpolation linéaire est effectuée le long de cette arête pour déterminer le point P_i où $\vec{v}.\vec{n}$ s'annule. Les segments de la silhouette sont créés en connectant ces points P_i.

1.4.5. *Cas volumique*

Les méthodes de détection de traits à la surface d'objets 3D polygonaux ont été adaptées au cas volumique [INT 97, BUR 05b]. Les lignes caractéristiques sont extraites sur les iso-surfaces du volume. La Figure 1.5 montre que l'abstraction fournie par le rendu à base de traits facilite la compréhension de modèles volumiques complexes.

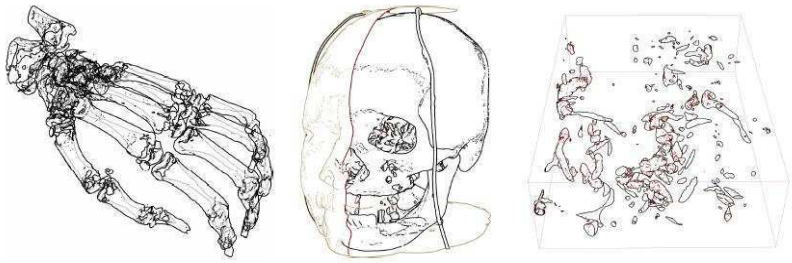


Figure 1.5. *Divers dessins au trait de volumes [BUR 05b] : visualisation en transparence des contours d'un squelette de main ; plan de coupe d'un crâne et de sa peau ; simulation de turbulence.*

1.4.6. *Quelles lignes veut-on dessiner ?*

De nombreux articles traitent donc d'extraction de lignes caractéristiques. L'extraction de ces lignes consiste en la détection de propriétés géométriques à la surface des objets. Malheureusement, peu de travaux se sont attachés à mesurer la qualité visuelle de ces lignes et leur adéquation à représenter correctement la forme d'un objet. Lum et Ma [LUM 05] proposent une méthode d'apprentissage pour la sélection de lignes pertinentes à la surface d'un objet lisse. Les utilisateurs entraînent un réseau de neurones en sélectionnant des lignes sur une série d'objets. Les inconvénients de la méthode proposée par Lum et Ma sont ceux des méthodes d'apprentissage : (i) constitution fastidieuse de la base de données, (ii) la structure du réseau de neurones ne

permet pas de comprendre les mécanismes mis en jeu dans la phase de sélection. De nouvelles voies sont donc à explorer. Une approche quasi-automatique pourrait être de guider la sélection des lignes par des facteurs psycho-visuels intervenant dans la compréhension d'une scène. Il peut aussi être intéressant (mais délicat) de modéliser l'activité de l'artiste lorsqu'il sélectionne les lignes importantes pour rendre un objet.

1.5. Choix des marques

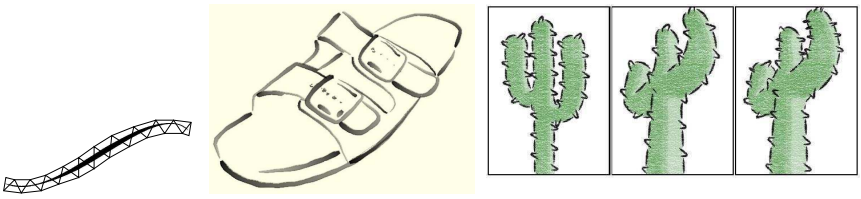
Une fois les primitives extraites la question se pose de les dessiner. Pour cela, selon le style visé, chaque primitive est "remplie" par des marques. Une marque est ainsi la représentation concrète de la trace de l'outil de dessin dans l'image : trait à l'encre, coup de pinceau, à-plat d'aquarelle, etc...

Divers algorithmes ont été proposés visant à reproduire des média particuliers. On peut les classer en deux grandes catégories : l'illustration et la peinture ; ou plus généralement le noir et blanc et la couleur. Le lecteur pourra se référer à l'état de l'art de Hertzmann [HER 03] pour plus de détails.

Mais avant de présenter ces deux familles d'algorithmes, il est important de discuter d'un problème fondamental en rendu expressif pour l'animation : la cohérence temporelle. Tout comme les primitives, les marques sont des éléments 2D qui sont utilisés pour représenter une scène en 3D, ce qui va donc provoquer des évènements d'apparition et de disparition de marques. Par exemple, le mouvement des objets en 3D va faire apparaitre des portions de surface qui n'étaient pas visibles dans les images précédentes, et afin de représenter ces nouvelles régions dans l'image, il va falloir introduire des marques dans le dessin. Un problème similaire se produit lors de la disparition de marques. L'apparition ou disparition subite d'une marque provoque une gène visuelle, un problème de cohérence temporelle. Dans la suite nous présentons des méthodes destinées à distribuer des marques et précisons, lorsque c'est le cas, lesquelles assurent une cohérence temporelle. Nous revenons sur ce problème récurrent en Section 1.7.1.

1.5.1. *Illustration*

L'illustration regroupe l'ensemble des techniques de dessin noir et blanc visant essentiellement à représenter l'ombrage des objets ainsi que leurs lignes caractéristiques. Ce type de visualisation est utilisé aussi bien par les graphistes (pour des bandes dessinées par exemple) que les dessinateurs techniques. On trouvera dans cette catégorie des algorithmes visant à produire des dessins à l'encre de chine [SAL 94, WIN 94, SAL 96, WIN 96, SAL 97, HER 00, PRA 01, ZAN 04], au crayon [SOU 99, RUD 05, MAT 05] ou des gravures [OST 99]. Les marques seront alors des traits, des hachures ou des points.



(a) (b) (c)

Figure 1.6. *Dessin de lignes : une bande de triangle texturée (à gauche) sert de support au dessin des lignes [NOR 00] (au milieu). Un suivi d'une image à l'autre permet de gerer la cohérence temporelle du style [KAL 03].*

1.5.1.1. *Lignes*

Dans le cas où la primitive à dessiner est une ligne (silhouette, crête, etc...) issue de l'extraction, on dessinera un ou plusieurs traits pour la représenter. Pour pouvoir styliser une telle ligne il nous faut disposer d'un paramétrage au long de la ligne, que ce soit une ligne brisée ou une courbe. Concrètement, la technique la plus utilisée est de construire une bande de triangles (*triangle strip*) le long de la ligne sur laquelle on viendra plaquer une texture, éventuellement semi-transparente. Le choix du style et des attributs du trait sera discuté dans la Section 1.6.4.

Enfin, se pose la question de la cohérence temporelle de la stylisation. Kalnins et al [KAL 03] ont proposé une méthode dont le principe général est un aller-retour entre l'espace image et l'objet 3D. Pour obtenir une stylisation continue entre deux images, on propage le paramétrage 2D des traits d'une image à l'autre.

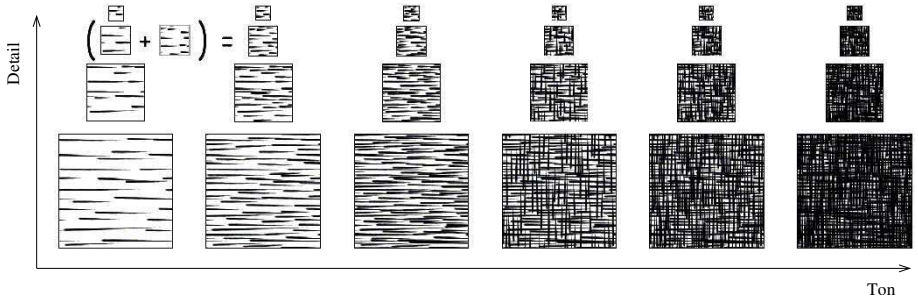
1.5.1.2. *Hachures*

De très nombreux travaux se sont intéressés au hachurage car cette technique est une des plus utilisée en illustration scientifique et technique ainsi qu'en dessin traditionnel. Les hachures servent à la fois à représenter l'ombrage de l'objet et décrire sa forme. Les hachures seront donc des marques servant à remplir des régions. Les questions posées sont alors : comment choisir les divers attributs (taille, épaisseur, orientation) des hachures et leur densité en fonction de l'illumination et de la forme de l'objet ? Comment assurer une cohérence du rendu lors d'une animation, c'est-à-dire comment gérer l'apparition et la disparition de hachures par exemple lors d'un changement d'illumination ?

Les premiers travaux dans ce domaine sont issus de chercheurs travaillant sur la problématique de la gestion du ton (*halftoning*) pour l'impression [SAL 94, WIN 94, SAL 96, WIN 96, SAL 97]. Ces travaux offrent des algorithmes produisant des illustrations hachurées à partir d'images noir et blanc ou de rendus de modèles 3D sous un point de vue fixe. Typiquement, l'utilisateur aura à fournir une succession de textures

de hachures donnant divers tons et l'algorithme choisira automatiquement quelle texture appliquer en fonction du niveau de gris correspondant dans l'image de référence. L'utilisateur pourra aussi choisir l'orientation des textures pour accorder la direction des marques en fonction de la scène représentée.

Si l'on veut produire un rendu hachuré d'un objet animé en plaçant des hachures à sa surface, il faut ajouter des niveaux de détail pour chaque texture de ton. Sans cela, lors d'un zoom, les hachures grossiraient progressivement et ni le ton ni le style ne seraient conservés. On ajoute alors un second axe à l'ensemble des textures disponibles qui est celui du détail pour obtenir une *tonal art map* [PRA 01] :



L'algorithme de rendu devra cette fois choisir en chaque point de la surface du modèle la bonne texture en fonction du ton et de la distance à la caméra (voir Figure 1.7).

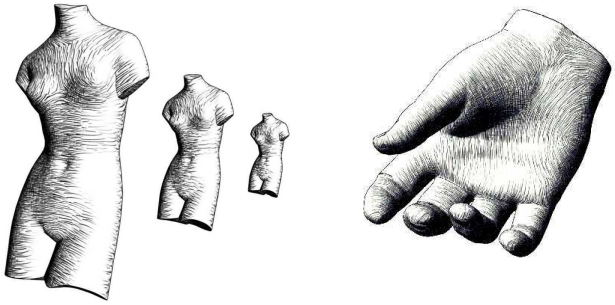


Figure 1.7. *Rendu temps réel de hachures [PRA 01].*

1.5.1.3. *Pointillage*

Le pointillage (*stippling*) se rapproche du hachurage dans le but visé : représenter l'illumination. Les marques utilisées sont cette fois des points et l'on retrouve ici la problématique de conservation du ton et de gestion du détail lors des zooms. Les techniques de hachures basées sur des textures peuvent ainsi être utilisées directement

(a) (b)

Figure 1.8. *Pointillage : (a) vue statique [DEU 00], (b) temps-réel [PAS 03].*

pour produire du pointillage.

Cependant, dans le cas du pointillage, la distribution des points est un aspect fondamental qui est moins traité dans le cas de hachures. En effet, tout l'intérêt visuel du pointillage réside dans la distribution des points qui permet de ne mettre en évidence que le ton de l'objet représenté. Plusieurs chercheurs se sont donc intéressés à la génération automatique de textures de points ayant de bonnes propriétés et de ton et de distribution. Ces algorithmes fonctionnent tous sur le même modèle : choisir le nombre de points nécessaires pour un ton donné et les répartir uniformément dans le plan puis modifier leur position pour obtenir une répartition visuellement satisfaisante. Deussen et al. [DEU 00] utilisent par exemple une relaxation de Voronoi par l'algorithme de Lloyd [LLO 82], pour obtenir une telle répartition.

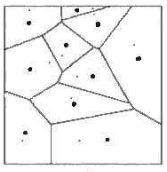
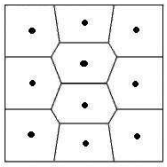


Diagramme de Voronoi



Apres relaxation de Llyod

De même que pour les hachures, si l'on veut produire un rendu d'une scène animée il faut assurer la cohérence temporelle des points ainsi que leur apparition et disparition en fonction du niveau de zoom. Pastor et al. [PAS 03] ont proposé un algorithme qui dessine les points directement sur le modèle 3D (voir Figure 1.8). Par un mécanisme de simplification et de subdivision progressive ils peuvent ajouter ou enlever des points de manière continue en fonction du ton et de l'échelle visée. En pratique, leur algorithme consiste à dessiner un point par sommet du maillage représentant l'objet. Puis ils utilisent des contractions d'arêtes successives pour supprimer des sommets ou une subdivision de surface pour en ajouter. Ils ajoutent ensuite un peu d'aléatoire dans le positionnement des sommets pour améliorer la répartition des points.

Une alternative à l'ajout et à la suppression de points est de faire varier leur taille en fonction du ton. Cependant une telle méthode s'éloigne du pointillage classique et n'est valable que pour de faibles variations d'échelle.

1.5.2. *Peinture - Couleur*

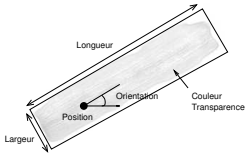
La peinture ou plus généralement les techniques utilisant la couleur visent à remplir les régions d'une image avec des marques colorées. Les primitives extraites seront ainsi des régions de couleur, généralement uniformes, que l'on veut rendre dans un style qui va de la peinture à l'huile (impressionisme, pointillisme...) à l'aquarelle. Dans cette catégorie se situe aussi la mosaïque [SMI 05] que nous ne détaillerons pas ici. La mosaïque ajoute une problématique d'arrangement dans le plan à celle de la reproduction de couleur et se rapproche ainsi du pointillage.

1.5.2.1. *Peinture à base de marques*

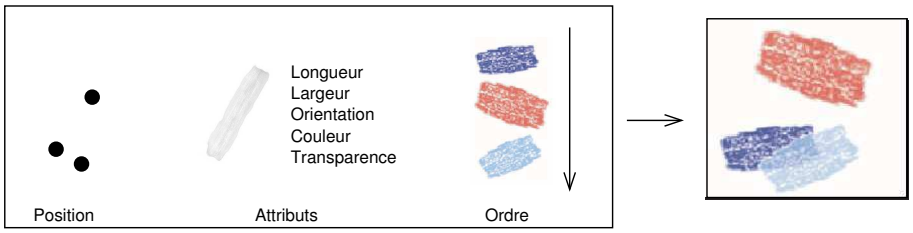
Figure 1.9. *Rendu peinture [HER 98].*

La peinture à l'huile avec des coups de pinceau bien identifiés a été largement étudiée. Alors que les premiers travaux concernent des images fixes (photographies ou images de synthèses), des méthodes récentes se sont intéressées au cas d'images animées (vidéos ou animations de synthèse).

La marque est alors un coup de pinceau, généralement défini par une petite région texturée. La question posée est alors comment répartir les coups de pinceaux dans l'image et quels attributs (taille, orientation, couleur, ...) leur donner pour obtenir le style visé.



La plupart des algorithmes sont des algorithmes gloutons : les marques sont placées et dessinées les unes après les autres jusqu'à arriver à un recouvrement complet de l'image. Les choix à faire pour de tels algorithmes sont alors : (1) comment choisir le lieu de placement des marques, (2) quels sont leurs attributs (ceci sera discuté en détail lorsque nous parlerons des attributs en Section 1.6.4), (3) comment choisir l'ordre de dessin des marques ?



La question du positionnement des marques est primordiale. On peut bien sûr donner la main à l'utilisateur [HAE 90][2] pour ce choix, créant ainsi un outil de peinture. Cependant des méthodes automatiques ont été développées depuis. La plus simple est de placer les marques sur une grille régulière, perturber leur position aléatoirement puis les dessiner dans un ordre aléatoire [LIT 97]. Cette méthode peut être améliorée en travaillant par couches successives : dessiner de grandes marques d'abord puis compléter par de plus petites là où il manque de l'information, c'est-à-dire là où il y a des détails dans l'image de départ. C'est la méthode utilisée par exemple dans [HER 98] qui place les marques en utilisant des versions plus ou moins floues de l'image de départ (voir Figure 1.9). Le positionnement peut aussi être effectué en utilisant les méthodes de pointillage pour répartir les marques de manière visuellement plaisante dans l'image.

La question du positionnement se complique lorsque l'on s'intéresse à des animations. Que ce soit une vidéo ou une animation de synthèse, la position des marques et tous leurs attributs doivent rester cohérents d'une image à l'autre.

Deux grandes classes de méthodes existent alors selon que l'on considère uniquement les images (comme pour une vidéo) ou que l'on se réfère à la géométrie de la scène animée. Dans le cas d'une vidéo on utilise généralement le flux optique du pixel au centre de chaque marque pour estimer le déplacement de la marque [LIT 97, HAY 04]. Dans le cas d'une scène 3D la principale méthode consiste à accrocher des points à la surface des objets. Leur projection dans l'image donnera la position des marques [MEI 96, DAN 99].

1.5.2.2. *Aquarelle*

Le rendu aquarelle se distingue des autres rendus à base de marque car il n'y a plus vraiment de marque individuelle (au sens d'un coup de pinceau ou d'un trait) mais plutôt des à-plats de couleurs possédant des caractéristiques propres : un grain caractéristique ; une accumulation des pigments aux bords ; des ondulations aux bords ; une dispersion et diffusion des pigments.

2. `ftp ://ftp.sgi.com/sgi/graphics/grafica/impression/index.html`

Figure 1.10. *Simulation physique [CUR 97] - Filtre [BOU 06].*

Deux grands types de méthodes ont été développés pour le rendu aquarelle : des simulations physiques ou des méthodes à base de textures.

Les simulations physiques reproduisent fidèlement l'interaction du papier, des pigments et de l'eau. Le travail le plus abouti étant celui de Curtis et al. [CUR 97]. Le principal inconvénient de ces méthodes est le temps de calcul bien que des méthodes de simulation de fluides temps-réel commencent à voir le jour. Nous citerons par exemple [CHU 05] qui simule la dispersion de l'encre de Chine.

D'un autre coté de nombreux chercheurs ont tenté de reproduire les effets visuels de l'aquarelle sans passer par une simulation physique. On obtient ainsi des algorithmes de rendu temps-réel comme dans [LUM 01, LEI 04, VAN 04, LUF 05, BUR 05a, JOH 05, BOU 06]. Ces méthodes ne reproduisent généralement pas les effets de dispersion mais en revanche donnent de bons résultats pour le grain, l'assombrissement des arêtes, etc...

Le rendu aquarelle pose encore une fois la question de la cohérence temporelle dès lors que l'on veut animer une scène. En particulier la texture de pigments, caractéristique de l'aquarelle, doit évoluer de manière cohérente avec les objets de la scène sans pour autant perdre ses caractéristiques intrinsèques comme la fréquence des grains de pigments par exemple. Des travaux sur la cohérence du papier [CUN 03, KAP 05] ont par exemple été étendus au cas de l'aquarelle [BOU 06]. Il est évident que dans le cas d'une simulation physique la cohérence temporelle devient extrêmement difficile à assurer et cela reste aujourd'hui un problème de recherche ouvert.

1.6. Système d'attributs

Dans la section précédente, nous avons montré comment des marques assimilables à des coups de pinceau ou de crayon sont distribuées afin de représenter des primitives dans l'espace image. Nous n'avons cependant pas toujours précisé comment les attributs de ces marques, tels que la couleur, le ton, la transparence, l'orientation, la texture ou l'épaisseur, pouvaient être choisis. De tels choix peuvent soit être contraints par la scène elle même, comme l'orientation des marques le long d'une silhouette ou la

couleur d'un objet, ou bien être spécifiés par l'utilisateur, comme la composition des couleurs ou la finesse du trait. L'enjeu du système d'attributs est donc de combiner les informations qui proviennent de la scène et le style spécifié par l'utilisateur afin de déterminer la valeur d'un attribut. Nous présentons tout d'abord un système qui laisse suffisamment de liberté à l'utilisateur pour réaliser cette combinaison lui-même ; puis nous détaillons les scénarios les plus fréquemment employés afin de choisir les attributs de couleur, d'épaisseur et d'orientation.

1.6.1. *Freestyle, un système générique*

Le programme FreeStyle [GRA 04b] conçu par Grabli et al. est un système de dessin au trait qui a pour but de connecter les attributs d'une scène 3D vue en perspective aux attributs d'un rendu au trait qui la représente. Le système est conçu de façon à ce que le contrôle donné à l'utilisateur soit maximal, et s'inspire de la notion de *shader* : le style du dessin est défini par un script écrit par l'utilisateur qui décrit explicitement les choix de valeurs d'attributs des lignes (épaisseur, couleur, texture, variations, etc) en fonction des informations provenant de la scène. Plusieurs styles sont ainsi possibles pour une même scène et une fois écrit, un même style peut être appliqué à plusieurs scènes automatiquement (c.f. fig 1.11). Le système étant limité aux dessins au trait et aux images fixes, des extensions à l'animation ou à d'autres média sont envisageables. D'autres systèmes proposent des fonctionnalités semblables à celles de Freestyle comme les G-Strokes d'Isenberg et al. [ISE 05].

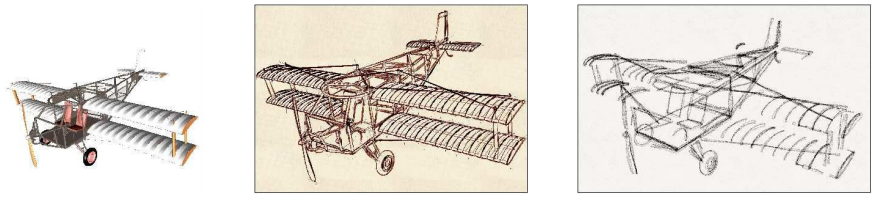


Figure 1.11. *Le système Freestyle [GRA 04b] : à partir d'un objet 3D (à gauche), l'utilisateur peut specifier les attributs des lignes dans deux styles différents (au milieu et à droite).*

1.6.2. *Attribut et contrôle de la couleur*

La couleur est l'attribut le plus important d'un rendu, c'est pourquoi de nombreuses méthodes ont été développées pour contrôler cet attribut. Nous présentons ici trois scénarios différents de contrôle de la couleur : les effets de type *cartoon*, les effets de type aquarelle et le post-traitement de couleurs.

Afin d'obtenir un effet *cartoon*, deux approches ont été précédemment utilisées : l'une nécessite un modèle 3D et consiste en une modification du modèle de Phong,

tandis que l'autre opère uniquement en espace image en appliquant une segmentation. Le modèle d'illumination de Phong a été intensément utilisé dans le passé ; il décrit de manière empirique le comportement d'un matériau par le biais de trois composantes : ambiante, diffuse et spéculaire. L'exemple le plus direct de modification du modèle de Phong est le Toon shader [LAK 00] (c.f. fig 1.12). L'idée est simple : on ne considère ici que la composante diffuse du modèle de Phong que l'on utilise pour indexer une texture 1D qui décrit comment la couleur du matériau évolue d'une intensité sombre à une intensité claire. La composante diffuse est définie par $\vec{l}.\vec{n}$ ou $\vec{l}$ est la direction d'illumination et $\vec{n}$ est la normale au point considéré. En introduisant des discontinuités dans la texture 1D (en utilisant par exemple 3 bandes de couleurs foncée, moyenne et claire), on parvient à reproduire un effet similaire à celui observé dans de nombreuses bandes dessinées et dessins animés (effet *cartoon*). Une autre approche est celle adoptée par Gooch et al. [GOO 99]. Ici, la composante diffuse est modifiée pour créer un dégradé d'une couleur froide vers une couleur chaude, avec des résultats similaires à ceux observés en illustration technique. A contrario, peu de méthodes se sont intéressées à la composante spéculaire du modèle de Phong. Anjyo et al. [ANJ 03] propose un modèle qui manipule un vecteur $\vec{h}$ (appelé half vector et dérivé de $\vec{l}$ et $\vec{n}$) afin de créer des matériaux de type plastique ou métal brillant dans un style *cartoon*. Outre le contrôle des tâches spéculaires, d'autres effets peuvent être contrôlés en partant du même principe, comme le fait la méthode de Barla et al. [BAR 06b], illustrée plus bas dans l'exemple de perspective atmosphérique

Les outils de segmentation ont eux pour but de partitionner une image en régions de couleur constante. Les paramètres fournis par l'utilisateur permettent de contrôler le nombre et la taille des régions obtenues. Malheureusement, ces paramètres ne sont pas toujours intuitifs, et une bonne segmentation est difficile à obtenir. Récemment, la méthode du *mean-shift* [COM 99] a donné de très bons résultats et s'est avérée être plus intuitive que les méthodes précédentes ; de plus, la méthode a également été étendue au traitement de vidéos [WAN 04a, WAN 04b] et assure une bonne cohérence temporelle (c.f. fig 1.12). Une segmentation est une abstraction radicale, puisqu'elle associe à une region de l'image une couleur uniforme. Cette méthode permet ainsi de produire des effets de type *cartoon* à partir d'une image ou d'une video.

Lorsque l'on s'intéresse au rendu de type aquarelle, un autre besoin se pose en terme de contrôle des couleurs : on veut pouvoir cette fois-ci abstraire non pas les variations de couleurs au sein d'une région comme avec une segmentation, mais la forme même des dites régions, afin notamment de se débarrasser de certains détails dans l'image. Les outils de morphologie permettent de donner une méthode intuitive afin d'atteindre ce but. Ici, l'idée est de contracter ou de dilater des régions de couleurs similaires. Une dilatation suivie d'une contraction permet d'obtenir un effet dit de fermeture, où deux régions similaires et proches sont fusionnées. Une contraction suivie d'une dilatation permet d'obtenir un effet dit d'ouverture, où deux régions connectées par une fine bande de pixels sont séparées. Ce dernier effet permet également de supprimer les petites régions.

Figure 1.12. *Deux exemples de rendus* cartoon *: à partir d'un objet 3D (à gauche), un éclairage de type* cartoon *est obtenu en utilisant une texture 1D qui caractérise l'illumination ; à partir d'une image ou d'une vidéo (à droite), des régions de couleur uniforme sont extraites [WAN 04a].*

Finalement, indépendamment du style choisi, des outils de correction colorimétrique peuvent être employés en post-traitement afin de contrôler la composition finale des couleurs. Ces outils incluent la correction du contraste, de la saturation, l'égalisation et l'édition d'histogrammes, etc.

Nous allons ici présenter deux outils spécifiques au rendu expressif : la perspective atmosphérique et le transfert de couleurs. La perspective atmosphérique est couramment employée en peinture afin de retranscrire la profondeur de la scène représentée : les couleurs sont ainsi altérées afin d'imiter les effets dus à la brume par exemple (voir Figure 1.13). Ces effets incluent une diminution de la saturation et une diminution du contraste avec la profondeur, une simplification de la distribution des couleurs dans les plans lointains, ainsi que la convergence vers une teinte spécifique en fonction du temps représenté (bleutée pour la journée, rosée pour le matin, etc). Ces effets peuvent être introduits dans un rendu expressif, par exemple en utilisant la texture toon 2D de Barla et al [BAR 06b] : l'axe horizontal de la texture correpond au contrôle du ton (couleurs foncées à couleurs claires), tandis que l'axe vertical représente la distance au point de vue. Les méthodes de transfert de couleur [REI 01, CHA 03], quant à elles, utilisent une image de référence, comme une peinture par exemple, ont pour but de modifier une image source afin que sa composition colorimétrique soit similaire à celle de l'image de référence (voir Figure 1.14). Ceci permet notamment de contrôler la composition des couleurs dans l'image, tout en conservant leurs valeurs relatives provenant de la scène.

1.6.3. *Attribut d'illumination, contrôle de la densité, de l'épaisseur*

Dans un style de rendu de type illustration, comme présenté Section 1.5.1, les variations de densité et d'épaisseur doivent permettre de représenter fidèlement le ton sous-jacent. Elles peuvent être réalisées semi-automatiquement : Deussen et al. [DEU 00] présentent ainsi une méthode ou l'utilisateur emploie des outils intelligents

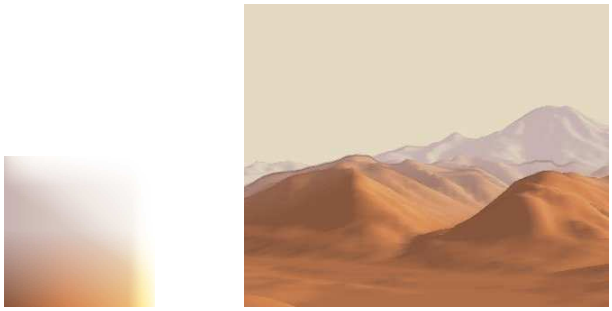


Figure 1.13. *Perspective atmosphérique : à gauche, la texture toon 2D utilisée pour contrôler l'effet ; à droite, l'effet appliqué à un modèle de terrain.*

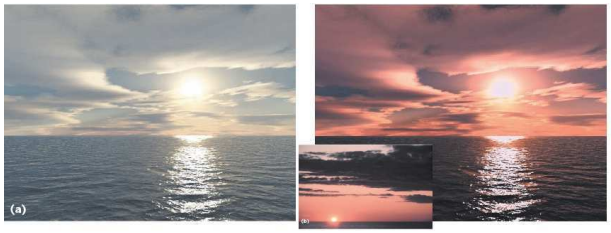


Figure 1.14. *Transfer de couleurs : à gauche, l'image source ; à droite, l'image de référence iconifiée et l'image source après transfert de couleurs*

de régularisation de la distribution (méthodes de relaxation), ou de réglage de l'épaisseur des primitives. Elles peuvent également être réalisées analytiquement : Secord et al. [SEC 02] utilisent des distributions de probabilité ayant des propriétés d'espacement des primitives adéquates. Elles peuvent finalement être pré-calculées dans ce qui est appelé une *dither matrix* : l'évolution de l'épaisseur des lignes est alors directement décrite dans la matrice, avec des applications en rendus de type gravure [OST 99] (voir Figure 1.15).

1.6.4. *Attribut de courbure, contrôle de l'orientation*

Un autre exemple de modification typique des attributs concerne l'orientation des marques dans un dessin. Girshick et al. ont montré que des lignes suivant les directions de courbure d'une surface permettent de mieux représenter sa forme. On peut donc lier l'orientation des marques d'un dessin aux directions de courbure principale ou secondaire d'un objet [INT 97, GIR 00] (c.f. Figure 1.15). Cependant, dans un autre contexte, on peut vouloir aussi lier l'attribut orientation à d'autres propriétés de la scène. Par exemple, les lignes remplissant une zone d'ombre peuvent être orientées

dans la direction perpendiculaire à la source de lumière afin d'augmenter la perception de la direction d'illumination [SAL 94].

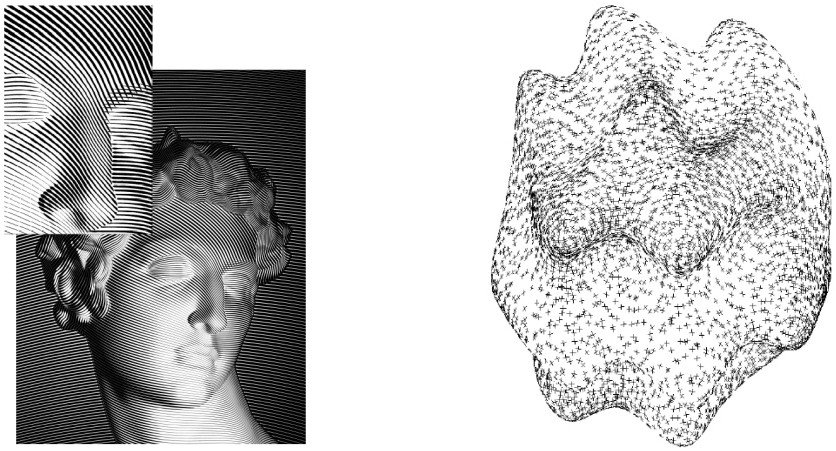


Figure 1.15. *A gauche : contrôle de la densité et de l'épaisseur des primitives. A droite : contrôle de l'orientation par le biais des informations de courbure.*

1.7. Discussion

Ce chapitre vous a présenté un aperçu des techniques et résultats en rendu expressif. Nous terminons par quelques éléments de discussion sur les principes généraux de ce domaine qui offrent autant de nouvelles pistes de recherche.

1.7.1. *Cohérence temporelle*

Nous avons vu au cours de ce chapitre que le problème de la cohérence temporelle était constamment présent. Ce problème est, en effet, inhérent au rendu expressif dès lors que l'on dessine des marques qui sont plus grosses que le pixel. Il faut ainsi répondre à la question du mouvement des marques fonction du mouvement des objets de la scène et savoir gérer l'apparition et la disparition des marques. Cette question met en lumière la contradiction existant entre un mouvement 2D et un mouvement 3D. Par exemple si l'on considère un zoom sur un objet que l'on dessine avec des hachures, on ne peut pas en même temps conserver le même nombre de traits et la même densité (qui donne le ton) alors que la surface occupée par la projection de l'objet augmente. Il faut donc faire des choix qui correspondront à diverses métaphores selon si les marques suivent les objets [MEI 96, DAN 99] ou bien les couleurs dans l'image [WAN 04b] ou bien le mouvement de la caméra [CUN 03, KAP 05].

1.7.2. *Le rendu expressif comme processus d'optimisation*

Une autre problématique qui apparait à chaque étape d'un rendu expressif est le conflit entre les contraintes provenant des informations de la scène à représenter et les contraintes liées au style employé. Durand [DUR 02] a reformulé le problème comme un problème d'optimisation : le rendu expressif consiste alors à produire une image qui satisfait au mieux les intentions de l'utilisateur.

Par exemple, pour la création d'une projection multi-perspective, l'utilisateur spécifie différentes vues qui doivent être ensuite "mélangées" dans une seule image en essayant de combiner les contraintes provenant de chaque vue. Un autre exemple concerne les contours suggestifs. Une extention de la méthode initiale permet de les obtenir interactivement [DEC 04]. Cependant, afin d'éviter des effets d'apparition/disparition abrupts, les contours suggestifs sont filtrés par leur vitesse d'apparition afin de ne pas distraire l'oeil. La création de marques est également soumise à des contraintes antagonistes. C'est notament le sujet de l'article de Kalnins et al. [KAL 03] qui proposent trois méthodes pour dynamiquement paramétriser une silhouette en vue de lui appliquer des marques : une méthode 2D, une 3D et une méthode d'optimisation. Enfin, le système d'attributs est directement concerné par ce processus, où les attributs des marques sont imposés par l'utilisateur mais doivent cependant représenter certains aspects de la scène.

1.7.3. *Style*

Le terme "style" a été jusqu'ici employé sans vraiment y prêter attention. En fait, il n'est pas évident du tout de définir ce qu'est le style et par conséquent de le modéliser de manière algorithmique. Pourtant on voit bien que le domaine du rendu expressif vise de manière générale à offrir des visualisations dans des styles les plus variés possible, et dans l'idéal à laisser l'utilisateur créer son propre style.

Le logiciel "freestyle" [GRA 04b] présenté Section 1.6.4 est une première tentative d'étudier ce que pourrait être un modèle de style dans le cas particulier du dessin au trait. La question reste largement ouverte et la recherche sur les systèmes d'attributs est certainement une voie prometteuse dans la compréhension algorithmique du style.

Une autre voie est celle de l'analogie. Au fond, peut-on reproduire un style donné par imitation sans nécessairement être capable de le décrire procéduralement ? C'est la voie prise par plusieurs travaux dans le cas de l'illustration [JOD 02, FRE 03, DRO 04, BAR 06a].

1.7.4. *Abstraction*

Pour conclure, la notion d'abstraction a jusque-là été relativement peu étudiée dans le cadre de la synthèse d'image. Pourtant aussi bien dans le domaine artistique que

technique l'évolution des représentations est allée vers plus d'abstraction. L'histoire de l'art est un exemple connu de ce phénomène mais l'illustration technique par exemple a suivi le même chemin.

La raison de cette évolution est probablement due au fait qu'une visualisation abstraite offre une représentation concise et schématique de l'objet à représenter. Un exemple typique est celui des cartes routières. Une photo aérienne est certainement moins pratique et véhicule moins d'information qu'une carte schématique. Les graphiques utilisés par les physiciens depuis le XVIIIème siècle sont eux aussi caractéristiques de la puissance d'une représentation visuelle schématique. Il nous parait ainsi crucial de développer des techniques d'abstraction et de schématisation pour l'informatique graphique. Au fond, si l'on voit l'ordinateur comme un outil de communication visuelle alors la question devient : quel message transmet-on pour quelle application ? D'un point de vue pratique cela suppose de savoir mettre en valeur les élements pertinents d'une scène, éventuellement de quitter la projection perspective classique et surtout de gérer correctement le détail. Un rendu offrant une certaine abstraction crée aussi des problèmes de cohérence par l'introduction de discontinuités : on ne peut pas simplement segmenter les images successives d'une animation pour obtenir un film cohérent.

La gestion du détail a été étudiée dans le cas du rendu réaliste pour des besoins de performance : si l'on simplifie la géométrie alors il y a moins de primitives à afficher. L'essentiel des travaux s'est ainsi concentré sur comment simplifier un objet, et comment choisir le bon niveau de detail à afficher en fonction de la position du point de vue et du temps de calcul disponible. Le rendu expressif permet d'aller plus loin en se posant la question du détail de manière plus générale. Par exemple si l'on s'intéresse à l'illustration on peut se poser la question de choisir correctement les lignes qui vont représenter au mieux un objet, de produire un dessin dans lequel la densité des traits est variable selon l'importance que l'on donne à telle ou telle partie de la scène [STR 98, GRA 04a, JEO 05], de simplifier un dessin existant tout en conservant sa structure globale [BAR 05]. Dans le cas d'un style peinture on peut vouloir abstraire plus ou moins l'image en choisissant des régions de couleur uniformes de taille variable selon un critère d'importance [SAN 02, BAR 06b].

Il nous semble que cette question de l'abstraction est aujourd'hui une des voies les plus prometteuses pour l'informatique graphique, depassant en cela le simple cadre du rendu expressif.

1.8. Bibliographie

[AGR 00] AGRAWALA M., ZORIN D., MUNZNER T., « Artistic Multiprojection Rendering », *Proceedings of the Eurographics Workshop on Rendering Techniques 2000*, London, UK, Springer-Verlag, p. 125–136, 2000.

[AGR 01] AGRAWALA M., STOLTE C., « Rendering Effective Route Maps : Improving Usability Through Generalization », FIUME E., Ed., *SIGGRAPH 2001, Computer Graphics Proceedings*, ACM Press / ACM SIGGRAPH, p. 241–250, 2001.

[AGR 03] AGRAWALA M., PHAN D., HEISER J., HAYMAKER J., KLINGNER J., HANRAHAN P., TVERSKY B., « Designing effective step-by-step assembly instructions », *ACM Trans. Graph.*, vol. 22, n°3, p. 828–837, ACM Press, 2003.

[ANJ 03] ANJYO K., HIRAMITSU K., « Stylized Highlights for Cartoon Rendering and Animation », *IEEE Comput. Graph. Appl.*, vol. 23, n°4, p. 54–61, IEEE Computer Society Press, 2003.

[APP 67] APPEL A., « The notion of quantitative invisibility and the machine rendering of solids », *Proceedings of the 1967 22nd national conference*, ACM Press, p. 387–393, 1967.

[BAR 05] BARLA P., THOLLOT J., SILLION F., « Geometric Clustering for Line Drawing Simplification », *Proceedings of the Eurographics Symposium on Rendering*, non-photorealistic rendering 2005.

[BAR 06a] BARLA P., BRESLAV S., THOLLOT J., SILLION F., MARKOSIAN L., « Stroke Pattern Analysis and Synthesis », *Computer Graphics Forum (Proc. of Eurographics 2006)*, vol. 25, 2006.

[BAR 06b] BARLA P., MARKOSIAN L., THOLLOT J., « X-Toon : An extended toon shader », *International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, 2006.

[BOU 01] BOURGUIGNON D., CANI M.-P., DRETTAKIS G., « Drawing for Illustration and Annotation in 3D », CHALMERS A., RHYNE T.-M., Eds., *Computer Graphics Forum*, vol. 20 de *EUROGRAPHICS Conference Proceedings*, EUROGRAPHICS, Blackwell Publishers, p. C114–C122, sep 2001.

[BOU 06] BOUSSEAU A., KAPLAN M., THOLLOT J., SILLION F. X., « Interactive watercolor rendering with temporal coherence and abstraction », *International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, 2006.

[BUR 05a] BURGESS J., WYVILL G., KING S. A., « A System for Real-Time Watercolour Rendering », *Computer Graphics International 2005*, 2005.

[BUR 05b] BURNS M., KLAWE J., RUSINKIEWICZ S., FINKELSTEIN A., DECARLO D., « Line Drawings from Volume Data », *SIGGRAPH '05*, August 2005.

[CHA 03] CHANG Y., SAITO S., NAKAJIMA M., « A Framework for Transfer Colors Based on the Basic Color Categories. », *Computer Graphics International*, p. 176-183, 2003.

[CHU 05] CHU N. S.-H., TAI C.-L., « MoXi : real-time ink dispersion in absorbent paper », *ACM Trans. Graph.*, vol. 24, n°3, p. 504–511, ACM Press, 2005.

[COL 04] COLEMAN P., SINGH K., « RYAN : Rendering your animation nonlinearly projected », *3rd International Symposium on Non-Photorealistic Animation and Rendering*

(NPAR'04), 2004.

[COM 99] COMANICIU D., MEER P., « Mean Shift Analysis and Applications », *ICCV (2)*, p. 1197-1203, 1999.

[CUN 03] CUNZI M., THOLLOT J., PARIS S., DEBUNNE G., GASCUEL J.-D., DURAND F., « Dynamic Canvas for Immersive Non-Photorealistic Walkthroughs », *Proc. Graphics Interface*, A K Peters, LTD., june 2003.

[CUR 97] CURTIS C. J., ANDERSON S. E., SEIMS J. E., FLEISCHER K. W., SALESIN D. H., « Computer-generated watercolor », *Siggraph 97, Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., p. 421–430, 1997.

[CUR 98] CURTIS C. J., « Loose and sketchy animation », *SIGGRAPH '98 : ACM SIGGRAPH 98 Electronic art and animation catalog*, New York, NY, USA, ACM Press, page145, 1998.

[DAN 99] DANIELS E., « Deep canvas in Disney's Tarzan », *SIGGRAPH '99 : ACM SIGGRAPH 99 Conference abstracts and applications*, New York, NY, USA, ACM Press, page200, 1999.

[DEC 03] DECARLO D., FINKELSTEIN A., RUSINKIEWICZ S., SANTELLA A., « Suggestive Contours for Conveying Shape », *ACM Transactions on Graphics, SIGGRAPH*, vol. 22, n°3, p. 848–855, July 2003.

[DEC 04] DECARLO D., FINKELSTEIN A., RUSINKIEWICZ S., « Interactive Rendering of Suggestive Contours with Temporal Coherence », *3rd International Symposium on Non-Photorealistic Animation and Rendering (NPAR'04)*, 2004.

[DEU 00] DEUSSEN O., HILLER S., OVERVELD C., STROTHOTTE T., « Floating Points : A Method for Computing Stipple Drawings », *Computer Graphics Forum (EG'00)*, vol. 19, n°3, p. 40–51, 2000.

[DRO 04] DRORI I., COHEN-OR D., YESHURUN H., « Example-based Style Synthesis », *Computer Vision and Pattern Recognition (CVPR '03)*, vol. 2, p. 143–150, 18-20 June 2004.

[DUR 02] DURAND F., « An Invitation to Discuss Computer Depiction », *2nd International Symposium on Non-Photorealistic Animation and Rendering (NPAR'02)*, Annecy, France, June 3-5 2002.

[FRE 03] FREEMAN W. T., TENENBAUM J. B., PASZTOR E., « Learning Style Translation for the Lines of a Drawing », *ACM Transactions on Graphics*, vol. 22, n°1, p. 1–14, January 2003.

[GIR 00] GIRSHICK A., INTERRANTE V., HAKER S., LEMOINE T., « Line direction matters : an argument for the use of principal directions in 3D line drawings », *NPAR 00, Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, ACM Press, p. 43–52, 2000.

[GOO 99] GOOCH B., SLOAN P.-P. J., GOOCH A., SHIRLEY P., RIESENFELD R., « Interactive technical illustration », *Proceedings of the 1999 symposium on Interactive 3D graphics*, ACM Press, p. 31–38, 1999.

[GOO 01] GOOCH B., GOOCH A., *Non-Photorealistic Rendering*, A. K. Peters, Ltd., 2001.

[GRA 04a] GRABLI S., DURAND F., SILLION F. X., « Density Measure for Line-Drawing Simplification », *12th Pacific Conference on Computer Graphics and Applications (PG'04)*, Seoul, Korea, p. 309–318, October 06 - 08 2004.

[GRA 04b] GRABLI S., TURQUIN E., DURAND F., SILLION F., « Programmable Style for NPR Line Drawing », *Rendering Techniques 2004 (Eurographics Symposium on Rendering)*, ACM Press, June 2004.

[HAE 90] HAEBERLI P., « Paint By Numbers : Abstract Image Representations », *ACM SIGGRAPH Computer Graphics*, vol. 24, n°4, p. 207–214, August 1990.

[HAL 02] HALPER N., SCHLECHTWEG S., STROTHOTTE T., « Creating Non-Photorealistic Images the Designer's Way », *2nd International Symposium on Non-Photorealistic Animation and Rendering (NPAR'02)*, Annecy, France, June 2002.

[HAY 04] HAYS J., ESSA I., « Image and Video Based Painterly Animation », *3rd International Symposium on Non-Photorealistic Animation and Rendering (NPAR'04)*, Annecy, France, p. 113–120, June 07-09 2004.

[HER 98] HERTZMANN A., « Painterly rendering with curved brush strokes of multiple sizes », *Siggraph 98, Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM Press, p. 453–460, 1998.

[HER 99] HERTZMANN A., Introduction to 3D Non- Photorealistic Rendering : Silhouettes and Outlines, Rapport, SIGGRAPH Course Notes, 1999.

[HER 00] HERTZMANN A., ZORIN D., « Illustrating smooth surfaces », *Siggraph 00, Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., p. 517–526, 2000.

[HER 03] HERTZMANN A., « A Survey of Stroke-Based Rendering », *IEEE Computer Graphics and Applications*, vol. 23, n°4, p. 70–81, July/August 2003, Special Issue on Non-Photorealistic Rendering.

[HIL 05] HILDEBRANDT K., POLTHIER K., WARDETZKY M., « Smooth Feature Lines on Surface Meshes. », *Symposium on Geometry Processing*, p. 85-90, 2005.

[INT 97] INTERRANTE V., « Illustrating surface shape in volume data via principal direction-driven 3D line integral convolution », *Siggraph 97, Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., p. 109–116, 1997.

[ISE 03] ISENBERG T., FREUDENBERG B., HALPER N., SCHLECHTWEG S., STROTHOTTE T., « A Developer's Guide to Silhouette Algorithms for Polygonal Models », *IEEE Computer Graphics and Applications*, vol. 23, n°4, p. 28–37, July/August 2003.

[ISE 05] ISENBERG T., BRENNECKE A., G-Strokes : A Concept for Simplifying Line Stylization, Rapport n°2005-780-11, Department of Computer Science, University of Calgary, Canada, April 2005.

[JEO 05] JEONG K., NI A., LEE S., MARKOSIAN L., « Detail Control in Line Drawings of 3D Meshes », *13th Pacific Conference on Computer Graphics and Applications (PG'05)*, 2005.

[JOD 02] JODOIN P.-M., EPSTEIN E., GRANGER-PICHE M., OSTROMOUKHOV V., « Hatching by Example : a Statistical Approach », *2nd International Symposium on Non-Photorealistic Animation and Rendering (NPAR'02)*, Annecy, France, June 3-5 2002.

[JOH 05] JOHAN H., HASHIMOTA R., NISHITA T., « Creating Watercolor Style Images Taking Into Account Painting Techniques », *Journal of Artsci*, p. 207-215, 2005.

[KAL 03] KALNINS R. D., DAVIDSON P. L., MARKOSIAN L., FINKELSTEIN A., « Coherent stylized silhouettes », *ACM Transactions on Graphics*, vol. 22, n°3, p. 856–861, July 2003.

[KAP 05] KAPLAN M., COHEN E., « A Generative Model for Dynamic Canvas Motion », *Computational Aesthetics, to appear*, 2005.

[LAK 00] LAKE A., MARSHALL C., HARRIS M., BLACKSTEIN M., « Stylized rendering techniques for scalable real-time 3D animation », *Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, ACM Press, p. 13–20, 2000.

[LEI 04] LEI E., CHANG C.-F., « Real-time rendering of watercolor effects for virtual environments », *PCM '04 : Proceedings of the 5th Pacific Rim Conference on Multimedia*, p. 474-481, 2004.

[LEV 98] LEVENE J., « A framework for non-realistic projections », 1998.

[LI 04] LI W., AGRAWALA M., SALESIN D., « Interactive image-based exploded view diagrams », *GI '04 : Proceedings of the 2004 conference on Graphics interface*, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, Canadian Human-Computer Communications Society, p. 203–212, 2004.

[LIT 97] LITWINOWICZ P. C., « Processing images and video for an impressionist effect », *SIGGRAPH 97*, p. 407–414, 1997.

[LLO 82] LLOYD S. P., « Least squares quantization in PCM », *IEEE Trans. on Information Theory*, vol. 28, n°2, p. 129–137, 1982.

[LUF 05] LUFT T., DEUSSEN O., « Interactive Watercolor Animations », *Pacific Graphics*, 2005.

[LUM 01] LUM E. B., MA K.-L., « Non-Photorealistic Rendering Using Watercolor Inspired Textures and Illumination », *PG '01 : Proceedings of the 9th Pacific Conference on Computer Graphics and Applications*, Washington, DC, USA, IEEE Computer Society, page322, 2001.

[LUM 05] LUM E., MA K.-L., « Expressive Line Selection by Example », *13th Pacific Conference on Computer Graphics and Applications (PG'05)*, 2005.

[MAR 97] MARKOSIAN L., KOWALSKI M. A., TRYCHIN S. J., BOURDEV L. D., GOLDSTEIN D., HUGHES J. F., « Real-Time Nonphotorealistic Rendering », *Computer Graphics*, vol. 31, n°Annual Conference Series, p. 415–420, 1997.

[MAT 05] MATSUI H., JOHAN H., NISHITA T., « A Method to Create Colored Pencil Style Images by Drawing Strokes Based on Boundaries of Regions », *Computer Graphics International (CGI'05)*, New York, p. 148–154, June 22-24 2005.

[MEI 96] MEIER B. J., « Painterly rendering for animation », *Siggraph'96, Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM Press,

p. 477–484, 1996.

[NOR 00] NORTHRUP J. D., MARKOSIAN L., « Artistic silhouettes : a hybrid approach », *Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, ACM Press, p. 31–37, 2000.

[OST 99] OSTROMOUKHOV V., « Digital facial engraving », *Siggraph 99, Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., p. 417–424, 1999.

[PAS 03] PASTOR O. E. M., FREUDENBERG B., STROTHOTTE T., « Real-Time Animated Stippling », *IEEE Computer Graphics and Applications*, vol. 23, n°4, p. 62-68, 2003.

[PRA 01] PRAUN E., HOPPE H., WEBB M., FINKELSTEIN A., « Real-time hatching », *Siggraph 01, Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM Press, page581, 2001.

[RAS 99] RASKAR R., COHEN M., « Image precision silhouette edges », *Proceedings of the 1999 symposium on Interactive 3D graphics*, ACM Press, p. 135–140, 1999.

[RAS 01] RASKAR R., « Hardware support for non-photorealistic rendering », *HWWS '01 : Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, New York, NY, USA, ACM Press, p. 41–47, 2001.

[REI 01] REINHARD E., ASHIKHMIN M., GOOCH B., SHIRLEY P., « Color Transfer between Images », *IEEE Comput. Graph. Appl.*, vol. 21, n°5, p. 34–41, IEEE Computer Society Press, 2001.

[RUD 05] RUDOLF D., MOULD D., NEUFELD E., « A Bidirectional Deposition Model of Wax Crayons », *Computer Graphics Forum*, vol. 24, n°1, p. 27–39, March 2005.

[SAI 90] SAITO T., TAKAHASHI T., « Comprehensible rendering of 3-D shapes », *Siggraph 90, Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, ACM Press, p. 197–206, 1990.

[SAL 94] SALISBURY M. P., ANDERSON S. E., BARZEL R., SALESIN D. H., « Interactive pen-and-ink illustration », *Siggraph 94, Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, ACM Press, p. 101–108, 1994.

[SAL 96] SALISBURY M., ANDERSON C., LISCHINSKI D., SALESIN D. H., « Scale-dependent reproduction of pen-and-ink illustrations », *Siggraph 96, Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM Press, p. 461–468, 1996.

[SAL 97] SALISBURY M. P., WONG M. T., HUGHES J. F., SALESIN D. H., « Orientable textures for image-based pen-and-ink illustration », *Siggraph 97, Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., p. 401–406, 1997.

[SAN 02] SANTELLA A., DECARLO D., « Abstracted Painterly Renderings Using Eye-Tracking Data », *2nd International Symposium on Non-Photorealistic Animation and Rendering (NPAR'02)*, Annecy, France, June 3-5 2002.

[SEC 02] SECORD A., HEIDRICH W., STREIT L., « Fast primitive distribution for illustration », *EGRW '02 : Proceedings of the 13th Eurographics workshop on Rendering*, Aire-la-Ville, Switzerland, Switzerland, Eurographics Association, p. 215–226, 2002.

[SMI 05] SMITH K., LIU Y., KLEIN A. W., « Animosaics », *ACM SIGGRAPH/Eurographics Symposium on Computer animation (SCA'05)*, p. 201–208, 2005.

[SOU 99] SOUSA M. C., BUCHANAN J. W., « Computer-Generated Graphite Pencil Rendering of 3D Polygonal Models », *Computer Graphics Forum (Proc. of EuroGraphics 99)*, vol. 18, n°3, p. 195–207, 1999.

[STR 98] STROTHOTTE T., *Computational Visualization : Graphics, Abstraction, and Interactivity*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1998.

[STR 02] STROTHOTTE T., SCHLECHTWEG S., *Non-Photorealistic Computer Graphics : Modeling, Rendering and Animation*, Morgan Kaufmann, 2002.

[VAN 04] VAN LAERHOVEN T., LIESENBORGS J., VAN REETH F., « Real-time watercolor painting on a distributed paper model », *Proceedings of Computer Graphics International 2004*, p. 640–643, 16-19 Jun 2004.

[WAN 04a] WANG J., THIESSON B., XU Y., COHEN M., « Image and Video Segmentation by Anisotropic Kernel Mean Shift », *ECCV*, Springer-Verlag, 2004.

[WAN 04b] WANG J., XU Y., SHUM H.-Y., COHEN M., « Video Tooning », *ACM Transactions on Graphics (proc. of SIGGRAPH 04)*, 2004.

[WIL 05] WILLATS J., DURAND F., « Defining Pictorial Style : Lessons from Linguistics and Computer Graphics », *Axiomathes*, vol. 15, n°2, 2005.

[WIN 94] WINKENBACH G., SALESIN D. H., « Siggraph 94, Computer-generated pen-and-ink illustration », *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, ACM Press, p. 91–100, 1994.

[WIN 96] WINKENBACH G., SALESIN D. H., « Rendering parametric surfaces in pen and ink », *Siggraph 96, Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM Press, p. 469–476, 1996.

[YOS 05] YOSHIZAWA S., BELYAEV A., SEIDEL H.-P., « Fast and robust detection of crest lines on meshes », *SPM '05 : Proceedings of the 2005 ACM symposium on Solid and physical modeling*, 2005.

[YU 04] YU J., MCMILLAN L., « A Framework for Multiperspective Rendering », *Eurographics Symposium on Rendering*, Norrkoping, Sweden, 2004.

[ZAN 04] ZANDER J., ISENBERG T., SCHLECHTWEG S., STROTHOTTE T., « High Quality Hatching », *Computer Graphics Forum (Proceedings of Eurographics)*, vol. 23, n°3, September 2004.

Annexe B

Principales publications

Cette annexe regroupe un ensemble de publications représentatives. Ces publications sont rangées par ordre chronologique selon la liste donnée ci-dessous. L'ensemble de mes publications est consultable sur `artis.imag.fr/Publications/`.

1. Matthieu Cunzi, Joëlle Thollot, Sylvain Paris, Gilles Debunne, Jean-Dominique Gascuel, and Frédo Durand. Dynamic canvas for immersive non-photorealistic walkthroughs. In *Proc. Graphics Interface*, pages 121–130. A K Peters, LTD., june 2003.
2. Pascal Barla, Joëlle Thollot, and François Sillion. Geometric clustering for line drawing simplification. In *Proceedings of the Eurographics Symposium on Rendering*, pages 183–192, 2005.
3. Adrien Bousseau, Matthew Kaplan, Joëlle Thollot, and François Sillion. Interactive watercolor rendering with temporal coherence and abstraction. In *International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, pages 141–149. ACM, 2006.
4. Pascal Barla, Joëlle Thollot, and Lee Markosian. X-toon : An extended toon shader. In *International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, pages 127–132. ACM, 2006.
5. Pascal Barla, Simon Breslav, Joëlle Thollot, François Sillion, and Lee Markosian. Stroke pattern analysis and synthesis. *Computer Graphics Forum (Proc. of Eurographics 2006)*, 25(3) :663–671, 2006.
6. David Vanderhaeghe, Pascal Barla, Joëlle Thollot, and François Sillion. Dynamic point distribution for stroke-based rendering. In *Rendering Techniques 2007 (Proceedings of the Eurographics Symposium on Rendering)*, pages 139–146, 2007.
7. Alexandrina Orzan, Adrien Bousseau, Pascal Barla, and Joëlle Thollot. Structure-preserving manipulation of photographs. In *International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, pages 103–110, aug 2007.
8. Adrien Bousseau, Fabrice Neyret, Joëlle Thollot, and David Salesin. Video watercolorization using bidirectional texture advection. *ACM Transaction on Graphics (Proceedings of SIGGRAPH 2007)*, 26(3) :104.1–104.7, 2007.

9. Simon Breslav, Karol Szerszen, Lee Markosian, Pascal Barla, and Joëlle Thollot. Dynamic 2d patterns for shading 3d scenes. *ACM Transaction on Graphics (Proceedings of SIGGRAPH 2007)*, 26(3) :20.1–20.5, 2007.

10. Kaleigh Smith, Pierre-Edouard Landes, Joëlle Thollot, and Karol Myszkowski. Apparent greyscale : A simple and fast conversion to perceptually accurate images and video. *Computer Graphics Forum (Proceedings of Eurographics 2008)*, 27(2) :193–200, apr 2008.

11. Alexandrina Orzan, Adrien Bousseau, Holger Winnemöller, Pascal Barla, Joëlle Thollot, and David Salesin. Diffusion curves : A vector representation for smooth-shaded images. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2008)*, 27(3) :92.1–92.8, 2008.

Dynamic Canvas for Non-Photorealistic Walkthroughs

Matthieu Cunzi [†⊙] Joëlle Thollot [†] Sylvain Paris [†] Gilles Debunne [†]
Jean-Dominique Gascuel [†] Frédo Durand [◇]
† ARTIS/GRAVIR ◇ MIT ⊙ REVES/INRIA Sophia-Antipolis

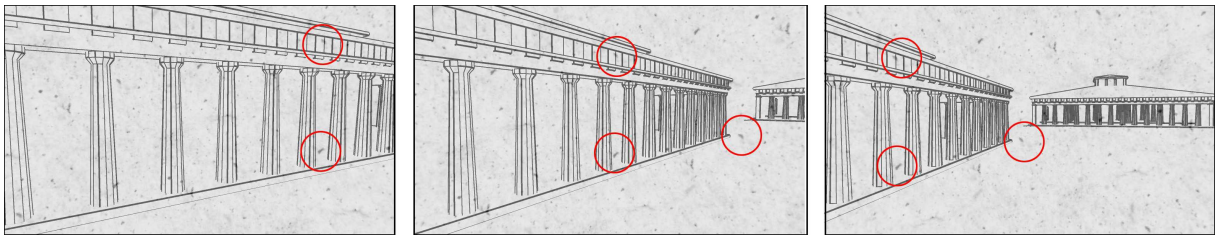


Figure 1: Walkthrough with a dynamic canvas. Note how the grain of the paper follows the strokes.

Abstract

The static background paper or canvas texture usually used for non-photorealistic animation greatly impedes the sensation of motion and results in a disturbing "shower door" effect. We present a method to animate the background canvas for non-photorealistic rendering animations and walkthroughs, which greatly improves the sensation of motion and 3D "immersion". The complex motion field induced by the 3D displacement is matched using purely 2D transformations. The motion field of forward translations is approximated using a 2D zoom in the texture, and camera rotation is approximated using 2D translation and rotation. A rolling-ball metaphor is introduced to match the instantaneous 3D motion with a 2D transformation. An infinite zoom in the texture is made possible by using a paper model based on multifrequency solid turbulence. Our results indicate a dramatic improvement over a static background.

1 Introduction

The field of *Non-Photorealistic Rendering* [GG01, SS02, Rey02, Dur02] not only captures the qualities of traditional media, but also permits their animation. Media that were inherently static come to life and can be animated and used for interactive walkthrough. This raises a number of challenges: How should the elements of the picture such as strokes (marks) or background paper be animated, and how can we ensure *temporal coherence*? Two basic strategies are possible, and neither of them is perfect. One can either attach the marks to the 2D space of the picture, or attach them to the 3D objects. In the first case, the scene appears to be viewed through a shower door, and in the second, the 3D objects seem to be textured with, or carved in, artistic strokes. The problem boils down to the tension caused by the dualism of pictures, both 2D compositions and representations of 3D scenes [Dur02].

In previous work, much attention has been paid to strokes and their temporal coherence, but the temporal behavior of the background canvas or paper has been mostly ignored. As a result, most NPR animations or walkthroughs seem to be projected on a paper or canvas texture using a slide projector, and the background does not participate in the animation or walkthrough experience. The strokes slide on the paper, which not only reduces the "immersion" and motion cues, but also impedes the sense of the picture as a whole, because paper and strokes do not interact and seem to lie in two different dimensions.

In this paper, we present a *dynamic canvas* where the background texture is animated to provide strong motion cues and bind the background and moving strokes. It dramatically improves the "immersive" impression and motion cues for non-photorealistic walkthroughs, and dramatically reduces the effect of strokes that slide on the background. Our method presents a careful balance between the 2D qualities of the background texture and the 3D motion of the observer. This is achieved by panning around and zooming in a 2D background paper texture in order to approximate 3D motion. The problem can be stated as follows: The motion of the observer is three-dimensional and results in a complex optical flow, including parallax effects. In contrast, the canvas or paper of a picture is characterized by its flat and rigid 2D quality. Our goal is to use a rigid 2D motion for the paper in picture space that provides a perceptual impression of motion as close as possible to the 3D displacement in object space.

1.1 Related work

The issues of animation and temporal coherence have received much attention in non-photorealistic rendering. We review the techniques most relevant to our approach, and we refer the interested reader to the books covering the field [GG01, SS02].

Meier [Mei96] rendered animations of 3D models in a painterly style using particles. Related techniques were used in the DeepCanvas system at Disney, where the brush strokes drawn by animators were attached to a 3D model [Dan99]. Curtis also uses particles to create loose and sketchy animations [Cur98]. Some techniques process video sequences into a painterly style. Temporal coherence can be ensured by using, e.g., optical flow [Lit97] or image differences [HP00].

For interactive NPR applications, temporal coherence and constant screen-space size of marks can be obtained by an appropriate use of the hardware mipmapping capabilities, as proposed by Klein et al. [KKL+00] and later refined by Praun et al. for pen-and-ink styles [PHMF01, WPFH02]. To ensure that the screen-space size of their mark remains relatively constant, they double the size of the marks for each mipmap level. Our solution for forward translation is related to their technique, in that our paper texture is self-similar and ensures a constant screen size of the grain.

The interaction between the paper and the marks has been carefully simulated or imitated, e.g., [LMHB00, MG02, SB00, CAS+97], but to the best of our knowledge, no previous work has addressed paper motion for NPR walkthroughs with arbitrary camera motion.

The technique most related to our approach is the work on multiperspective panoramas by Wood et al. [WFH+97]. Inspired by traditional cel animation, they build a single 2D background to simulate a complex 3D path. The effect is obtained by sliding a rectangular window on a multiperspective panorama. The non-linear projection is computed by matching the optical flow of the simulated 3D path using a least-squares approach. In this paper, we also propose to approximate complex 3D movements with 2D transformations on a background texture, but in an interactive context for arbitrary paths.

1.2 Method overview

Our goal is to preserve the 2D quality of the background texture while providing compelling 3D motion cues and avoiding the effect of strokes that seem to "slide" on the paper. The exact definition of "2D quality" will be described shortly and is one of the key decisions that influences the motion depiction style. The camera motion usually results in a complex optical flow that does not correspond to a simple 2D transformation. Matching the 2D paper motion with the complex movement of the 3D observer or with object motion is therefore usually overconstrained. The grain of the paper cannot follow the 3D points on the scene objects without distorting the paper texture. Therefore, a compromise has to be chosen, and the concessions will depend on the task and style. We propose a solution that is a good tradeoff for most situations, but the infinite range of possible strategies is an example of the richness of non-photorealistic styles.

Our method can be intuitively expressed for the two most basic types of observer motion: camera rotation and forward translation. When the camera is translating forward, we exploit the ambiguity between 2D zoom and forward translation. We zoom in the background texture, which provides a strong forward motion cue. In fact, this approximates the visual impression of moving in a snow storm or in the fog. Note that in contrast to Wood et al. [WFH+97], we use the zoom not only to simulate a change in the focal length, but also to simulate forward translation. In order to implement our zooming approximation of translation, we need to be able to infinitely zoom in the background texture. This will be described in Section 5.

When the camera rotates, the projective picture (i.e. the picture projected on the sphere of directions) should not be altered since the eye position is fixed. In particular the strokes used to draw the 3D model should not seem to slide on the background paper. The screen motion of the paper should match the screen motion of the 3D model induced by the camera rotation as much as possible. We therefore want to rotate the background texture as if it were projected on a sphere centered on the viewpoint. It is well known that, unfortunately, texture mapping on a sphere raises singularity problems at the poles. Moreover, combining the 2D zoom for translation with the spherical rotation is far from trivial. Indeed, the zoom is inherently a linear transformation of the Cartesian plane. In contrast, the rotation is defined on the sphere of directions, which produces non-linear transformations in the plane. Both the topology (infinite plane vs. sphere) and the linearity issue make the compatibility between the two approaches difficult. This is why we developed an approximation strategy to map the entire technique to the Cartesian 2D plane.

Following Wood et al. [WFH+97], we introduce a general framework for the study of temporal coherence in NPR, using the notion of *motion field*. This allows us to formally justify our method and to numerically link the 2D transformation of the background and the 3D motion.

This paper makes the following contributions:

- We show how simple 2D transformations of the background paper can dramatically enhance the motion cues for interactive NPR walkthroughs.

- We introduce the *rolling-ball* metaphor to match the spherical trajectory of the center of the screen to a Cartesian 2D displacement.
- We warp the result of the above motion to better match the motion cues for camera rotations.
- We introduce a technique to perform an infinite zoom into a 2D texture defined procedurally or using a scanned image.

The paper is organized as follows: In Section 2, we introduce the framework of motion fields. In Section 3, we propose a simple technique to approximate the motion field of a 3D displacement with 2D similarity transforms. In Section 4 we introduce a screen-space spherical warp in order to better match the 3D motion cues for rotations. Section 5 presents our self-similar paper model that permits infinite zooms. We describe our implementation and results in Section 6 before concluding.

2 Motion field and 3D motion cues

The *motion field* [Hor86] represents the time derivative of the 2D projection of a 3D point. The motion field is slightly different from the optical flow, which is derived from image brightness, while the motion field is purely geometric. As shown by the landmark work by Gibson, [Gib79] the motion field and optical flow are crucial visual cues for moving observers. The qualitative pattern of the motion field is a strong indication of the motion of the observer (*egomotion*), or of the motion of dynamic objects in the scene. In this paper, we focus on the motion of the observer, but this framework can be used to study temporal coherence with any motion.

To express the motion field equations [Hor86], we consider a point $M = (X,Y,Z)$ in a camera coordinate system that projects on a screen at distance f on a point m with coordinates (x,y,f).

We first compute the motion field for a translation along the Z axis at speed $\frac{dZ}{dt}$. From the classical relations $x = \frac{fX}{Z}$ and $y = \frac{fY}{Z}$, we get:

$$\begin{aligned} \frac{dx}{dt} &= -\frac{f\,X}{Z^2}\frac{dZ}{dt} = -\frac{x}{Z}\frac{dZ}{dt} \\ \frac{dy}{dt} &= -\frac{f\,Y}{Z^2}\frac{dZ}{dt} = -\frac{y}{Z}\frac{dZ}{dt} \end{aligned} \tag{1}$$

The motion field for the forward translation towards a vertical plane parallel to the image plane is depicted in Fig. 3(a). It is a radial field. In contrast, the motion field for the translation towards a more complex scene, such as a sphere and a plane (Fig. 3(c)) exhibits a more involved pattern and discontinuities along silhouettes. Such motion fields typically cannot be matched with simple 2D transformations.

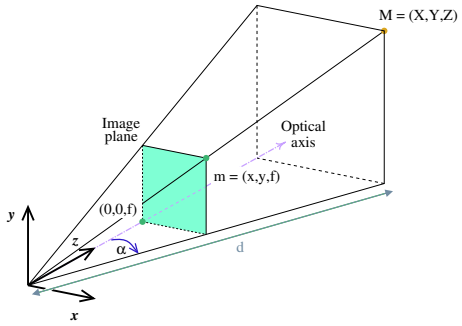


Figure 2: Motion field calculation for a pinhole camera.

We also compute the motion field for a rotation around the Y axis at speed $\frac{d\alpha}{dt}$ (see Fig. 2). We introduce the 3D distance $d = \sqrt{X^2+Z^2}$ and obtain the relation $Z = d\cos\alpha$. We can express both x and y as function of α:

$$x = f\,\tan\alpha \qquad\qquad y = \frac{f\,Y}{Z} = \frac{f\,Y}{d\cos\alpha}$$

and obtain the final result:

$$\frac{dx}{dt} = \left(f + \frac{x^2}{f}\right)\frac{d\alpha}{dt} \qquad\qquad \frac{dy}{dt} = \frac{xy}{f}\frac{d\alpha}{dt}$$

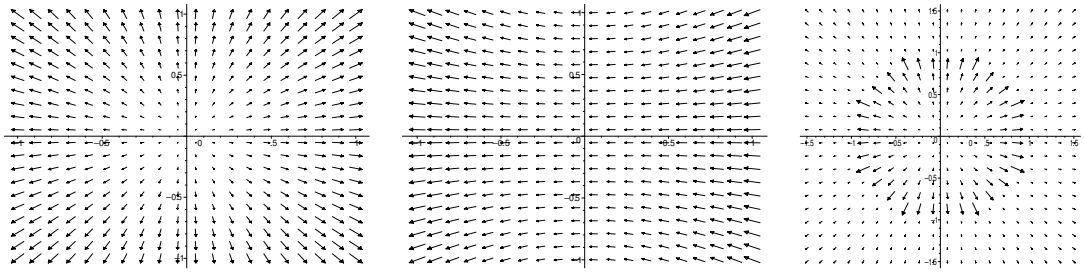
Figure 3(b) shows a motion field for a rotation.

3 Matching the motion field with 2D similarity transforms

We want to match or approximate such motion fields by the motion field induced by simpler 2D transformations. This will permit the animation of the background paper or canvas in a 2D fashion, while providing the user with convincing 3D motion cues. We need to define what "2D quality" or "simple 2D transformation" mean, and therefore which constraints must be enforced on the motion. In contrast to Wood et al. [WFH+97], our approximation is computed directly from the motion and is not a least-square optimization.

The most immediate choice is to allow only 2D *similarity* transformations [WFH+97]. Similarity transforms are composed of a rigid transformation and an isotropic scale. They completely respect the rigid 2D nature of the background. We first discuss the simple cases of translation along the optical axis and rotations around the vertical axis, before discussing the general case.

The motion field of a 3D forward translation towards a vertical plane can be perfectly matched by a zoom in the background, that is, by a 2D scaling, as suggested by Fig. 3(a). However, we cannot match a complex motion field such as the one shown in Fig. 3(c) with a simple 2D transform. Our technique chooses a *subjective distance*,



(a) (b) (c)

Figure 3: (a) Motion field for a forward motion towards a vertical plane. (b) Motion field for a camera rotation. (c) Motion field for a forward translation towards a sphere in front of a plane. Note the discontinuity and parallax effect at the silhouette of the sphere.

D, for which the motion field is correctly matched. For example, in Fig. 3(c), we can choose to match the motion field of the sphere or the motion field of the background plane. We propose two strategies to choose this subjective distance. It can be set constant, to approximate a "fog" at a given distance, or it can track a given object of interest.

The motion field for a camera rotation along the vertical axis (Fig. 3(b)) cannot be easily approximated, since it describes hyperbolae. However, if the field of view is not too wide, these hyperbolae are very flat and close to horizontal lines. The similarity that can best approximate such motion fields is a horizontal translation. In Section 4, we will introduce a spherical warp to compensate for this approximation.

In fact, the problem we are trying to solve for camera rotations can be simply visualized by considering the sphere of directions, that is, an arbitrary sphere centered on the observer. We can map the sphere of directions with a paper texture to perfectly respect the camera rotation. But we choose to approximate such a sphere rotation by a planar motion. In addition to alleviating the pole problem, it respects the 2D quality of the paper better, and it makes it possible to combine this transformation with our 2D zoom.

3.1 Matching the translation

We decompose the translation of the observer into two components: one along the optical axis, and one translation parallel to the plane of the image.

Translation along the optical axis

The forward translation along the optical axis is matched with a 2D zoom. In order to numerically relate them, we study the motion field (i.e. the screen velocity) of a 2D zoom. If the camera is translating towards a plane at distance D, from (1) we have:

$$\frac{dx}{dt} = -\frac{x}{D}\frac{dZ}{dt} \quad \text{and} \quad \frac{dy}{dt} = -\frac{y}{D}\frac{dZ}{dt} \tag{2}$$

To obtain the relation between two consecutive frames t and $t+\Delta t$, we consider a camera translating at constant speed $\frac{dZ}{dt} = \zeta$ during this short Δt. We simplify (2) and obtain $\frac{dx}{dt} = k\,x$ and $\frac{dy}{dt} = k\,y$ with $k = -\frac{\zeta}{D}$, which leads to the classical differential equation solution:

$$\begin{pmatrix} x(t+\Delta t) \\ y(t+\Delta t) \end{pmatrix} = e^{k\Delta t} \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} \tag{3}$$

This shows that we can match the motion field exactly with a zoom at exponential rate k.

$\zeta\,\Delta t = \Delta Z$ is the signed distance corresponding to the forward translation between the two frames. We therefore have the final relation:

$$\begin{pmatrix} x(t+\Delta t) \\ y(t+\Delta t) \end{pmatrix} = e^{-\frac{\Delta Z}{D}} \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} \tag{4}$$

In the above equations, the *subjective distance* D, plays a major role. The motion field of objects at distance D is correctly matched. It relates the zoom rate to the 3D scene, and a shorter subjective distance results in a faster motion cue. The choice of the subjective distance is an important decision. It is application- and style-dependent.

Translation parallel to the image plane

The translation component parallel to the image plane at speed $(\frac{dX}{dt}, \frac{dY}{dt})$ is matched with a 2D translation where the translation vector has been scaled using the perspective ratio at the subjective distance D:

$$\frac{dx}{dt} = -\frac{f}{D}\frac{dX}{dt} \qquad \frac{dy}{dt} = -\frac{f}{D}\frac{dY}{dt}$$

For a general translation, this results in off-center zooms because of the composition of the translation and

the zoom. In this case, the final zoom center lies on the vanishing point of the translation direction, which is what we would expect intuitively.

3.2 The rolling-ball metaphor for camera rotation

For camera rotation, we decide to perfectly match the motion of the center of the image. We consider the Cartesian texture plane as tangent to the sphere of directions, with the tangent point at the center of the screen[1]. Our method locally approximates the sphere by this tangent plane (Fig. 4). The motion then corresponds to the sphere "rolling without sliding" on the plane, a classical situation in mechanics, e.g., [Fre65]. Matching the motion field for the center point between the sphere of direction and the texture plane means that the velocity on the sphere and on the plane are equal. This is the definition of rolling without sliding (a.k.a. without slipping). For example, we show in Fig. 4 that if the camera rotates around the vertical axis, the contact point on the texture plane is translated in the opposite direction. The trajectory of the center of the image in the texture plane is simply a straight line. This case is related to the cylindrical projection used in cartography to map the Earth, e.g., [Den96]. Note that after a camera rotation of 360 degrees, the background is usually not the same, unless the texture happens to be tileable with a period equal to the length of the equator of the rolling ball.

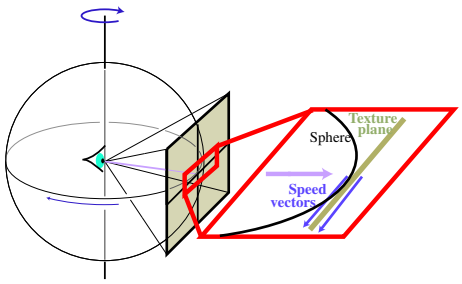


Figure 4: A ball rolling without sliding. The speed vectors of the sphere contact point and of the texture plane contact point are equal.

If the rotation axis is not orthogonal to the optical axis, the rolling motion describes a circle in the texture plane (Fig. 5a). The center of this circle is the intersection C of the rotation axis with the texture plane. The 2D transformation is the rotation around this intersection point. This corresponds to the conical projection in cartography [Den96]. Similar to the previous case, the background after a rotation of 360 degrees is usually not the same, because the length of the parallel circle of the sphere tangent to the texture plane is usually not the length of the circular trajectory in the texture plane (see Fig. 5a). This point will prove important in the next section.

As a special case, if the rotation axis is the optical axis (Fig. 5b), then the 2D texture plane undergoes the same rotation along the center of the image as the spherical rotation.

Figure 5: (a) A ball rolling on a plane with rotation axis intersecting the plane. (b) Extreme case where the optical and rotational axis are identical.

In the general case, we consider the instantaneous rotation between two adjacent frames. We use instantaneous Euler angles in the local frame of the observer. This does not suffer from the singularities of the Euler angles because we recenter the Euler axis for each frame, and because the rotation between two frames is usually less than 90 degrees. The details of the formulae for the 2D rigid transformation approximating camera rotation are given in appendix A.

4 Spherical warp

In the previous section, we have shown how zooming and the rolling-ball metaphor can approximate the motion field with simple 2D similarity transformations. The trajectory of the picture center is matched exactly. We now introduce a 2D warp that improves the approximation for the rest of the picture in the case of camera rotation. We focus on rotations that keep the horizon level. That is, the camera is allowed to rotate only along the vertical axis of world space (left-right rotation), or along the camera horizontal axis (up-down rotation). Our warp results in a perfect match for left-right camera rotation, and very good approximations for up-down rotation.

4.1 Basic warp

As discussed in Section 2, the motion field and the trajectories describe conics on the screen during a camera rotation (Fig. 3b). Moreover the speed varies along these conics. In our rolling-ball approximation, the trajectories are circles (with center the intersection of the rotation axis and the texture plane, see Fig. 5a) or lines, and they have constant speed. Our warp therefore maps circles or lines

[1] formally speaking, it is tangent to an embedding of the spheres of direction with radius f

in the 2D texture as obtained from Section 3.2 to the appropriate conics with varying speed.

Since we want to convey the impression that the paper texture is applied on a sphere centered on the camera, the warp is conceptually decomposed into two steps: the texture is mapped onto the sphere, and the sphere points are then mapped onto the screen (Fig 6). The trajectories on the sphere are parallel circles (latitude=cte on the Earth).

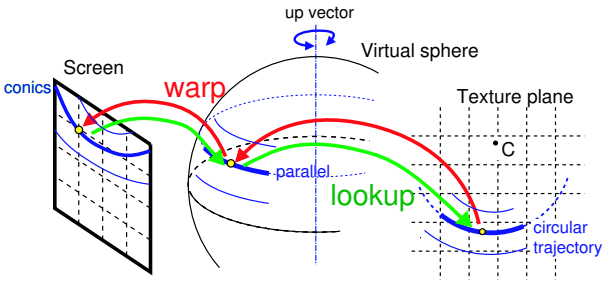


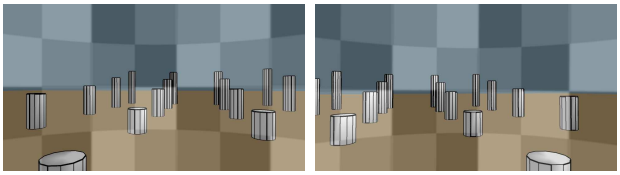
Figure 6: A warp in two steps: a screen point is mapped onto a point on a sphere parallel which is then locally mapped onto a circle in the texture plane.

In practice, we need to perform the inverse mapping operation (lookup), going from a screen point to a parallel on the sphere and finally to a circle on the texture plane. The first step is a simple spherical coordinate computation that goes from a screen point (x,y) to spherical coordinates (θ,φ), where $\varphi = cte$ describes the parallel and θ is the location on the parallel. Since we use the up vector as a reference, the center of the screen is at $(\varphi = \alpha, \theta = 0)$, where α is the tilt of the camera (elevation from the horizon).

For the second step, we map a parallel circle $\varphi = cte$ to a trajectory (circle or line) in the the texture (Fig. 6, right). Detailed formulae and illustrations are given in appendix B and Fig. 12. Note that for a parallel, we can choose any texture circle as long as the parallels are monotonously mapped to the circles. The meridians on the sphere defined by $\theta = cte$ corresponds in the texture to lines going through the intersection C of the rotation axis and the texture plane. We have one remaining degree of freedom: we need to decide which texture circle is mapped to which parallel.

We use this degree of freedom to optimize the motion field for up-down rotation. The spacing of the texture-space circle is chosen so that the motion field of the points of the vertical line in the center of the picture is matched perfectly. This also makes the pattern of vertical velocities for the whole picture qualitatively correct. That is, from top to bottom, the points slow down and then accelerate. The detailed formulae are given in appendix B.

The result of this warp is illustrated in Fig. 7 with a checkerboard texture for better visualization. The effect is best seen in the accompanying video published on the graphics interface web site.



(a) (b)

Figure 7: Between (a) and (b), the observer is looking at the right and thus the checkerboard moves to the left. Note the distortion of the texture such that no sliding occurs.

4.2 Case of closed trajectories

The above warp assumes that no screen trajectory is a closed curved (ellipse). Indeed, recall that when rotating by 360 degrees, the trajectory in the texture is likely not a full circle. This is not a problem with the basic rolling-ball technique because the texture is displayed "as is" after the similarity transform. But when combined to the warp, it can lead to discontinuities on the ellipse trajectories. This occurs when the vanishing point of the vertical direction (corresponding to C in the texture) is visible on the screen. For open trajectories, the problem does not occur, in a sense because the discontinuity happens off-screen.

We however note that no warp is required in the extreme case where C is in the center of the screen, that is, when the viewer is looking completely up or down. In this case, the rolling ball results in the exact motion field. Intuitively, when the observer is looking higher and higher, the screen trajectories transition from hyperbolae to ellipses and then to circles. Therefore, the circular trajectories described by the rolling ball become better and better approximations, and the warp is not required.

We therefore progressively turn the warp off when the observer is looking up. In practice, we simply use a $\sin^2$ interpolation that reaches zero when the vanishing point of the vertical reaches the screen.

5 Procedural zoomable paper

In order to produce our approximate motion field for forward translation, we need to be able to infinitely zoom in the paper texture. We want to ensure that for any zoom level, the texture looks like paper. This is related to self-similar fractals, e.g., [Hut81, Bar88], or to super-

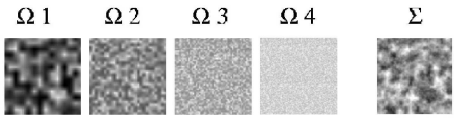


Figure 8: Successive octaves in a Perlin noise. On the right we show the summation of the four octaves.

resolution, e.g., [BK00]. We also need to be able to apply arbitrary rotations and translations. The latter can be obtained using tileable texture. We present a solution based on procedural solid textures [Per85]. We show that an infinite zoom can be obtained by cyclically shifting the frequency spectrum of a Perlin turbulence, and that the method can be applied to scanned textures as well.

5.1 Static paper texture

Our basic technique simulates a simple paper grain using a Perlin turbulence. The principle of solid turbulence is to synthesize and render complex noise using a sum of signals with different frequencies or *octaves* [Per85] (Fig. 5.1). We define a *base frequency* F_1 and a *base amplitude* A_1 for the first octave Ω_1. The amplitude intuitively corresponds to the "hardness" of the paper texture and the frequency to the size of the paper grain. Each octave Ω_i is defined from the one preceding it by multiplying the frequency by 2: $F_{i+1} = 2F_i$ and dividing the amplitude by 2: $A_{i+1} = \frac{1}{2}A_i$.

We use the method by Miné and Neyret [MN99] to implement procedural textures on graphics hardware. They use multipass texture mapping to sum a random noise texture at different scales. On modern graphics hardware, the availability of multiple texture accesses per pass can be used to avoid multiple passes. In practice, since the size of the largest-frequency octave is usually smaller than the screen, we compute this blending off-screen for one tile and store it in texture memory. All the octaves used for the texture are calculated using the same noise texture. In order to hide the grid-structure of the texture, we add a random translation to each newly created octave. An additional random rotation could further reduce regularity.

5.2 Infinite zoom using a frequency shift

In order to zoom in or out the paper, we continuously shift the frequencies of the octaves. To achieve an infinite zoom, we add new octaves in the low frequencies (zoom in) or in the high frequencies (zoom out). The addition of these frequencies is made smooth using an envelope, in practice a simple linear blend for the highest and lowest octave. Our technique is very similar to the classic audio illusion that creates a sound with an ever-ascending

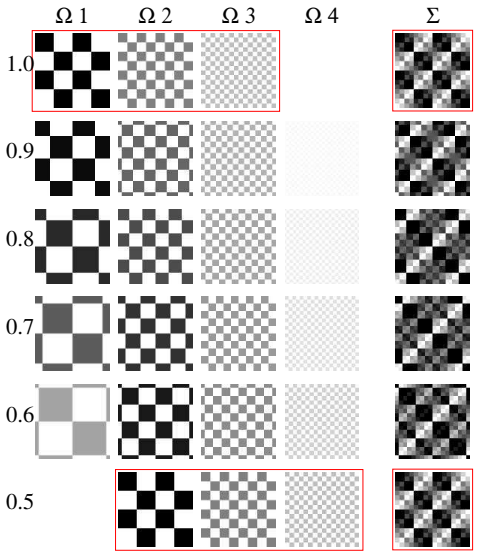


Figure 9: Illustration of infinite zoom on a checkerboard texture. The zoom factor is shown on the left. Note the inclusion of lower frequencies when the zoom factor increases. On the last line we can see that octave Ω_i and the sum Σ are the same as Ω_{i-1} and Σ of the first line (red boxes). We are thus ready to wrap-around.

or ever-descending pitch introduced by Shepard [She64]. It is also similar to the classical stairs by M.C. Escher. It is a shift in frequency space of a repetitive spectrum controlled by a frequency envelope.

When the observer moves forwards at a rate dZ/dt, the texture is scaled by a zoom factor: $zoom = zoom * \frac{D-\Delta Z}{D}$ (Eq. 4 first order approximation). This *zoom* factor *decreases* when we zoom in since a smaller portion of the image must then fill the screen (see Appendix A).

In screen space, each octave is scaled by this zoom factor, which corresponds in the frequency domain to a scale of all the frequencies of the octaves. The amplitudes have to be tuned accordingly (divided by *zoom*) in order to preserve the properties of the resulting texture.

Initially *zoom* is set to 1. During a forward translation the value of *zoom* decreases (zoom in). When *zoom* becomes equal to $\frac{1}{2}$, each octave Ω_i satisfies: $F_i(zoom = \frac{1}{2}) = \frac{1}{2}F_i(zoom = 1)$ and $A_i(zoom = \frac{1}{2}) = 2A_i(zoom = 1)$. This corresponds to a shift of the octaves.

In order to introduce new octaves in the high frequencies and to suppress the very low frequencies introduced, we shift the octaves cyclically: each octave Ω_{i+1} becomes Ω_i and the first octave becomes the last one. Smoothness is ensured by linearly blending the first and last octaves. This can be seen as our frequency envelope. The process is illustrated in Fig. 9 with a checkerboard

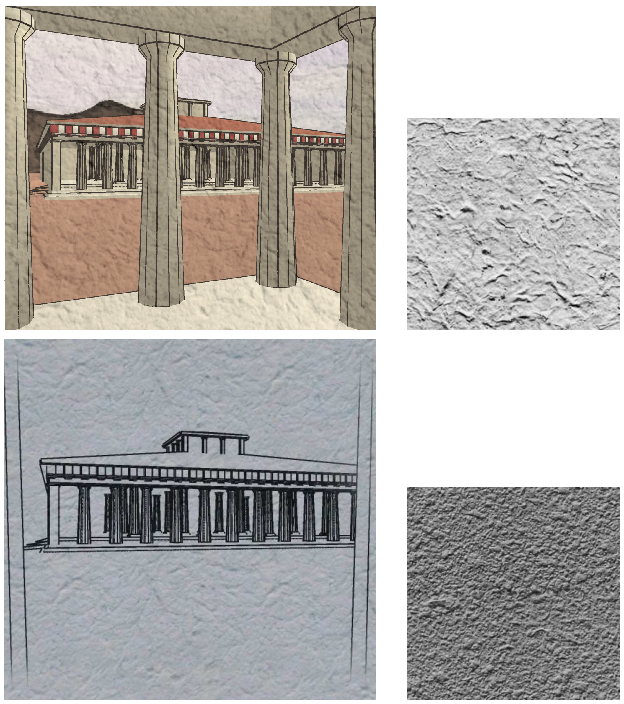


Figure 10: Different scanned papers used with our infinite zoom technique. We show a screenshot and the corresponding paper texture (with contrast enhanced for clarity.)

texture for illustration and with an actual Perlin noise. Note that the increase in the amplitude of low frequency octaves is counteracted by the frequency envelope.

At this point, we reinitialize *zoom* to 1 and we continue the forward motion in the same way. We obtain an *infinite zoom*, by compositing a fixed number of octaves. Just as in the static case, we randomly translate the newly created octave to avoid cyclic period appearance.

For backward translation, *zoom* increases and when it reaches 1, the same shift of the octaves can be applied and zoom is reset to $\frac{1}{2}$.

5.3 Using real textures

A scanned image can also be used for the octaves. We use the same technique: each octave is calculated by zooming in the original picture with a factor of 2. This permits the use of scanned paper images as shown in Figure 1 and 10.

Note that when used with the rolling-ball technique, the resulting texture may be displayed rotated on the screen. If the texture is anisotropic, the vertical direction might not be preserved. This is however a problem only with textures such as graph paper.

6 Implementation and results

The paper texture is rendered using a textured polygonal mesh. The texture coordinates are computed using the 2D transformation given by the zoom and rolling-ball technique (see Sec. 3) and modified by the spherical distortion described in Sec. 4. For each frame, we pre-render the composition of 3 to 5 octaves in a 256x256 texture tile to generate the zoomable canvas. By varying the base amplitude A_1 and lowest frequency F_1, we can modify the hardness and sharpness of the texture.

The technique described can be used with any non-photorealistic rendering method. We have implemented it in an NPR walkthrough system that uses line drawing and a simple paper-mark interaction model. When the objects are filled with color, we use a multiplicative blend between the paper color and the object color. Other paper models [LMHB00, MG02, SB00, CAS+97] could also greatly benefit from our background animation. Several paper types have been tested (Perlin noise and scanned images) and provide convincing results. The scanned image needs to be periodic, and better results are obtained if it is reasonably isotropic.

The zoom depends on the subjective distance D, in order to match the canvas zoom speed to the 3D objects located at this distance. A constant value works well for walkthrough applications where no special object should stand out. The strokes slide a little on the canvas for objects at a different distance, but we did not find it disconcerting. When a special object is used from the outside in, we use the distance to the object as subjective distance. In this case, the dynamic canvas not only provides strong motion cues, but it also helps focus the attention onto the object of interest.

The method provides a dramatic improvement for NPR walkthroughs, as demonstrated by the accompanying video. The strong motion cues induced by the motion field are very effective at reinforcing the immersion. The strokes almost do not slide on the dynamic canvas, which greatly improves the impression of the picture as a whole. And because the canvas undergoes only 2D transformation, we preserve the 2D quality of the drawing.

Preliminary feedback by artists and archaeologists were very promising. As described by Strothotte et al. [SSRL96] in the context of architecture, the major advantage of non-photorealistic rendering for archaeological walkthroughs is that it emphasize that the displayed scene is only an hypothesis. An NPR walkthroughs allows archaeologists to use both the strength of 3D graphics for model exploration and their traditional drawing style. In this context, our dynamic canvas provides strong motion cues and reinforces the 3D immersion, while respecting the traditional 2D qualities of the drawing.

7 Conclusions and future work

We have presented a method that animates the background canvas for NPR walkthroughs. Using simple 2D transformations and a spherical distortion, we approximate the complex motion field produced by a 3D displacement. This provides compelling motion cues and greatly improves the "immersive" impression for NPR walkthroughs.

This work opens several directions of future research. We are working on more advanced paper-stroke models. The subjective distance could be computed on the fly in an "autofocus" fashion. The choice of paper motion might be influenced by the medium used for the marks. For example, some marks are mostly opaque (e.g., oil painting) and the background canvas might be visible only in some places.

The rolling-ball metaphor could also be used to create multiperspective panoramas [WFH+97]. In the case of off-line animation, the optimization approach by Wood et al. could be applied to the motion of the background texture to obtain a better approximation. More generally, we believe that the framework of motion fields and their approximation with 2D transform has great potential for NPR animation. We are planning to apply it to marks such as brush or pencil strokes, and also to larger regions such as watercolor effects. In these cases, the problem is more complex because marks need to be more strongly attached to the depicted 3D objects.

Acknowledgments

This research was funded in part by the INRIA Action de Recherche Coopérative ARCHEOS (http://www-sop.inria.fr/reves/Archeos).

Thanks to Fabrice Neyret and Samuel Hornus for numerous discussions and observations. Thanks to Ray Jones, Sara Su and Victor Ostromoukhov for proofreading the paper.

References

[Bar88] Michael Barnsley. *Fractals Everywhere*. Academic Press, 1988.

[BK00] S. Baker and T. Kanade. Limits on super-resolution and how to break them. In *Proc. of CVPR*, 2000.

[CAS+97] C. Curtis, S. Anderson, J. Seims, K. Fleischer, and D. Salesin. Computer-generated watercolor. *Proc. SIGGRAPH*, 1997.

[Cur98] C. Curtis. Loose and Sketchy Animation. In *SIGGRAPH: Conf. Abstracts and Applications*, 1998.

[Dan99] Eric Daniels. Deep canvas in disney's tarzan. In *ACM SiGGRAPH sketch and applications*, 1999.

[Den96] Borden D. Dent. *Cartography: Thematic Map Design*. WCB/McGraw-Hill, 4th edition, 1996.

[Dur02] F. Durand. An invitation to discuss computer depiction. In *Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, 2002.

[Fre65] A. P. French. *Newtonian Mechanics*. W. W. Norton Company, 1965.

[GG01] Gooch and Gooch. *Non-Photorealistic Rendering*. AK-Peters, 2001.

[Gib79] J. J. Gibson. *The Ecological Approach to Visual Perception*. Houghton Mifflin, 1979.

[Hor86] B. P Horn. *Robot Vision*. MIT Press, Cambridge 1986, 1986.

[HP00] A. Hertzmann and K. Perlin. Painterly rendering for video and interaction. In *Non-Photorealistic Animation and Rendering*, 2000.

[Hut81] J. Hutchinson. Fractals and self-similarity. *Indiana Univ. J. of Mathematics*, 30:713–747, 1981.

[KKL+00] A. Klein, M. Kazhdan, W. Li, W. Toledo Corra, A. Finkelstein, and T.Funkhouser. Non-photorealistic virtual environments. *Proc. SIGGRAPH*, 2000.

[Lit97] P. Litwinowicz. Processing images and video for an impressionist effect. *Proc. SIGGRAPH*, 1997.

[LMHB00] A. Lake, C. Marshall, M. Harris, and M. Blackstein. Stylized rendering techniques for scalable real-time 3d animation. In *Non-Photorealistic Animation and Rendering*, 2000.

[Mei96] B. Meier. Painterly rendering for animation. *Proc. SIGGRAPH*, 1996.

[MG02] A. Majumder and M. Gopi. Hardware accelerated real time charcoal rendering. In *NPAR 2002: Symposium on Non Photorealistic Animation and Rendering*, June 2002.

[MN99] A. Miné and F. Neyret. Perlin textures in real time using OpenGL. Technical report, RR-3713, INRIA, 1999. http://www-imagis.imag.fr/Membres/Fabrice.Neyret/publis/RR-3713-eng.html.

[Per85] Ken Perlin. An image synthesizer. In *Computer Graphics (Proc. SIGGRAPH 85)*, volume 19, pages 287–296, July 1985.

[PHMF01] E. Praun, H. Hoppe, M., and A. Finkelstein. Real-time hatching. *Proc. SIGGRAPH*, 2001.

[Rey02] C. Reynolds. Stylized depiction in computer graphics non-photorealistic, painterly and 'toon rendering. http://www.red3d.com/cwr/npr/, 2002.

[SB00] M. Costa Sousa and J. Buchanan. Observational models of graphite pencil materials. *Computer Graphics Forum*, 19(1):27–49, March 2000.

[She64] R. Shepard. Circularity in judgments of relative pitch. *J. Acoust. Soc. Am.*, 36:2346–2353, 1964.

[SS02] Thomas Strothotte and Stefan Schlechtweg. *Non-Photorealistic Computer Graphics. Modeling, Rendering, and Animation*. Morgan Kaufmann, San Francisco, 2002.

[SSRL96] J. Schumann, T. Strothotte, A. Raab, and S. Laser. Assessing the effect of non-photorealistic rendered images in cad. In *Proceedings of ACM CHI*, 1996.

[WFH+97] D. Wood, A. Finkelstein, J. Hughes, C. Thayer, and D. Salesin. Multiperspective panoramas for cel animation. *Proc. SIGGRAPH*, 1997.

[WPFH02] M. Webb, E. Praun, A. Finkelstein, and H. Hoppe. Fine tone control in hardware hatching. In *NPAR 2002: Symposium on Non Photorealistic Animation and Rendering*, 2002.

A Texture space transformation from 3D rotation

We describe the 2D transformation that approximates a 3D camera rotation. This rotation is defined by an instantaneous axis Ω and an incremental angle $\Delta\alpha$.

The paper texture coordinates are in the range $[-zoom, zoom]$ for the vertical direction, and are scaled by the screen aspect ratio in the horizontal direction. The radius of the rolling ball is then defined by:

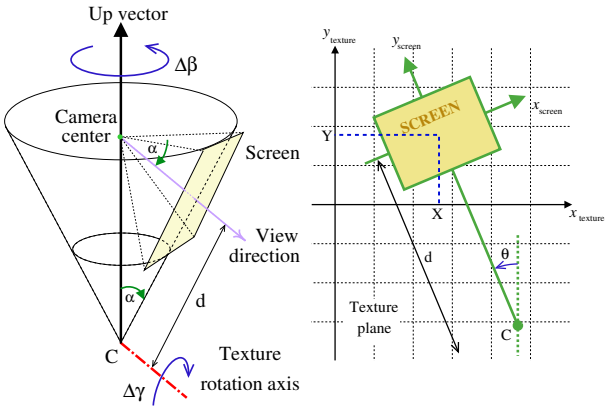
$$R = \frac{zoom}{\tan(\frac{1}{2} fov)}$$

where fov is the camera vertical field of view.

We first consider the case where Ω is orthogonal to the viewing direction (Fig. 4). The texture coordinates then have to be translated in a direction orthogonal to Ω, by an offset $\delta = R\,\Delta\alpha$, the distance made by the rolling ball.

In most applications, the camera is rotated around its own X and Y axis and the previous equation is sufficient. However, if one wants for instance to maintain the horizon horizontal in a walkthrough application, the camera will be rotated around a fixed up vector. We introduce the elevation angle α around the camera X axis and the rotation angle β around the world vertical axis (see Fig. 11a).

In this general case, Ω intersects the texture plane at a point C, which is the instantaneous rotation center of the texture coordinates (see Fig. 5). The screen center then describes a cone when the camera rotates around the vertical up vector.



(a) (b)

Figure 11: (a) Geometric configuration of the camera movements : cone described by the screen during a $\Delta\beta$ rotation. (b) Geometric configuration of the texture plane.

In the screen-texture plane, a world $\Delta\beta$ rotation will result in a rotation around C, with an angle $\Delta\gamma$:

$$\Delta\gamma = \sin(\alpha)\Delta\beta \qquad d = \frac{R}{\tan(\alpha)}$$

In our walkthrough application, we combine a camera X axis up-down rotation ($\Delta\alpha$), with a vertical axis panning rotation ($\Delta\beta$). The paper texture rectangular window is then defined by its center (X,Y) and its orientation γ (Fig. 11b), updated as follows:

$$\begin{pmatrix} X \\ Y \end{pmatrix} \mathrel{+}= \mathcal{R}_\gamma \begin{pmatrix} d\,\sin\Delta\gamma \\ d\,(1-\cos\Delta\gamma) \;+\; R\,\Delta\alpha \end{pmatrix} \qquad \gamma \mathrel{+}= \Delta\gamma$$

where $\mathcal{R}_\gamma$ is the rotation of angle γ in the texture plane.

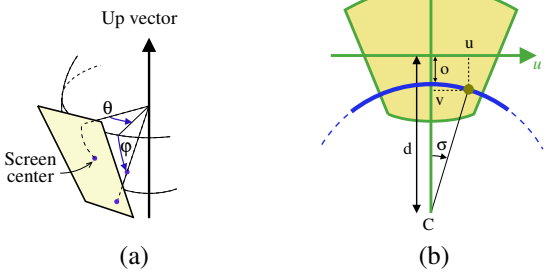
Note that this equation is still valid when $\alpha = 0$, where it reduces to $\begin{pmatrix} X \\ Y \end{pmatrix} \mathrel{+}= \mathcal{R}_\gamma \begin{pmatrix} R\,\Delta\beta \\ R\,\Delta\alpha \end{pmatrix}$, a simple rotation around the camera X and Y axis.

If a is the camera aspect ratio, the final rectangular texture coordinates are: $\begin{pmatrix} X \\ Y \end{pmatrix} + \mathcal{R}_\gamma \begin{pmatrix} \pm zoom\,a \\ \pm zoom \end{pmatrix}$.

B Spherical warp equations

The first step of the spherical deformation algorithm (see Sec. 4) projects a point of the screen onto the sphere. These spherical coordinates (θ, φ) are measured with respect to the up-vector, so that for the center of the screen $\theta = 0$ and $\varphi = \alpha$ (Fig. 12a).

The sphere parallel for φ is projected in the texture plane onto a circle of center C, and radius defined by the offset o relative to the projection of the center of the screen (see Fig. 12b). In order to ensure no sliding along the screen vertical center line for up-down rotation , o has to satisfy $o = (\varphi - \alpha)\,R$.



(a) (b)

Figure 12: Spherical and texture plane coordinates

Once the texture circle has been determined, the (u,v) texture coordinates of the screen point are determined by the angle σ (Fig. 12b). This angle is such that when the sphere rotates, the distance covered on the texture circle matches the arc length on the sphere $(d-o)\,\sigma = R\cos(\varphi)\,\theta$. Finally we get:

$$u = (d-o)\,\sin(\sigma) \qquad v = o \;+\; (d-o)(1-\cos(\sigma))$$

Eurographics Symposium on Rendering (2005)
Kavita Bala, Philip Dutré (Editors)

Geometric clustering for line drawing simplification

P. Barla, J.Thollot and F. X. Sillion

ARTIS GRAVIR/IMAG INRIA †

Figure 1: *The two stages of our method. Lines of the initial drawing (left) are first automatically clustered into groups that can be merged at a scale ε (each group is assigned a unique color). A new line is then generated for each group in an application-dependent style (at right, line thickness indicates the mean thickness of the underlying cluster).*

Abstract

We present a new approach to the simplification of line drawings, in which a smaller set of lines is created to represent the geometry of the original lines. An important feature of our method is that it maintains the morphological structure of the original drawing while allowing user-defined decisions about the appearance of lines. The technique works by analyzing the structure of the drawing at a certain scale and identifying clusters of lines that can be merged given a specific error threshold. These clusters are then processed to create new lines, in a separate stage where different behaviors can be favored based on the application. Successful results are presented for a variety of drawings including scanned and vectorized artwork, original vector drawings, drawings created from 3d models, and hatching marks. The clustering technique is shown to be effective in all these situations.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Line and Curve Generation Picture/Image Generation

1. Introduction

Line drawing is an important aspect of modern graphics; it allows an intuitive depiction of complex scenes with a remarkable economy of means. This is probably due to the ability of the human visual system to perceive shape from intensity discontinuities.

Artists have since long learned to use this perceptual property to provide stunningly expressive pictures, by cleverly tuning line density across their drawings. However, most of the time in computer graphics, lines do not come with an appropriate density. Simply scaling a drawing, for instance for displaying on low-resolution devices, creates a need for density adjustment; moreover, density reduction in 3d is not yet mature and most non-photorealistic rendering (NPR) systems extract far too many lines. Thus there is a need to adapt the number of lines in a drawing, otherwise the effectiveness of such a representation may be compromised.

There is not a single way to simplify a set of lines, depending on the envisioned application. In the context of density reduction, we may want to adjust the line density of a drawing where too many lines project in a given region of the image. This is needed when scaling a line drawing, as well as when rendering from a 3d scene. In this context only the most "significant" lines should be drawn. Level-of-detail (LOD) representations for line-based rendering (contours and hatching), where the number of lines must vary with scale, constitute another simplification approach. Finally, in the context of progressive editing (sometimes called over-sketching), the user refines a curve by successive sketches. This can be viewed as an iterative simplification of the set of line sketches provided by the user.

In this paper, we analyse the properties shared by these three applications and propose a common solution.

† ARTIS is a research team of the GRAVIR/IMAG laboratory,a joint unit of CNRS, INPG, INRIA and UJF.

© The Eurographics Association 2005.

1.1. Problem statement

We consider a drawing to be a digital image composed of a number of vectorized 2d lines. Such images can be obtained in various ways: by scanning and extracting lines from a hand-made drawing; by direct digital creation using appropriate input devices (mouse, tablet, etc); by detecting contours in an image [ZT98]; or by rendering a 3d scene in a line style [SS02, GG01, GTDS04]. We thus limit our approach to static 2d drawings.

We are focusing on simplification of such static 2d drawings, *i.e.*, the creation of another set of lines containing fewer lines than the original set. We propose a generic approach for this type of problem, where simplification is controlled by a single distance-based scale parameter ε.

Of course, this rather restricted view (in which spatial proximity is used as the main discriminating criterion) implicitly assumes that all lines belong to a coherent set, over which simplification can be carried out using a very low level semantic description. In particular this approach is not tailored to regular structures or other higher order arrangements. However, nothing prevents the user from preprocessing the data to organize lines in different categories and to apply our method to simplify independently each category.

1.2. Related work

We now review relevant work involving line drawing simplification. Two research fields deal with line drawing treatments but are beyond the scope of this paper: beautification of a drawing that essentially tries to satisfy some geometric constraints to correct technical diagrams; and simplification of a single curve that deals with maintaining the global shape of a curve while decreasing its resolution. We do not consider these two fields but rather concentrate on techniques that simplify a set of curves without imposing a predefined model.

Progressive drawing tools [IMKT97, Bau94] are useful in the context of sketch-based modeling, or within vector graphics packages such as Adobe Illustrator™. These dedicated tools assist the user in adjusting the shape of a line: they are essentially semi-automatic, work iteratively and are not designed to edit more than one line at a time. Thus they are not easily adapted to other, non-progressive applications.

Several algorithms have been proposed to control the density of lines in 3d renderings. Deussen et al. [DS00] present a simplification technique dedicated to trees and vegetation, which relies on their intrinsic hierarchy, and works in object space. Preim et al. [PS95] and Wilson et al. [WM04] measure density in image space in order to limit the number of lines drawn for complex objects. Similarly, Grabli et al. [GDS04] introduce density measures in image space, used to select the most significant lines. The use of information extracted from the 3d scene (silhouettes, creases, etc.) allows them to evaluate this "significance" and order the lines by decreasing priority. The simplification process is then carried out by deleting the least significant lines. Similar approaches have been proposed in the context of resolution-dependent display and printing, since they are related to halftoning [SALS96, ZISS04]. Here again the authors add a notion of priority to ensure that the most important lines are drawn first for any tone level and then offer a selection mechanism of some lines among the original ones.

In the field of illustration, Winkenbach et al. [WS94] introduced the notion of indication: complex textures are only fully rendered in certain places of the drawing at an appropriate density, to suggest the complexity of a pattern (such as a brick wall).

Several papers about non-photorealistic rendering deal with level-of-detail rendering for animated scenes. Praun et al. [PHWF01] present an image-based method to handle LOD in hatching. Their *tonal art maps* (TAMs) are mip-mapped textures allowing real-time display of hatching styles. Other LOD creation systems have been proposed, such as the "WYSIWYG NPR" system [KMM*02] that lets the user specify the appearance of the object for several view points. Relevant LODs are then blended for a given view.

The methods presented above **only delete** the less significant lines and do not consider any **perceptual** aspect of line simplification. On the other hand, perceptual grouping approaches provide effective ways of consistently grouping lines, even if they do not adress the problem of simplification directly. Most of the work in this area deals with the extraction of closed paths in drawings [Sau03, EZ96], focusing on grouping criteria such as good continuation and closure. However, other criteria are more relevant for simplification purposes, e.g. proximity and parallelism. Unfortunately, even if each criterion has been studied in isolation [Ros94], their relative influence is yet to be determined.

1.3. Contributions

Our two main contributions reside in an attempt to model the common properties of target applications of line drawing simplification while allowing various simplification behaviors: contrary to previous methods, we construct a partition of the original set into **consistent groups** that can be replaced by an **entirely new line**. To this end, we draw inspiration from perceptual grouping.

We decompose the process into two main stages (see Fig. 1): a clustering stage in which we group the original lines and a geometric stage where a new line is created for each group. While the former is entirely automatic and common to all applications, the latter is oriented toward the specific needs of each of the envisioned applications. Our approach is general in the sense that it considers a set of minimal and low level goals shared by those applications. Therefore, it is not able to deal with higher-level structures like Winkenbach and Salesin [WS94].

© The Eurographics Association 2005.

We begin by describing our methodology in Section 2. The common, automatic clustering stage is presented in depth in Sections 3 and 4. Our contribution here is the definition of a modular algorithm that clusters any kind of line.

Section 5 shows how to adapt our clustering algorithm to different needs, giving simplification results in the contexts of density reduction, LOD and progressive drawing; for lines coming from different sources: scanned drawings or non-photorealistic renderings. Finally we discuss limitations of our method in Section 6.

2. Methodology

We now present the principles of our simplification method, including formal definitions that will help to clarify our approach.

2.1. Input lines

We define a line-drawing as a set of 2d lines holding a set of attributes (color, thickness, style parameters, etc.), without any assumption on their nature. Thus a line l is only defined by two end points and a continuous path between them.

$$l : [0,1] \rightarrow \mathbb{R}^2 \times \mathbf{A}$$

where $\mathbf{A}$ is the space of attributes.

In the following, we will denote by $l|_{[a,b]}$ the part of l restricted to $[a,b]$ with $0 \leq a,b \leq 1$. We are not interested in an exact parameterization of lines, but only on their geometric properties. Therefore in the rest of the paper, we will refer to a line l indiscriminately to refer to the set of geometric points that constitutes it.

2.2. Objectives

As already stated, our main goal is to create a set of lines containing fewer lines than the original one. For that, we first need to control the amount of simplification accomplished by our method. Our target applications all have a single common parameter: the simplification scale. This scale, which we denote by ε, is thus the only parameter needed by our approach. Intuitively, we only simplify the existing information at a scale smaller than ε, keeping all the information present at a larger scale.

In the applications we envision, we first need to ensure that the overall configuration of the original drawing is respected in the simplified one. Regarding perceptual grouping, this means taking **proximity** and **continuation** effects into account. To this end, we impose a **coverage** property which consists of creating new lines only in regions where initial lines can be found.

However, we are not only focusing on the line positions, but also on their shape. We want the new lines to respect the way the initial lines have been created. For instance, in Fig. 2-(a), the new line folds onto itself to cover the original lines that form a fork (Y-shapes), although no such fold was initially present. We prefer a solution such as the one shown in Fig. 2-(b), using one more simplified line, but with more fidelity to the global shape of the original lines. The same problem can be found in other examples, such as hatching groups used to shade regions (see Fig. 2-(c),(d)). In order to preserve the shape of the original drawing, we thus need another perceptual property: we want the new line and the clustered lines to be **parallel** at the scale ε. To do that, we impose a **morphological** property on simplified lines that prevents them from folding onto themselves.

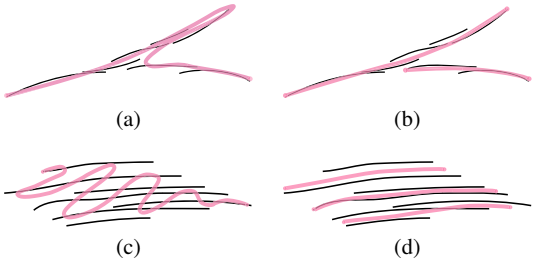


Figure 2: *(a) The simplified line (in pink) has a fold while the initial group (in black) does not - (b) Two simplified lines are preferred to represent this group - (c) The simplified line does not reflects the initial orientation and shape of the group - (d) Using three simplified lines better maintains the shape of the hatching group.*

Finally, still following perceptual grouping, we want to be able to reject the simplification of a pair of lines if their attributes (e.g. **colors**) are too different.

Following our objectives, we now give formal definitions related to our simplification approach.

2.3. Definitions

We begin by the definition of an ε-line and use it to define a group that can be simplified by a single line at the scale ε.

Following our **morphological** property, an ε-line is a line that does not fold onto itself at the scale ε. It corresponds to the fact that, for each point of l, there is no point along the normal at a distance less than ε that also belongs to l (see Fig. 3). We assume l to be G^1 in order to ensure that its normal is uniquely defined at each point:

Definition 1 Let l be a line. l is an ε-line if and only if l is G^1 and

$$\forall p \in l,\ \nexists q \in l,\ \begin{cases} q = p + \sigma \vec{n}_l(p) \\ \sigma = ||p-q|| \leq \varepsilon \end{cases}$$

where $\vec{n}_l(p)$ is the normal vector of l at point p.

This definition is equivalent to saying that l does not intersect either of its two offset curves $l^{+\varepsilon}$ and $l^{-\varepsilon}$ (see Fig. 3).

We now define an ε-group as a group of lines that can

© The Eurographics Association 2005.

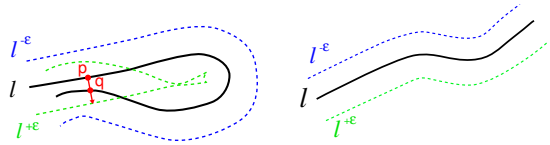


Figure 3: *(a) q is along the normal at p thus the line l has a fold and hence is not an ε-line - (b) l is an ε-line.*

be simplified by a single ε-line, as stated in our **coverage** property (see Fig. 4):

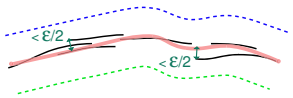


Figure 4: *An ε-group is a group (in black) that can be covered by an ε-line (in pink) at the scale ε.*

Definition 2 A group of lines G is an ε-group if and only if there exists an ε-line l such that[†]:

$$d_{SH}(l,G) < \frac{\varepsilon}{2}$$

where d_{SH} is the symmetric Hausdorff distance defined between two sets of points by:

$$\begin{aligned} d_{SH}(P,Q) &= \max(h(P,Q),h(Q,P)) \\ h(P,Q) &= \max_{p\in P}(\min_{q\in Q}||p-q||) \end{aligned}$$

An ε-group is thus a group that meets the proximity, continuation and parallelisms requirements at the scale ε.

2.4. Our approach

Following our definitions, our approach states that a simplified line-drawing is a set of ε-lines that covers the original drawing at a scale ε. We first need to cluster the original lines in a set of ε-groups before being able to create any new line. Therefore our simplification method is organized in two stages:

1. A clustering stage first groups the lines of the original drawing in ε-groups. No line is created at this stage and the process is entirely automatic using a greedy algorithm to iteratively group the original lines.
2. A geometric stage then creates a single line for each cluster of the original drawing. For each of the three target applications, we use the clusters differently and apply dedicated strategies.

[†] $\frac{\varepsilon}{2}$ ensures that two lines of the same ε-group are at a distance smaller than ε

The clustering stage is the main contribution of this paper, thus it is presented in detail in the next two sections. We then give in Section 5 various examples of the geometric stage and show that our approach can address specific applications without demanding too much effort on the user side.

3. Clustering

We use a greedy algorithm to partition the set of input lines. It is based on the iterative clustering of pairs of ε-lines and maintains the ε-group property during the entire process: clustering a pair of ε-lines that each represent an ε-group results in a new ε-line that represents the merged group. The first step consists of converting the original lines into ε-lines by splitting them at their points of intersection with their offset curves (see Section 4.1). Our clustering algorithm then iteratively clusters the pairs of ε-lines (Section 3.1) that have the minimum error (Section 3.3) until no more clusters can be created. At each step we store the hull of the clustered pair in order to take into account the result of previous clusterings in the next steps (Section 3.2).

3.1. Clustering pairs of ε-lines

Following our definitions, a pair of ε-lines (l_1,l_2) can be clustered if and only if (l_1,l_2) is an ε-group. This definition is not constructive: it only says that there must exist an ε-line that covers (l_1,l_2). We now describe a way of building this new ε-line from l_1 and l_2.

3.1.1. Possible configurations for (l_1,l_2) to be an ε-group

Observation 1 The coverage property (Def. 2) implies that if (l_1,l_2) is an ε-group then there exists a point p_1 (resp. p_2) of l_1 (resp. l_2) such that $||p_1-p_2|| < \varepsilon$. If not, it would not be possible to find a line l such that $d_{SH}(l,(l_1,l_2)) < \varepsilon/2$.

We call *overlapping zones* the portions where l_1 and l_2 are at a distance less than ε, formally defined as:

Definition 3 An overlapping zone, Z, is a pair of line portions $(l_1|_{[a_1,b_1]}, l_2|_{[a_2,b_2]})$ such that:

$$d_{SH}(l_1|_{[a_1,b_1]}, l_2|_{[a_2,b_2]}) < \varepsilon$$

where $\{a_1,a_2\}$ and $\{b_1,b_2\}$ are the extremities of Z.

Observation 2 The morphological property (Def. 1) implies that (l_1,l_2) is an ε-group if it is not a fork. Indeed, if it were the case, then any line l representing (l_1,l_2) would have to fold onto itself.

This implies that the overlapping zones must not fork, thus there must be at least one of the two lines that ends at each extremity of the zone. A simple forking configuration is shown in Fig. 5-(a). Other forking configurations exist, but are not represented because we mainly direct our attention to valid zones (Fig. 5(b) and (c)).

We call such an overlapping zone a *path* and define it by:

© The Eurographics Association 2005.

(a) Fork (b) Four possible configurations with one path (c) Two paths: closed curve

Figure 5: *Possible configurations of a pair of ε-lines. Only (b) and (c) can form an ε-group.*

Definition 4 A path on a pair of lines (l_1, l_2) is a maximal overlapping zone, Z, such that there is at least one extremity of l_1 or l_2 at each extremity of Z.

Knowing that each line has 2 extremities, there are five combinations for the paths between two lines, illustrated in Fig 5-(b), (c). Thus, if a pair of lines does not correspond to one of these five combinations, it is not an ε-group.

The only configuration that corresponds to a closed curve is when a pair of lines (l_1, l_2) has two paths (see Fig 5-(c)). For the sake of brevity, we will not detail this case in the following, as it is essentially equivalent to the others. However Section 3.4 shows that our algorithm correctly handles it.

3.1.2. Building the new ε-line

Now that we have identified valid configurations (paths), we build the new line l, and make sure that it is an ε-line, i.e. that it respects Definition 1. We create an ε-line l that passes from l_1 to l_2 and lies in the middle of the path.

l is obtained by concatenating the portions of lines outside the path (in purple) with a line created inside the path by interpolating between l_1 and l_2 from one extremity to the other (in pink).

Such a construction insures that l is an ε-line outside the path since l_1 an l_2 are themselves ε-lines. However for zones inside the path, some particular cases when l is not an ε-line exist. Indeed, l may fold onto itself if the curvature of l_1 or l_2 is too close to $1/\varepsilon$. In these cases, (l_1, l_2) is simply not considered as an ε-group.

3.2. Building the hull of an ε-group

The new line l we created only ensures that (l_1, l_2) is an ε-group; we cannot use it iteratively since it would not take into account the error made by the clustering of l_1 and l_2. Indeed, consider an ε-line l_3; determining if (l_1, l_2, l_3) is an ε-group is not directly equivalent to determining if (l, l_3) is an ε-group.

In order to propagate the clustering result of (l_1, l_2) to l, we define the hull of an ε-group by assigning a varying thickness to the ε-line that represents it. This thickness describes the result of the clustering of two or more lines and is used in subsequent clusterings (see Fig. 6).

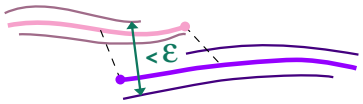


Figure 6: *To decide if the four thin lines are an ε-group we use their two representatives (in pink and purple) and compute the error measure (see Section 3.3) between the farthest lines, which will be represented by the hull.*

For each point $l(x)$, the points of the hull $l^+(x)$ and $l^-(x)$ are obtained by taking the extremal intersections along the normal with (l_1, l_2) (see Fig. 7).

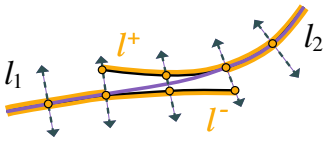


Figure 7: *The hull (in orange) of a line l (in purple) representing a pair of lines (l_1, l_2) (in black) is defined by the farthest points of (l_1, l_2) along each normal of l (dashed line).*

All the definitions given in the previous section are easily extended by considering the two hulls instead of the two ε-lines. Indeed, to decide if a pair of ε-lines (l_1, l_2) is an ε-group we only need to compute distances between pairs of points. By considering $l_1^+, l_1^-, l_2^+, l_2^-$ for the distance computation, we can determine the overlapping zones and then the paths between l_1 and l_2 while taking into account the two ε-groups they already represent. Therefore, while computing overlapping zones between two ε-lines, the distance between $l_1(x_1)$ and $l_2(x_2)$ will be taken as:

$$\max \{ ||l_1^+(x_1), l_2^+(x_2)||, ||l_1^+(x_1), l_2^-(x_2)||, ||l_1^-(x_1), l_2^+(x_2)||, ||l_1^-(x_1), l_2^-(x_2)|| \}$$

Note that the hull of an ε-line l is not defined on points p where there is no intersection with l_1 and l_2 and the normal at p. For those points, we project the closest points of the two hulls as shown in Fig. 8. This will only have an impact on the error computed on the hull as explained in the next section and our choice favors pairs of aligned lines (i.e., those with **good continuation**).

© The Eurographics Association 2005.

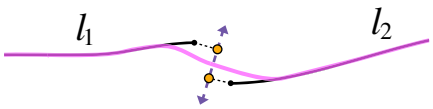


Figure 8: *In places where there is no intersection with the normal, we project the closest points on the pair of lines.*

3.3. Error measure of an ε-group

In order to use a greedy algorithm we now need to choose which pairs of ε-lines we want to cluster at each step. To this end we define an error measure.

Intuitively we want to cluster the closest lines first. By closest we mean not only spatially close but also with similar attributes. We first show how to compute the spatial error, then we explain how to incorporate an attribute error in order to orient the simplification toward a given application.

When computing the spatial error of a pair (l_1, l_2) of lines, we want to favor pairs of lines that could be clustered with the smallest possible ε. We thus define the spatial error $E_s(l_1, l_2)$ of an ε-group relative to the ε-line l chosen to represent it by the maximum thickness of the hull associated with l. This heuristic favors the clustering of the thinest groups first. This error is normalized between 0 and 1 using a division by ε:

$$E_s(l_1, l_2) = \max_{x \in [0,1]} ||l^+(x) - l^-(x)||/\varepsilon$$

The user can also define an attribute error measure $e_a(p_1, p_2)$ (normalized between 0 and 1) for a particular attribute space if he or she wants to take it into account in the clustering process. For the single attribute we used in our implementation (i.e., color), we found that a mean was better than a max to give a good estimation of the total error between two groups. This gives the following attribute error:

$$E_a(l_1, l_2) = \int_0^1 e_a(l^+(x), l^-(x))dx$$

The spatial and attribute error measures are then classically combined in a multiplicative way to give the error measure $E(l_1, l_2)$:

$$E(l_1, l_2) = 1 - (1 - E_s(l_1, l_2)) * (1 - E_a(l_1, l_2))$$

The attribute error is only computed for ε-groups, that is groups of lines that can be spatially clustered. In order to forbid clustering if the attributes of the lines of the group are too different, we add the constraint that for an ε-group (l_1, l_2) to be clustered, it must satisfy $E(l_1, l_2) < 1$.

3.4. Closed curves

Most of this method holds for closed curves and the algorithm is very similar. However, we need to implement some additional processes. First, the lines closed at the scale ε are detected. Those are the lines whose endpoints are at a distance less than ε. Moreover, when identifying paths, if the path configuration found in Fig. 5-(c) arises, the resulting ε-line becomes closed.

4. Implementation details

We have implemented the greedy iterative clustering by an edge collapse algorithm applied on a graph whose edges represent pairs of ε-lines which are ε-groups.

4.1. Preprocessing input lines

The lines we take as input can be of any kind. The only constraints are that we need to sample them. In our implementation, we use regularly-sampled Catmull-Rom splines. For all distance computations involving such samples, we use an acceleration grid of cell size ε, allowing us to quickly find candidate samples.

In order to initialize the algorithm, we need to convert an initial line l into an ε-line. To do that, we follow l, progressively creating its two offset curves, and we split l as soon as it crosses one of the already created offset curves. Note that this splitting process gives different results depending on the extremity at which one starts. We have not found any remarkable difference in the results; however, one may want to choose a more symmetric way of splitting. After this initialization step, each line is its own hull.

4.2. Building the graph

Once the input lines have been converted into ε-lines, we build a graph with a node for each input ε-line, and whose edges represents pairs of lines that can be clustered, i.e. that are ε-group.

A pair of ε-lines can only be clustered if it corresponds to one of the configurations shown in Fig 5-(b),(c). Thus, for an ε-line pair, if there are more than two extremities at a distance greater than ε from the other ε-line, we can reject it directly, saving a lot of computation time.

In practice, we compute a hull for each potential cluster and store it on the corresponding edge along with its error.

4.3. Updating the graph

Then, at each step of the algorithm, we collapse the edge with minimum error and update the graph edges locally. Collapsing an edge is done by creating a new node that stores the edge's ε-line and hull. The collapsed edge is deleted and by definition of a hull, we only have to inspect the edges incident to the collapsed nodes. Those edges are removed from the graph and new edges are created between the new node and its neighbors. We also compute the attributes of the new ε-line by linearly interpolating the attributes of the two original ε-lines.

© The Eurographics Association 2005.

Finally, the two collapsed nodes are removed from the graph. But instead of deleting these nodes, we keep them in a history of collapse sequences which is stored as a tree under the newly created node. This gives us access to the underlying input lines and the collapsing scheme of each cluster.

The algorithm stops when no more clusters can be created.

5. Results

In this section, we give some results to illustrate the overall simplification process, i.e. both clustering and geometric stages for each of the target applications: density reduction, level-of-detail and progressive drawing[‡]. The geometric stage is clearly a more specialized operation since the choice of the new line to be drawn is left to the chosen strategy. We implemented two "standard", pre-defined strategies:

- **Average line:** the new line interpolates all the original lines in the cluster (with application-defined weights);
- **Most significant line:** the new line is one of the original lines, chosen according to an application-defined priority measure (base on length, nature...).

The former supports a broad range of simplification behaviors, while the latter gives a simple selection/deletion scheme.

In the worst case, the total process increases quadratically with the number of input lines at a fixed scale parameter ε. In practice, for our examples, it ranges from several seconds to a minute. For each example we give the number of input lines and resulting clusters. The simplification scale is shown by a circle of diameter ε.

Density reduction Fig. 9 shows a straightforward illustration of our approach. Lines have been extracted from a scanned line drawing. The user chooses a simplification scale ε and the lines are simplified. We applied an average line strategy without smoothing the results, so that simplified lines exhibit the ε-line of each group.

Fig. 11 shows a similar scenario that takes the color attribute into account. The attribute error is a $L^*a^*b^*$ color distance.

Fig. 12 shows the use of categories to separate lines of different nature: external contour on one side and internal and suggestive contours [DFRS03] on the other side. In examples coming from 3d renderings like this one, we make use of object IDs and line nature to detect categories automatically. This allows for the use of two different geometric strategies: for the external contour an average line is drawn, whereas the longest line of each cluster is drawn for the internal and suggestive contours. Moreover, the external contour is simplified at a larger scale than the other lines. Resulting lines are better organized and keep the most salient features of the model.

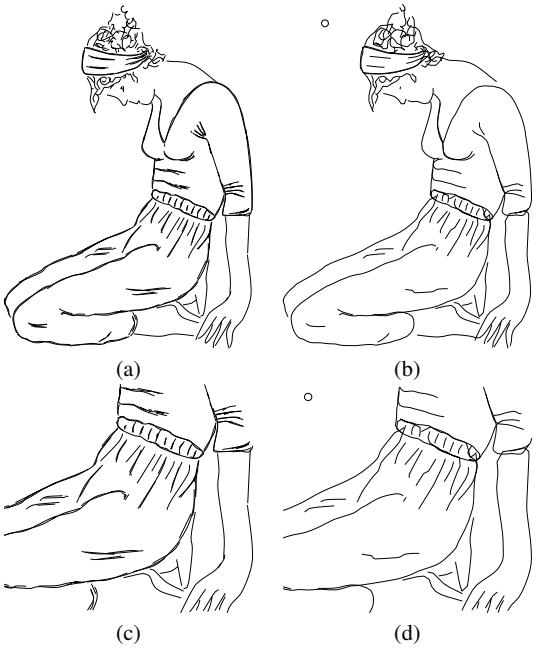


Figure 9: ***Density reduction:*** *(a) The original scanned and vectorized drawing: 357 input lines - (b) The resulting simplification: 87 clusters - (c,d) Zoom on the above images. The scale* ε *is indicated by the circle in the upper left corner.*

Level-of-detail Fig. 10 shows an example of a LOD sequence produced with our approach. Progressively scaling down a drawing is equivalent to choosing an increasing ε. Thus we apply a series of simplifications with an ε step, each time starting from the previous, finer level. Here again, two different geometric strategies are used: the average line for the contour and the longest line for the hatchings. To do that, we created five categories by hand: one for the contours, and one for each of the four orientations for hatchings. Note that although no particular treatment was applied to preserve tone across simplifications, this result is quite convincing. Tone preservation could be explicitly included in the method at the geometric stage, by choosing appropriate line attributes such as width and/or color.

Progressive drawing Fig. 13 shows a drawing sequence using our progressive drawing tool. Here, the clustering algorithm is applied iteratively: The user chooses a sensitivity ε and draws a sketched line over an initial drawing; the lines are then simplified; and finally, the resulting lines constitute the initial drawing for the next step. This tool requires

[‡] A video showing an oversketching session and two example LODs (including the tree of Fig. 10) is available at http://artis.imag.fr/Publications/2005/BTS05a/

© The Eurographics Association 2005.

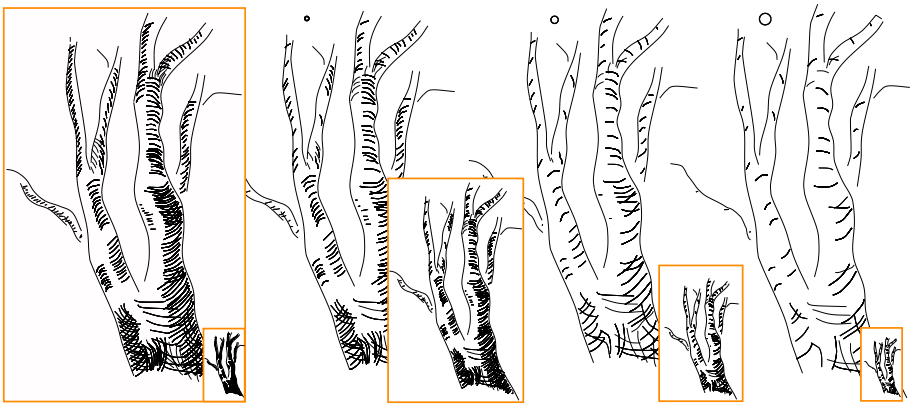


Figure 10: ***LOD:*** *A series of LODs made by progressively increasing* ε*. Using different categories and geometric strategies prevents undesired hatching lines from merging. Compare the small resized images with (right) and without (left) simplification.*

an additional feature: we only want the simplification to be done between initial lines and the new sketch. Thus the input lines are organized in two sets: the initial lines and the new sketched line. During the clustering, only the edges between pairs of nodes that lie in different sets are built.

Finally we choose a priority-based strategy because we want the last drawn line to have a greater priority than initial lines. In practice, that consists of using an average line strategy, giving greater weights to samples belonging to the last drawn line. This is made possible by the history tree stored at each cluster. We found this tool to be very intuitive, particularly for modifying lines coming from 3d renderings or extracted from images.

6. Discussion

In this paper we opted to remain very general, trying to find the common properties of some target simplification methods. However, it is clear that such a low-level method can still be specialized to adapt to other specific applications. In particular, we believe that the separation of the clustering and geometric stages is crucial for all simplification methods.

Other attributes than color could be used in the attribute error definition. However, we did not consider input lines exhibiting wiggling patterns and implicitly assumed that they come at an appropriate scale. The problem of extracting the so-called natural scale of a line has been previously adressed (e.g. [Ros98]).

Our method is invariant under rotation, scale, and translation, since it operates only on euclidean distances between pairs of points. However it has two limitations: it is not transitive and prevents simplifying forks. The former means that simplifying a drawing at scale ε_1, then simplifying the result at scale $\varepsilon_2 > \varepsilon_1$ is not guaranteed to provide the same result as a direct simplification at scale ε_2. However, this is not a problem in the applications we envision, for instance generating a discrete set of LOD representations. The latter assumes that the problem of forks is rather separate from geometric clustering (appearing at a higher level of processing and depending on the application) and thus is left as a post process.

The choice of a greedy algorithm for clustering implies that we only reach a local optimum in general. This turns out to be sufficient in practice for the applications we have tested. Other optimization techniques could be used if reaching a global optimum is important.

The evaluation of a simplification method for line drawings is not an easy task. Indeed, there is no simple and obvious quality measure for a simplified drawing. Visual evaluation involves a number of high-level interpretation processes, which are difficult to model and quantify. Our approach offers the convenience of a guaranteed geometric criterion: the resulting drawing is "within a distance ε" from the original drawing. The direct evaluation of the result is the number of clusters. However, in an attempt to provide finer evaluation tools we identified two other criteria. First, the reduction in the number of lines composing the drawing; Second, the variation of the total arc-length in the drawing. Both are strongly related to the geometric strategy chosen for an application: keeping a line per cluster clearly decreases the total number of lines, and the arc-length may strongly vary depending on the new lines created. For instance, in Fig. 9 the number of lines was divided by 4 and the arc-length reduced by 30%.

In the examples shown, we observe that a purely distance-based simplification should generally not be applied to all lines of the drawing at once, because there are categories of lines that should not be clustered. one example is lines

© The Eurographics Association 2005.

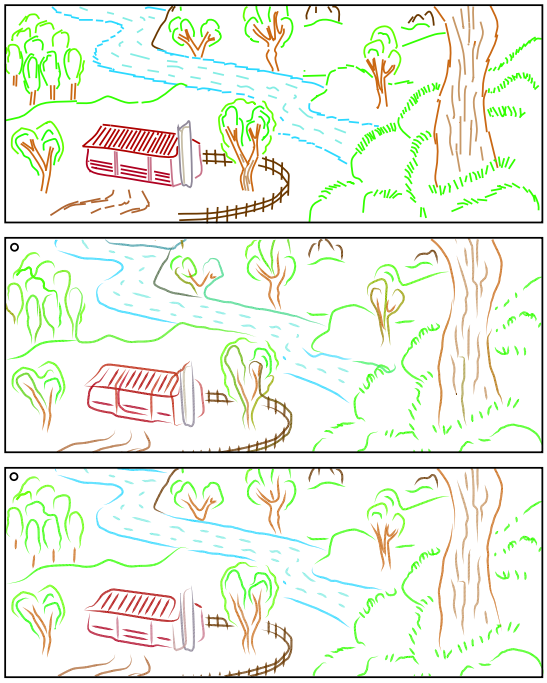


Figure 11: *Top: input drawing. Middle: simplified drawing without taking color error into account during the clustering stage. Bottom: taking color error into account better preserves the original drawing (see the fence and the tree, the trunks and the leaves...).*

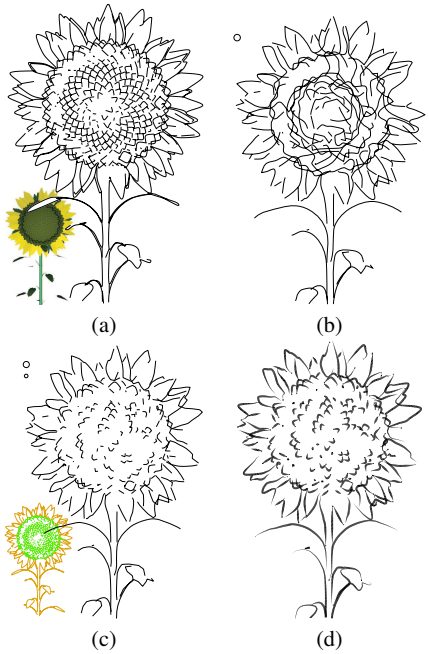


Figure 12: ***Simplification of a 3d rendering:*** *(a) 3d model and its line rendering using silhouettes and suggestive contours (531 input lines) - (b) Simplification without any category (256 clusters) - (c) Using two categories (external and internal contours) each with a different scale and geometric strategy (294 clusters) - (d) Same result in a calligraphic style.*

depicting different objects placed near each other. Segmenting the drawing and applying the simplification algorithm to each category is better for the scope of an automatic process. Naturally this raises the question of how to segment or define the categories automatically, which is beyond the scope of this paper.

Finally, we think that our approach could be extended to animation. The idea would be to guide the clustering stage temporally, not only to ensure temporal coherence, but also to cluster lines that go together across time. This could be accomplished by incorporating another perceptual grouping criterion: common fate, which states that the visual system tends to group elements with similar velocity.

7. Future work

Our approach can be extended in many ways. Forks, where a line separates into two distinct lines, are not handled explicitly in our technique. We plan to take them into account in the density reduction application, along with new geometric strategies. We plan to extend our LOD system to more elaborate transitions between levels: since we keep all the history of agglomerations, we have all the information needed to realize geometric morphs instead of blends. Moreover, we will add new features to our oversketching tool (such as alignment, anchoring, etc.).

Another interesting issue lies in the coupling of our method with contour detection in images, with applications in medical imagery and image-based rendering. Our approach is well adapted to these applications since it is able to incorporate any kind of data associated with the extracted lines (gradient, color, etc.) and to take them into account in the simplification process.

Finally, a long-term goal is to adapt our approach to the simplification of animated line drawings, taking into account the observations made at the end of Section 6.

8. Conclusions

We have presented a new, generic approach to the simplification of line drawings, that supports a large variety of applications. The clustering stage identifies groups of lines that capture the morphological structure of the original drawing; the geometric stage builds the actual lines that will represent this structure in the final drawing.

© The Eurographics Association 2005.

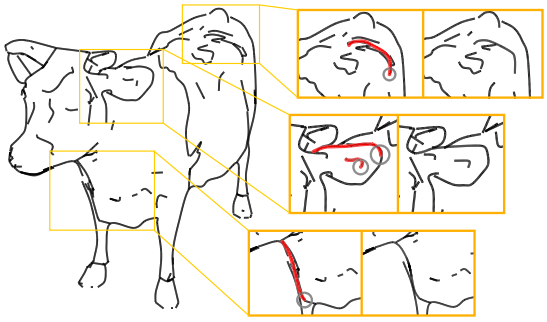


Figure 13: ***Oversketching:*** *A 3d model has been rendered in a line drawing style. The user adds new lines (in red) which are clustered with the old ones; the scale ε (gray circle) can be changed at each step.*

The low level and generic nature of this process makes it a solid foundation for many applications. We have demonstrated three possible scenarios: Density reduction of a set of lines where input lines, possibly classified in categories, are replaced by a smaller set of lines; an automatic level-of-detail system for line drawings with specific behaviors for silhouette lines and hatching groups; and a progressive drawing tool, which provides an intuitive and flexible interaction environment where the user creates or modifies existing lines with drawing gestures on the canvas.

Acknowledgement

We would like to thank Gilles Debunne for his help on the making of the video. This paper benefited greatly from suggestions given by the members of the ARTIS team and Lee Markosian. This work was supported in part by Region Rhone-Alpes (DEREVE project and EURODOC grant).

References

[Bau94] BAUDEL T.: A mark-based interaction paradigm for free-hand drawing. In *UIST '94: Proceedings of the 7th annual ACM symposium on User interface software and technology* (New York, NY, USA, 1994), ACM Press, pp. 185–192. 2

[DFRS03] DECARLO D., FINKELSTEIN A., RUSINKIEWICZ S., SANTELLA A.: Suggestive contours for conveying shape. *ACM Transactions on Graphics 22*, 3 (July 2003), 848–855. 7

[DS00] DEUSSEN O., STROTHOTTE T.: Pen-and-ink illustration of trees. *Proceedings of SIGGRAPH* (2000). 2

[EZ96] ELDER J. H., ZUCKER S. W.: Computing contour closure. In *ECCV (1)* (1996), pp. 399–412. 2

[GDS04] GRABLI S., DURAND F., SILLION F.: Density measure for line-drawing simplification. In *Proc. of Pacific Graphics* (2004). 2

[GG01] GOOCH, GOOCH: *Non-Photorealistic Rendering*. AK-Peters, 2001. 2

[GTDS04] GRABLI S., TURQUIN E., DURAND F., SILLION F.: Programmable style for npr line drawing. In *Rendering Techniques 2004 (Eurographics Symposium on Rendering)* (june 2004). 2

[IMKT97] IGARASHI T., MATSUOKA S., KAWACHIYA S., TANAKA H.: Interactive beautification: A technique for rapid geometric design. In *UIST (ACM Annual Symposium on User Interface Software and Technology)* (1997), pp. 105–114. 2

[KMM*02] KALNINS R. D., MARKOSIAN L., MEIER B. J., KOWALSKI M. A., LEE J. C., DAVIDSON P. L., WEBB M., HUGHES J. F., FINKELSTEIN A.: WYSIWYG NPR: drawing strokes directly on 3d models. In *SIGGRAPH 2002* (2002). 2

[PHWF01] PRAUN E., HOPPE H., WEBB M., FINKELSTEIN A.: Real-time hatching. In *SIGGRAPH 2001, Computer Graphics Proceedings* (2001), Fiume E., (Ed.), pp. 579–584. 2

[PS95] PREIM B., STROTHOTTE T.: Tuning rendered line-drawings. In *WSCG'95* (February 1995), pp. 228–238. 2

[Ros94] ROSIN P.: Grouping curved lines. In *5th British Machine Vision Conf* (York, 1994), pp. pp. 265–274. 2

[Ros98] ROSIN P. L.: Determining local natural scales of curves. *Pattern Recognition Letters 19*, 1 (1998), 63–75. 8

[SALS96] SALISBURY M., ANDERSON C., LISCHINSKI D., SALESIN D. H.: Scale-dependent reproduction of pen-and-ink illustrations. *Computer Graphics 30* (1996), 461–468. 2

[Sau03] SAUND E.: Finding perceptually closed paths in sketches and drawings. *IEEE Trans. Pattern Anal. Mach. Intell. 25*, 4 (2003), 475–491. 2

[SS02] STROTHOTTE T., SCHLECHTWEG S.: *Non-photorealistic computer graphics: modeling, rendering, and animation*. Morgan Kaufmann, San Francisco, CA, USA, 2002. 2

[WM04] WILSON B., MA K.-L.: Representing complexity in computer-generated pen-and-ink illustrations. In *NPAR* (2004). 2

[WS94] WINKENBACH G., SALESIN D.: Computer-generated pen-and-ink illustration. *Proc. SIGGRAPH* (1994). 2

[ZISS04] ZANDER J., ISENBERG T., SCHLECHTWEG S., STROTHOTTE T.: High quality hatching. *Computer Graphics Forum 23*, 3 (2004), 421–421. 2

[ZT98] ZIOU D., TABBONE S.: Edge detection techniques - an overview. *International Journal of Pattern Recognition and Image Analysis 8* (1998), 537–559. 2

© The Eurographics Association 2005.

Interactive watercolor rendering with temporal coherence and abstraction

Adrien Bousseau, Matt Kaplan, Joëlle Thollot and François X. Sillion
ARTIS GRAVIR/IMAG INRIA* Grenoble France

(a) (b) (c)

Figure 1: Various watercolor-like images obtained either from a 3d model (a,b) or from a photograph (c) in the same pipeline.

Abstract

This paper presents an interactive watercolor rendering technique that recreates the specific visual effects of lavis watercolor. Our method allows the user to easily process images and 3d models and is organized in two steps: an abstraction step that recreates the uniform color regions of watercolor and an effect step that filters the resulting abstracted image to obtain watercolor-like images. In the case of 3d environments we also propose two methods to produce temporally coherent animations that keep a uniform pigment repartition while avoiding the shower door effect.

Keywords: Non-photorealistic rendering, watercolor, temporal coherence, abstraction.

1 Introduction

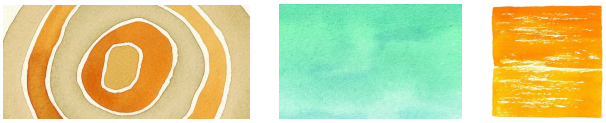
Watercolor offers a very rich medium for graphical expression. As such, it is used in a variety of applications including illustration, image processing and animation. The salient features of watercolor images, such as the brilliant colors, the subtle variation of color saturation and the visibility and texture of the underlying paper, are the result of a complex interaction between water, pigments and the support medium.

In this paper, we present a set of tools for allowing the creation of watercolor-like pictures and animations. Our emphasis is on the development of intuitive controls, placed in the hands of the artists, rather than on a physically-based simulation of the underlying processes. To this end, we focus on what we believe to be the most significant watercolor effects, and describe a pipeline where each of these effects can be controlled independently, intuitively and interactively.

Our goal is the production of watercolor renderings either from images or 3d models, static or animated. In the case of animation rendering, temporal coherence of the rendered effects must be ensured to avoid unwanted flickering and other annoyances. We describe two methods to address this well-known problem that differ in the compromises they make between 2d and 3d.

In the following, we will first review the visual effects that occur with traditional watercolor and present related work. Our pipeline is then described for images and fixed viewpoint 3d models, followed by our methods that address the temporal coherence of animated 3d models. Finally, we will show some results before concluding.

*ARTIS is a team of the GRAVIR/IMAG research lab (UMR C5527 between CNRS, INPG, INRIA and UJF), and a project of INRIA.



(a) (b) (c)

Figure 2: Real watercolor (©Pepin van Roojen). (a) Edge darkening and wobbling effect, (b) pigments density variation, (c) dry brush.

2 Watercolor effects

Curtis et al. [1997] listed the specific behaviors which make watercolor so distinct among painted media, that probably account for its popularity. We describe here the most significant and specific effects in watercolor, as illustrated in Figure 2 by pictures of real watercolor found in [van Roojen 2005].

First, the pigments are mixed with water but do not dissolve

totally. The result is that, after drying, even on a totally smooth paper, the pigment density is not constant (see Fig. 2-(b)). This manifests itself at two different scales: on the one hand there are low-frequency density variations due to a non homogeneous repartition of water on the canvas: the turbulence flow; on the other hand there are high-frequency variations due to non-homogeneous repartition of pigments in water: the pigment dispersion. In addition, the grain of the canvas can also introduce density variations since the pigments are deposited in priority in cavities of the paper.

Second, the fluid nature of watercolor creates two effects along the borders of colored regions: edge darkening due to pigment migration and a wobbling effect due to paper grain (see Fig. 2-(a)). In the case of a "dry brush" the paper grain also creates some white holes in the brush strokes when a nearly-dry brush only touches raised areas of the paper (see Fig. 2-(c)).

Finally, watercolor is usually created by a human, using a brush, often of a large size, and therefore represents a scene in a more abstracted way than a photograph. We believe that abstracting the shape, colors or illumination of the original scene is crucial to keeping the aesthetics of watercolor. Although this is true of most painting and drawing styles, the technical constraints of watercolor typically result in less precise details than other techniques.

We do not target the diffusion effect, also described by Curtis, for two main reasons. First this effect is often avoided by the artist when trying to depict "realistic" scenes. In such a case the lavis technique (wet on dry) is preferred. Second, this effect typically requires a fluid simulation, and would certainly not be temporally coherent. We thus leave this type of visual effect for future work and instead concentrate on uniform-color regions.

3 Related Work

Previous research in watercolor rendering can be grouped into two broad categories: Physical simulation of the media and image filters.

Physical simulations attempt to simulate all fluid, pigment and paper interactions. Curtis et al. [1997] described the critical effects created by watercolor media and presented a system that created convincing watercolor renderings. Their work was an extension of Small's [1991] connection machines. Curtis' method, while convincing, is extremely computationally expensive. Van Laerhoven et al. [2004] demonstrated an interactive method for rendering brush strokes painted by the user. Rendering eastern style inks is a related area that may be considered in this category [Chu and Tai 2005]. In this case the focus is more on the dispersion effects (wet on wet technique) than on lavis technique. Moreover, watercolor is a media consisting of a complex series of interactions between multiple layers of paint that diffuse pigment and water in ways that relate precisely to the method and area of their application. It is not clear that such methods are suitable for animation since the positions of painted regions in relation to one another necessarily change in animations. The interdependence of the qualities that make an image look like a watercolor painting may be too high to allow true temporal coherence.

Image processing filters consist of using image, and other G-buffers (like depth or normal buffer) as input to create a watercolor-like image that is based on an empirical recreation of the relevant effects rather than a physical simulation. These methods are fast and can typically be performed at interactive speeds but lack the richness of physical simulations. Lum and Ma [2001] present a multi-layer approach inspired by watercolor painting. Their work is performed in 3d object space to ensure frame-to-frame coherence. Line integral convolution is applied along the principal curvature directions to render brush-like textures. Burgess et al [2005] created a similar system, using Wyvill noise to create stroke textures. Luft and Deussen [2005] abstract ID images at their boundaries to create flow patterns and edge darkening. These effects are typically isolated in real paintings and do not occur simultaneously as in their method. Furthermore, abstracting at the object level removes all detail about surface shape. Lei and Chang [2004] demonstrate a rendering pipeline using a GPU to create watercolor effects at interactive speeds. They require the user to "pre-paint" each object using a lit sphere interface. All flow and granulation effects are then encoded in a 1d texture and rendered in a manner similar to a toon rendering [Lake et al. 2000]. Encoding flow effects in a 1d texture is too abstract for adequate detail on a smoothly changing 3d surface. Furthermore, lit sphere lighting specification may be prohibitive for a large number of objects or changing light sources.

Johan et al. [2005] used watercolor principles to create brush strokes in image space but they do not present a method for watercolor rendering.

Much of this previous work makes use of the Kubelka-Munk model to simulate the composition of pigments. This model is physically based but limits the choices of colors to a predefined pigment library. In the case of an image input (like in [Curtis et al. 1997]) a color reconstruction has to be done to choose the right pigments for a given color. We propose in this work a more intuitive color treatment that stays close to the original color of the input.

On the other hand, temporal coherence of brush strokes has been an active area of research since there is an inherent conflict between a 3d motion field and marks created in a 2d plane. Various methods has been presented either in object space like [Meier 1996] or in image space like [Wang et al. 2004]. The shower door problem between the canvas and objects in the scene has been addressed in [Cunzi et al. 2003; Kaplan and Cohen 2005].

Watercolor is, in our opinion, not exactly a mark-based rendering technique because the pigments can not be considered as single brush strokes; on the other hand the pigments have to follow the objects in the scene so we cannot directly apply the dynamic canvas technique as in [Cunzi et al. 2003]. The temporal coherence methods we present are extensions of these previous techniques.

4 Single-image pipeline

Our work is directly inspired from previous techniques, however we do not restrict ourselves to physically realizable images. Instead, we purposely lift the constraints inherited from the physics of real watercolor since they can be avoided by the use of the computer. We prefer to offer a system that reproduces watercolors visual effects, and let the user control each of these effects independently, even if the particular combination that results would not make sense in the physical world. Thus, we do not need to use the Kubelka-Munk pigment model or a physical fluid simulation as in previous approaches.

In order to obtain the effects mentioned in Section 2 we propose a unified framework using the pipeline shown in Figure 3 and the accompanying videos, which takes as input either a 3d scene or a single image.

Our contributions are (a) to provide the user with various abstraction steps allowing the creation of the uniform color regions needed in watercolor, and (b) to provide a simple color combination model that ensures a plausible result while avoiding physical constraints.

We now describe each step of the pipeline. For clarity of exposition, we begin with the second stage (watercolor effects) before considering the various abstraction possibilities.

4.1 Watercolor effects

The input to this part of the pipeline is an abstracted image that can come either from a 3d rendering or from a processed photograph. Figure 4 illustrates the successive steps in the case of a 3d model.

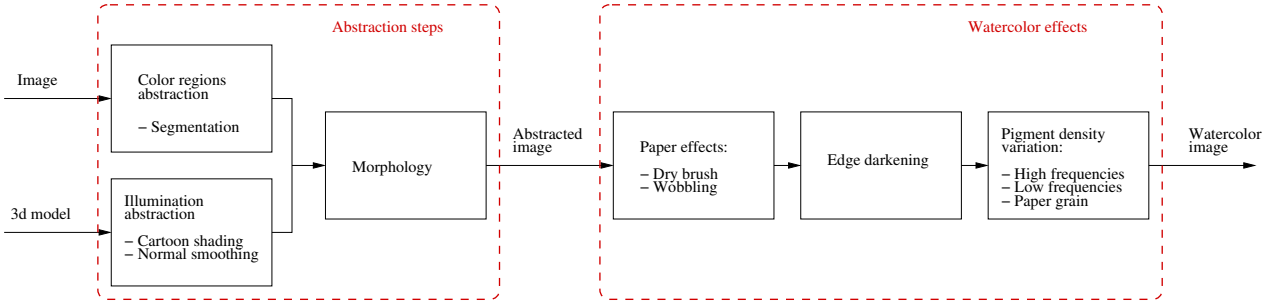


Figure 3: Static pipeline: Our pipeline takes as input either a 3d model or a photograph, the input is then processed in order to obtain an abstracted image and finally watercolor effects are applied to produce a watercolor-like image.

The main effect in watercolor is the color variation that occurs in uniformly painted regions. We first present the technique we use to reproduce these variations by modifying the color of the abstracted image and then we describe all the effects specific to watercolor.

4.1.1 Color modification

Figure 5: Darkening and lighting of a base color C (here $(0.90, 0.35, 0.12)$) using a density parameter d.

As mentioned earlier we chose not to use a physical model to describe color combinations. The essential positive consequence of this choice is that the user can freely choose a base color (think of a virtual pigment), whose variation in the painted layer will consist of darkening and lightening, as shown in Figure 5.

We build an empirical model based on the intuitive notion that the effects due to pigment density can be thought of as resulting from light interaction with a varying number of semi-transparent layers. Consider first a color C ($0 < C < 1$) obtained by a uniform layer of pigment over a white paper, we can interpret this as a subtractive process where the transmittance of the layer is $\tau = \sqrt{C}$ (the observed color is the transmittance squared since light must travel through the layer twice). Adding a second layer of the same pigment therefore amounts to squaring the transmittance, and hence also the observed color.

We need to be able to continuously modify the color, rather than proceeding with discrete layers or squaring operations. A full, physically based analysis could be based on the description of a continuous pigment layer thickness and the resulting exponential attenuation. Rather we obtain similar phenomenological effects by introducing a pigment density parameter d, which provides the ability to continuously reinforce or attenuate the pigment density, where $d = 1$ is the nominal density corresponding to the base color C chosen by the user. We then modify the color based on d by removing a value proportional to the relative increase (respectively decrease) in density and the original light attenuation $1 - C$. The modified color C' for density d is thus given by

$$\begin{aligned} C' &= C(1-(1-C)(d-1)) \\ &= C-(C-C^2)(d-1) \end{aligned} \quad (1)$$

Note that the value $d = 2$ corresponds to the two-layer case discussed above and to a squared color.

4.1.2 Pigment density variation

As described in section 2 the density of pigments varies in several ways. We propose to add three layers, one for each effect: turbulent flow, pigment dispersion and paper variations. No physical simulation is performed. Instead, each layer is a gray-level image whose intensity gives the pigment density as follows: An image intensity of $T \in [0,1]$ yields a density $d = 1 + \beta(T - 0.5)$, which is used to modify the original color using formula 1. β is a global scaling factor used to scale the image texture values and allow arbitrary density modifications. The three effects are applied in sequence and each can be controlled independently (see Figure 4-(g,h,i)). The computation is done per pixel using a pixel shader. The paper layer is applied on the whole image whereas the other layers are applied only on the object projection.

The user is free to choose any kind of gray-level textures for each layer. We found it convenient to use Perlin noise textures [Perlin 1985] for the turbulent flow, a sum of gaussian noises at various scales for pigment dispersion and scanned papers for the paper layer.

4.1.3 Other watercolor effects

The pigment density treatment recreates the traditional texture of real watercolor. We then add, as in most previous techniques, several typical watercolor effects:

Edge darkening: Edges are darkened using the gradient of the abstracted image (see Figure 4-(f)). The gradient is computed on the GPU using a fast, symmetric kernel:

$$\Delta(p_{x,y}) = |p_{x-1,y} - p_{x+1,y}| + |p_{x,y-1} - p_{x,y+1}|$$

The gradient intensity (used to modify the pigment density using formula 1) is computed by averaging the gradient of each color channel.

Any other gradient computation could be used depending on the compromise between efficiency and realism needed by the user. Our method is clearly on the efficiency side.

Wobbling: The abstracted image is distorted to mimic the wobbling effect along edges due to the paper granularity. The x offset is computed with the horizontal paper gradient, and the y offset with the vertical paper gradient (see Figure 4-(e)). The user can change the paper texture resolution: Indeed, the real process involves complex interactions and using the paper texture directly may produce details at too fine a scale. By decreasing the resolution we keep the overall canvas structure while decreasing the wobbling frequency. We could have also used another texture for this effect as in [Chu

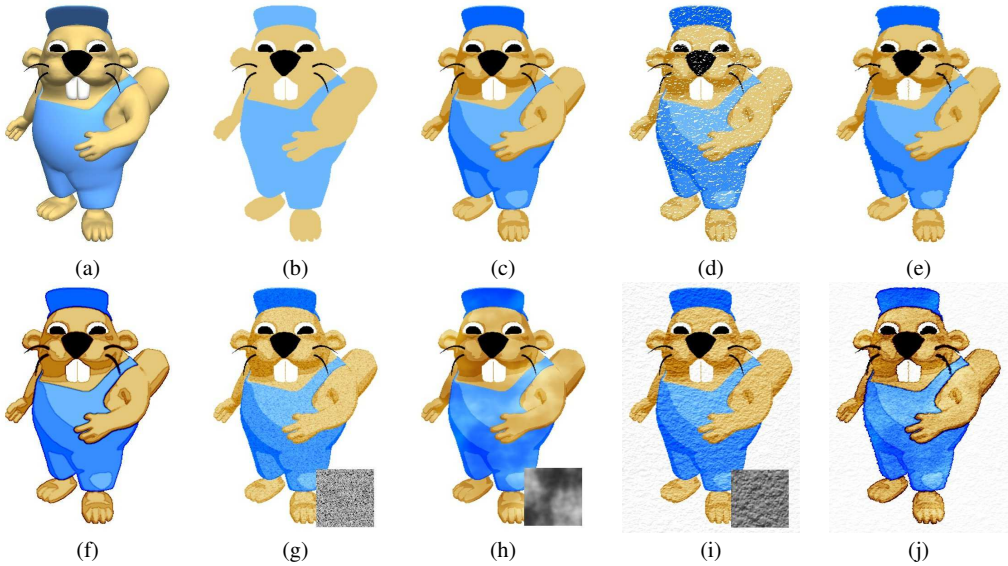


Figure 4: Static pipeline illustrated for a 3d model: (a) 3d model, (b) original color chosen by the user, (c) toon shading, (d) dry brush (not kept for this example), (e) wobbling, (f) edge darkening, (g) pigment dispersion layer, (h) turbulence flow layer, (i) paper, (j) final result.

and Tai 2005]. As a side effect, the paper offset adds noise along edges, which decreases the aliasing effect.

Dry brush: The dry brush effect occurs in watercolor painting when a nearly-dry brush applies paint only to the raised areas of a rough paper. We simulate this effect with a simple threshold of the paper height (represented by the paper texture intensity) as illustrated in Figure 4-(d).

This effect has not proven to be very useful in practice because it is applied to the whole scene. However one can easily add color conditions or masks in the pipeline to selectively apply such an effect.

4.2 Abstraction

The watercolor rendering obtained by our pipeline generally seems too detailed compared to a painting done by hand. Indeed, the small details produced by a 3d rendering are often "too perfect", which is a common issue in computer graphics. Similarly, if we take a photograph as input, the result appears too precise.

We propose to use various abstraction steps to simplify the shapes and to abstract the color and the illumination of our input data before adding the watercolor effects already described. The first abstraction step aims at reproducing uniform color regions by abstracting the illumination in the case of a 3d model (see Section 4.2.1) or segmenting the original image (see Section 4.2.2) and the second abstraction step aims at discarding the small regions that remains after the first step (see Section 4.2.3).

4.2.1 Abstraction of the illumination of a 3d model

In watercolor painting, shading is usually performed by compositing pigment layers. In wet-in-wet painting, this superposition results in a smooth shading, whereas in wet-on-dry painting, it results in sharp color areas. To obtain these shaded areas, a toon shader as described in [Lake et al. 2000] is used. A smooth transition between areas can be obtained by smoothing the toon texture (Figure 6). We apply toon shading using our color modification method so that it stays consistent with the rest of the pipeline. The user thus simply has to provide a gray-level 1d texture.

Figure 6: Toon shading and the corresponding toon texture

Depending on the original 3d model and even with a toon shading there are still sometimes lots of details. Therefore, we propose to smooth the normals of the model before computing the illumination. The normal smoothing is applied by replacing a vertex normal by the average of its neighbors normals. This process can be repeated to obtain a higher smoothing. Normal smoothing removes shading details but preserves silhouettes (Figure 7).

Figure 7: Normal smoothing: by averaging the normals the user simplifies the shading progressively to the desired level.

4.2.2 Abstraction of the color of an image

When dealing with a photograph, we no longer have uniform color regions as given by the toon shader. To obtain such regions, we

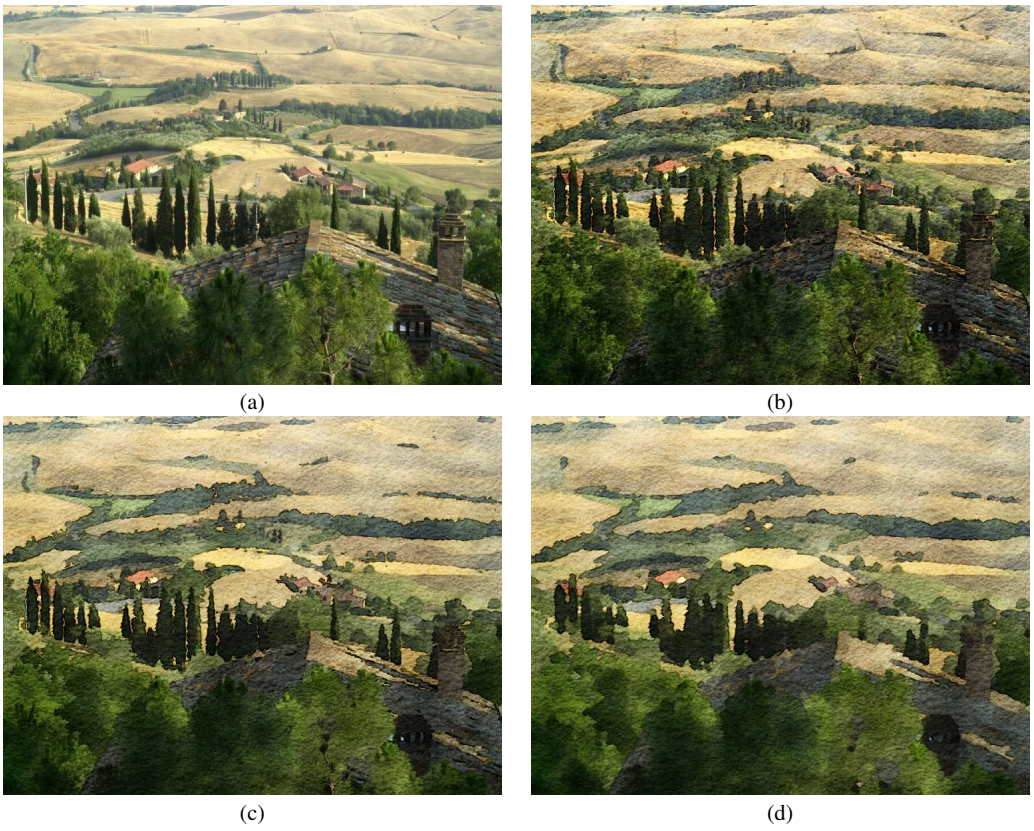


Figure 8: Image abstraction: (a) original image, (b) without abstraction: direct application of watercolor effects, the image is too detailed, (c) result after segmentation, (d) final result with segmentation and morphological smoothing.

propose a color segmentation step. We use a mean shift algorithm [Comaniciu and Meer 1999] to perform such a segmentation. This gives a less detailed image as presented Figure 8-(c) showing the uniform color regions typical in watercolor.

4.2.3 Morphological Smoothing

After the previous step, either segmentation or illumination smoothing, some small color regions remain. We allow the user to apply a morphological smoothing filter (sequence of one opening and one closing) to reduce color areas details. The size of the morphological kernel defines the abstraction level of color areas and silhouettes (Figure 8-(d)). It can be viewed as the brush size. The same kind of 2d approach is used by [Luft and Deussen 2005], except that they use a Gaussian filter and thresholds to abstract their color layers.

5 Temporal coherence

In the static version of our watercolor pipeline, we have shown that a convincing watercolor simulation may be created by using texture maps that represent both low frequency turbulent flow and high frequency pigment dispersion. Using this method in animation leads to the "shower door" effect, where the objects slide over the pigments textures. Ideally, we would like the pigment effects to move coherently with each object, rather than existing independently.

While it is tempting to simply map pigments textures onto each object (as in [Lum and Ma 2001] for example), this leads to problems with ensuring the scale and distribution of noise features. For example, the features of a pigment dispersion texture that appears correct from a specific camera position may blur if the camera is zoomed out. Therefore, we seek a compromise between the 2d location of pigments and the 3d movements of objects.

This question has already been addressed in the case of the canvas, leading to two approaches of a dynamic canvas [Cunzi et al. 2003; Kaplan and Cohen 2005]. The problem there was to match the canvas motion to the camera motion. In the watercolor case, we want the pigment textures to follow each objects motion, not just the camera. We thus propose two different methods that each extend one of the previous dynamic canvas approaches.

As far as the abstraction steps are concerned, the toon shader and normal smoothing are intrinsically coherent since they are computed in object space. On the other hand, the morphological filter is not coherent from frame to frame, since it is computed in image space. Solving this question would probably need to follow color regions along the animation as in "video tooning" [Wang et al. 2004]. Such a method would imply the loss of interactivity and thus we have preferred to concentrate here on the watercolor effects part of the pipeline and leave the incoherency of the morphological smoothing for future work.

5.1 Attaching multiple pigment textures in object space

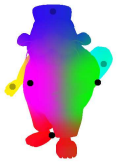
Our first method takes inspiration both from Meier's work on painterly rendering [Meier 1996] and Cunzi et al.'s work on a dynamic canvas [Cunzi et al. 2003].

(a) (b) (c)

Figure 9: (a,b) Our particle method illustrated with a chessboard texture. (a) Rotation: 2d textures follow 3d particles (black dots), (b) z-translation: An infinite zoom of the texture maintains a constant frequency. (c) Result when using our pigment and paper textures.

We decompose the possible movements of the object between translation along the $z-$axis and the other transformations. The idea is that, when an object is moving towards (or away from) the camera, we have to maintain a constant frequency of pigments while the size of the object projection increases or decreases in image space. For the other motions the pigments have to follow the object without being distorted by the perspective projection.

Like Meier, we use particles (points on the 3d mesh) to attach our drawing primitives to the object. Our primitives are the pigment density textures. They are much larger than brush strokes, therefore we need far fewer particles.



The particles are distributed on the mesh by projecting onto the mesh regularly sampled points of the object bounding box. In practice, for a simple object that stays totally in the camera field of view, we use 6 particles (the middles of the 6 faces of the bounding box), shown by the black dots on the right hand side. Pigment textures are then drawn in image space centered on each particle projection, and blended while rendering the mesh. The blending amount of each texture is defined for a vertex according to the 3d distance between the vertex and each particle. The closer a vertex is to a particle, the more the corresponding texture will be taken into account (see the blending in false colors). Such a blending avoids the popping effect that occurs in painterly renderings when strokes appear due to visibility. Indeed, a pigment texture attached to a non visible particle can continue to influence the final rendering, allowing for a continuous evolution. The result is that pigments are drawn in 2d space but move according to the 3d object motion (fig.9-(a)).

This does not define how to scale the texture according to a $z-$translation. As pigments are defined in screen space, we would like to keep their frequency constant. But if we directly keep the texture scale unchanged, the object will seem to slide on the texture during a zoom. To avoid this effect, an infinite zoom in the texture is used, as described in [Cunzi et al. 2003][1]. Such an infinite zoom is produced by cycling continuously between successive octaves (that is successive scales) of the pigment textures. Here we use four octaves. The observer seems to zoom into the pigment textures, while the frequency of the pigments stays constant (fig.9-(b)).

5.2 3D Animation of 2D Flow Effects

As pointed out in [Kaplan and Cohen 2005], no transformation on a 2d texture map can exactly match motion fields produced by arbitrary movements in a 3d environment. An alternative solution is to move the constituent elements of the 2d map in 3d and then to reproject those elements back to 2d to create a new texture map. We associate noise features with 3d particles on the surface of the objects. At runtime, we project those particles into screen space and draw features into a 2d texture, reconstructing new noise and turbulence textures for every frame (see Figure 10).

[1]The source code is available at `artis.imag.fr/Members/Joelle.Thollot/dynamic_canvas/`

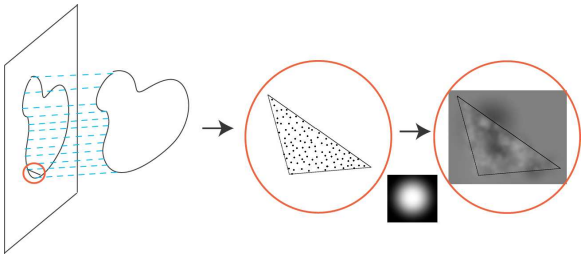


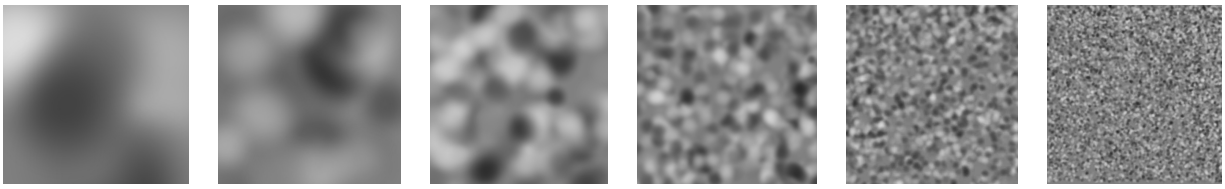
Figure 10: An overview of the interactive noise generation system. First, the models vertices are projected to screen space. Next, the system determines the amount of screen space occupied by each visible triangle. This determines the number of randomly positioned features to draw to create a noise texture of the appropriate frequency. Randomly colored, alpha blended quadrilaterals of that frequency are drawn at feature locations, creating the texture. The alpha texture is shown inset.

Turbulent flow can be represented as a harmonic summation of noise textures [Perlin 1985]. Noise is characterized by randomly valued (a color value typically) changes in a field of arbitrary dimension where the changes in value occur with a constant spatial separation, referred to as frequency f. *Features* are the locations at which changes in the field occur. Therefore, following the Perlin noise, we desire a continuously generateable 2d noise texture with the following properties, 1) features occur at a specific frequency, 2) feature values are pseudo-random, 3) frequency range is band-limited, 4) features move with the surfaces. Note that properties 1 and 4 are contradictory and may not be simultaneously achievable for arbitrary transformations.

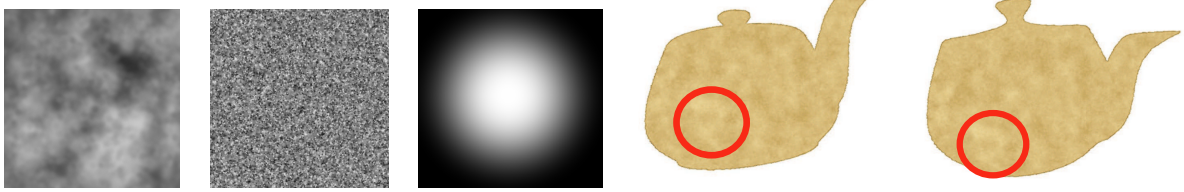
An approximation of the 2d Perlin noise may be obtained by interpolating between random grayscale values located at grid points, representing features, placed at distance f apart, where f is the desired frequency of the noise [Ebert 1999]. Though we cannot use a grid, since our feature points move in 3d, this is the basic approach we adopt.

A noise feature is represented in our program with a structure consisting of random barycentric coordinates, $\beta = b_0, b_1, b_2$ and a random grayscale color, c. A feature of frequency f may be drawn using a quadrilateral in screen space centered at β with width and height f, using an alpha texture defined by the function $6t^5 - 15t^4 + 10t^3$ [Perlin 2002] (shown in Figure 11d) and color c. The alpha texture is used to emulate interpolation between adjacent features.

A noise texture N_f is computed as follows. First, for every visible triangle (surface face), t_i, we compute the amount of image space, λ_i it occupies. Then, we draw $\lambda_i/(f*f)$ features for every t_i. The location of each feature is calculated by using each t_i's vertices (projected into screen space) as control points for the features



(a) Noise textures



(b) (c) (d) (e) (f)

Figure 11: (a) Progressive levels of noise generated interactively with increasing frequencies. (b) Compositing noise levels from (a) yields a turbulence texture or (c) a pigment texture - using only high frequency noise textures. (d) The alpha texture used to draw the features, (e,f) Notice how the pigments and turbulence features follow the surface. No paper texture is used for these images.

barycentric coordinates. This creates a noise texture with average feature distribution of frequency f.

A turbulence texture may be created by summing a series of such noise textures - a typical series of frequencies might be, for base frequency $F_0 = f, F_1 = f/2, F_2 = f/4 \cdots F_K = f/2_K$ - by blending them with the contribution of each N_j being defined by the harmonic function $1/f$. Examples are shown in Figure 11.

5.2.1 Minimizing Artifacts

This method yields an average frequency distribution of f rather than an exact distribution. Two problems may occur because of this random placement. First, we are not guaranteed to completely cover the image space. Attempting to order the spacing of the particle placement may not be useful since we cannot predict what type of perspective deformation the triangle will undergo. Therefore, the screen color is initially cleared to a flat middle gray; features then color the screen as an offset from a strict average rather than defining the actual screen color themselves as in standard noise. This may overly weight the screen to gray, but if our random distribution of feature locations is good (see [Turk 1990] for a discussion), then this problem will be minimal and in practice, this seems to be true. Second, we may create hotspots of feature placement when features overlap. Again, a good initial distribution of features will minimize this problem, but in practice, the blending function is smooth enough that this issue is not apparent.

The amount of screen space occupied by each surface face λ_i changes with camera motion. Because the number of features drawn is dependent on this value, new features may come into and out of existence very quickly which will cause popping artifacts if not handled with care. We minimize this artifact by taking advantage of the fact that our turbulence texture is the sum of a series of noise textures. Continuity is maintained by moving features between each noise texture N_j and textures N_{j+1} and N_{j-1}. When λ_i grows larger, features are moved from N_{j-1} to N_j proportional to the amount of screen space gained. Conversely, when λ_i shrinks, features are moved from N_{j+1} to N_j and from N_j to N_{j-1}. Because features are transitioned between noise textures at different frequencies an ordering between noise texture levels is created. Therefore, the turbulence appears consistent at different resolutions, i.e. if an object is zoomed away from the viewer, the low frequency noise is transitioned to high frequency noise so the visual characteristics of the turbulence appears consistent. In order to maintain the frequency distribution, one feature is moved between each noise texture levels at a time, linearly scaling the width between levels. With this method, popping artifacts occur only at the highest and lowest frequencies. At high frequencies, the artifacts were very tiny and so, not very noticeable. Low frequency errors can be avoided by making the lowest frequency larger than the screen.

As a final consideration, the surface meshes in the scene may be composed of triangles that occupy screen space of less than $f * f$ pixels, in which case *no* face will draw any features of the desired frequency! We create a hierarchy of faces, grouping faces into superfaces as an octree. We then analyze the screen space occupied by the superfaces in order to determine the number of features to draw at low frequencies. Only normal similarity is considered when creating our hierarchies, and though better schemes may exist for grouping faces, this method works well in practice.

6 Results and Discussion

We show some results obtained by our method using as input either a photograph or a 3d model Figure 12. More watercolor results and some videos can be found in `artis.imag.fr/Membres/Joelle.Thollot/Watercolor/`.
As most of the work is done by the GPU, the static pipeline runs at a speed similar to a Gouraud shading (between 25 frames per second for 25k triangles downto 3 frames per second for 160k triangles) for a 750×570 viewport on a Pentium IV 3GHz with a GeForce FX 6800.

Evaluating the visual quality of watercolor renderings is not an easy task. Figure 13 shows a comparison we have made with Curtis original method [Curtis et al. 1997] that was a full physical simulation. Our method allows us to produce images that stay closer to the original photo while allowing the user to vary the obtained style. On the other hand Curtis' result benefits from his simulation by showing interesting dispersion effects. However, it is precisely these dispersion effects that make that method unsuitable for animation. In light of this, it is unclear whether we lose anything by not doing a full fluid flow simulation in an animated or interactive environment.

The temporal coherence effects of our two methods yield distinct results with several tradeoffs. The texture attachment method yields good results for a single object viewed with a static background.

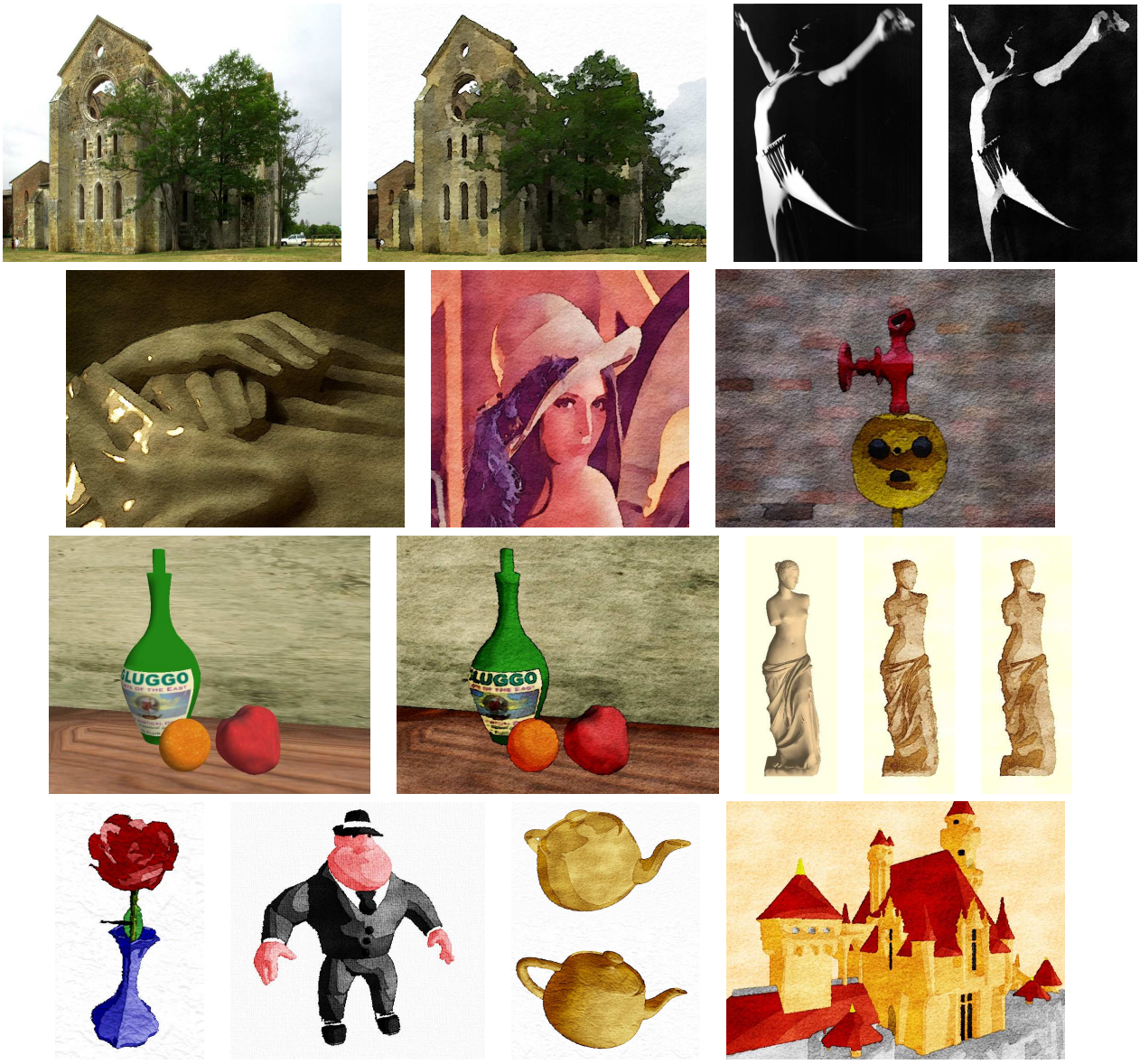


Figure 12: Watercolor-like results. The first line shows the original photograph and the resulting watercolor-like image. The second line shows other filtered photographs. The third line shows a gouraud shaded 3d scene and the resulting watercolor rendering with two versions of the Venus model without and with morphological and normal smoothing. The fourth line shows more watercolor-like 3d models.

(a) (b) (c) (d)

Figure 13: Comparison with Curtis result: (a) original image, (b) Curtis watercolor result, (c,d) our results.

The number of particles used to render the animation has to be a compromise between two effects: Too many particles would tend to blur the high frequencies due to texture blending whereas not enough particles creates scrolling effects due to independent movements of each patch of texture. Some distortion occurs at pixels far from attachment points, but in practice, using at least 6 attachment points minimized this effect. Ideally the number of particles should be adapted to the viewpoint. For example after a zoom, the particles may become too far from each other and some new particles should be added.

Calculating flow textures interactively matched the movement of the turbulent flow and pigment dispersion exactly with the motion of the objects in the scene, making it useful for complex animations in immersive environments. Unfortunately, this method is much more expensive to compute since between 1-10 noise textures and 1-2 turbulence textures must be interactively constructed for every frame, yielding as few as 1 frame a second for 60k triangles. Finally, small scale popping artifacts may be visible yet we found this was not very noticeable for complex turbulence textures possibly due to the low alpha value assigned to such high frequency features. We allowed the user to control several variables (such as which frequencies to use) in order to construct the pigment and turbulence textures. This yielded a wide range of visual styles yet those styles did not always exactly correspond with those produced by the static method. Reasonable fidelity to the static method was achievable as long as the frequencies we used corresponded well with the static pigment repartition texture.

7 Conclusions

We have presented a watercolor rendering technique that is fully controllable by the user, allowing the production of either coherent animations or images starting from a 3d model or a photograph. Our framework is interactive and intuitive, recreating the abstract quality and visual effects of real watercolors.

Each step of the pipeline can still be improved, especially by offering the user a choice between slower but better methods for each effect. This can be suitable for producing movies when high quality is most important. Ideally our idea would be to keep our pipeline as a WYSIWYG interface for preparing an offline full computation of more precise effects.

As mentioned in the paper, there are several issues that we want to address in the future. The diffusion effect would be interesting to recreate. We can think of using work like [Chu and Tai 2005] to perform diffusion computation but it opens the question of how to control such an effect when dealing with an already given scene. To stay in our philosophy we have to think of simple and intuitive ways of deciding which part of the image must be concerned with the diffusion.

A lot of questions remain concerning the temporal coherence. The methods we propose do not target the morphological smoothing step or any image processing. Taking inspiration from "video tooning" [Wang et al. 2004] we would like to address this problem in our pipeline. Another possibility would be to use image filters like in [Luft and Deussen 2005] to decrease the popping effects without adding too much computation.

Acknowledgments

This research was funded in part by the INRIA Action de Recherche Cooprative MIRO (`www.labri.fr/perso/granier/MIRO/`).

Thanks to Laurence Boissieux for the 3d models Fig 1-(a,b), 4 (©Laurence Boissieux INRIA 2005). Thanks to Gilles Debunne for the reviewing and the web site.

References

BURGESS, J., WYVILL, G., AND KING, S. A. 2005. A system for real-time watercolour rendering. In *CGI: Computer Graphics International*, 234–240.

CHU, N. S.-H., AND TAI, C.-L. 2005. Moxi: real-time ink dispersion in absorbent paper. In *Siggraph 05*, ACM Press, 504–511.

COMANICIU, D., AND MEER, P. 1999. Mean shift analysis and applications. In *ICCV (2)*, 1197–1203.

CUNZI, M., THOLLOT, J., PARIS, S., DEBUNNE, G., GASCUEL, J.-D., AND DURAND, F. 2003. Dynamic canvas for immersive non-photorealistic walkthroughs. In *Graphics Interface*, A K Peters, LTD., 121–129.

CURTIS, C. J., ANDERSON, S. E., SEIMS, J. E., FLEISCHER, K. W., AND SALESIN, D. H. 1997. Computer-generated watercolor. In *Siggraph 97*, ACM Press, 421–430.

EBERT, D. 1999. Simulating nature: From theory to application. In *Siggraph Course*. ACM Press.

JOHAN, H., HASHIMOTA, R., AND NISHITA, T. 2005. Creating watercolor style images taking into account painting techniques. *Journal of Artsci*, 207–215.

KAPLAN, M., AND COHEN, E. 2005. A generative model for dynamic canvas motion. In *Computational Aesthetics*, 49–56.

LAKE, A., MARSHALL, C., HARRIS, M., AND BLACKSTEIN, M. 2000. Stylized rendering techniques for scalable real-time 3d animation. In *NPAR: International symposium on Non-photorealistic animation and rendering*, ACM Press, 13–20.

LEI, E., AND CHANG, C.-F. 2004. Real-time rendering of watercolor effects for virtual environments. In *PCM: Pacific Rim Conference on Multimedia*, 474–481.

LUFT, T., AND DEUSSEN, O. 2005. Interactive watercolor animations. In *PG: Pacific Conference on Computer Graphics and Applications*, 7–9.

LUM, E. B., AND MA, K.-L. 2001. Non-photorealistic rendering using watercolor inspired textures and illumination. In *PG: Pacific Conference on Computer Graphics and Applications*, 322–331.

MEIER, B. J. 1996. Painterly rendering for animation. In *Siggraph 96*, ACM Press, 477–484.

PERLIN, K. 1985. An image synthesizer. In *Siggraph 85*, ACM Press, 287–296.

PERLIN, K. 2002. Improving noise. In *Siggraph 02*, ACM Press, 681–682.

SMALL, D. 1991. Modeling watercolor by simulating diffusion, pigment, and paper fibers. In *SPIE*, vol. 1460, 140–146.

TURK, G. 1990. Generating random points in triangles. In *Graphics Gems I*, A. Glassner, Ed. Academic Press.

VAN LAERHOVEN, T., LIESENBORGS, J., AND VAN REETH, F. 2004. Real-time watercolor painting on a distributed paper model. In *CGI: Computer Graphics International*, 640–643.

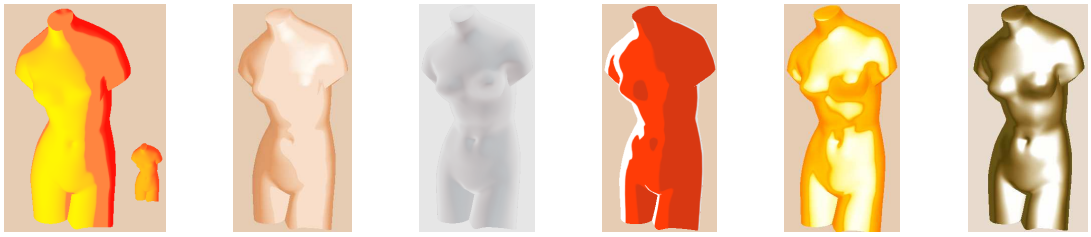
VAN ROOJEN, J. 2005. *Watercolor Patterns*. Pepin Press / Agle Rabbit.

WANG, J., XU, Y., SHUM, H.-Y., AND COHEN, M. F. 2004. Video tooning. In *Siggraph 04*, ACM Press, 574–583.

X-Toon: An Extended Toon Shader

Pascal Barla, Joëlle Thollot
ARTIS GRAVIR/IMAG INRIA* Grenoble France

Lee Markosian
University of Michigan



(a) (b) (c) (d) (e) (f)

Figure 1: Some example effects achieved by our extended toon shader: continuous levels of abstraction (a); abstraction of near-silhouette regions: smoothing (b), and opacity (c); backlighting (d); and highlighting: plastic (e), and metal (f).

Abstract

Traditional toon shading uses a 1D texture that describes how tone varies with surface orientation relative to a given light source. In this paper we describe two extensions to the basic algorithm that support view-dependent effects. First, we replace the 1D texture with a 2D texture whose second dimension corresponds to the desired "tone detail," which can vary with depth or surface orientation. This supports effects such as levels-of-abstraction, aerial perspective, depth-of-field, backlighting, and specular highlights.

Second, we further extend the toon shader to use a modified normal field that can range from the original normals to a simpler set of normals taken from an "abstracted shape." A global shape detail parameter determines the degree of interpolation between the original and abstracted normal fields. We explain how to implement these ideas efficiently on the GPU via vertex and fragment shaders, and discuss ways to extend our approach to alternative tone and shape detail maps.

1 Introduction

Over the past decade, "toon" shading has proven popular in a variety of 3D renderers, video games, and animations. The idea is simple but effective: extend the lambertian shading model by using the computed illumination (a dot product between a light vector and the surface normal) to index into a 1D texture that describes how the final shading varies from "dark" to "light" regions. The designer controls the behavior of the shader by creating a 1D texture, typically composed of two or three regions of constant color, to mimic the flat regions of constant colors found in comics and traditional animation. Toon shading can be implemented efficiently via vertex and fragment programs on modern GPUs.

On the other hand, toon shading does not reflect the desired level of abstraction (LOA) of a surface. Such LOA behavior plays an important role in traditional media, however. Often, some objects are considered less important (e.g., characters in the background) and thus are depicted with greater abstraction [McCloud 1994]. In paintings and drawings, an effect known as aerial perspective makes objects in the background appear desaturated and less detailed than those in the foreground. And in scientific illustration, a technique similar to depth-of-field is used to focus on a region of a shape by decreasing contrast or opacity in less-important or "out of focus" parts of the surface [Wood 1994].

Conventional toon shading is view-independent, and so cannot represent plastic or metallic materials, for which view-dependent highlights are of primary importance. Similarly, it cannot support the view-dependent backlighting effects, often used in traditional comics and animation, in which a virtual "back light" illuminates the surface near the silhouette.

Finally, in conventional toon shading, every surface location is rendered with full accuracy, so that even small shape features are depicted by the shading (at least for some light directions). This can be desirable, but often designers working traditionally apply a degree of abstraction so that small shape features are omitted. A similar ability to depict an abstracted version of the shape is thus desirable in an automatic toon shader.

In this paper we describe X-Toon, a toon shader that supports view-dependent effects through two independent extensions to conventional toon shading. The first incorporates a notion of **tone detail**, so that tone varies with depth or orientation relative to the camera. For this, we replace the conventional 1D texture used in toon shading with a 2D texture, where the second dimension corresponds to desired tone "detail" (see below). We describe several ways to define the additional texture coordinate.

Our second extension, presented in Section 4, lets us vary the perceived **shape detail** of the shading. We achieve this by using a modified normal field defined by interpolating between normals of the original shape and normals of a highly abstracted shape. This approach has the advantage of abstracting the shading from a shape (while preserving silhouettes).

Note that "detail" here corresponds to visual abstraction: less detail means greater abstraction. We use the term LOA instead of LOD to emphasize that our goal is visual abstraction, not increased rendering speed (the usual motivation for LODs in computer graphics).

The main contribution of this paper is a simple, unified approach that lets a designer intuitively achieve a variety of effects, including those that rely on *correlating* desired tone detail with the underlying tone value, as explained in Section 3.

*ARTIS is a research team of the GRAVIR/IMAG laboratory, a joint unit of CNRS, INPG, INRIA and UJF.

2 Related Work

Toon shading is available in production software such as Renderman, Softimage, Maya, and Autodesk 3ds Max. It has been used in many video games, TV shows, and feature films. The first published description of the method is due to Lake *et al.* [2000].

Related methods include the technical illustration shader of Gooch *et al.* [1998] and the "Lit Sphere" of Sloan *et al.* [2001]. The latter uses a painted spherical environment map instead of a 1D texture. Neither handles the view-dependent behaviors we are targeting.

Mip-maps provide one means of taking depth into account as an attribute. Klein *et al.* [2000] used specially constructed mip-maps (called "art maps") to achieve constant-sized strokes in textures applied to 3D scenes. The "tonal art maps" of Praun *et al.* [2001] extended this approach to take lighting into account in hatching patterns.

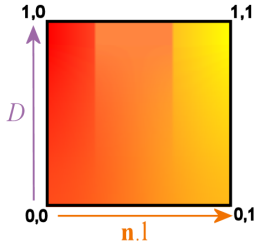
We could similarly define specialized art maps for the management of 1D toon textures with LOAs. Instead, we prefer a more general approach whereby a 2D texture represents an ordinary toon texture at a continuous range of abstraction represented by the added dimension. This way, the LOA selection mechanism can be more general than the fixed depth-based computations used in mip maps. It also lets designers create LOA toon textures directly as 2D images, e.g. by painting them in a paint program. The only requirement is to understand that the 2 dimensions of the texture correspond to tone and LOA, respectively. This approach retains the simplicity of the classic toon shader and does not constrain the designer with a set of predefined behaviors and discrete LOAs. We compare our approach with a 1D mip-map approach in the accompanying video.

Anjyo *et al.* [2003] described techniques to add stylized highlights to a classic toon shader, thus taking viewing orientation into account. Their technique, which acts on top of a toon shader, supports abstraction of highlights through the use of translation, rotation, splitting and squaring operations. Other methods could be devised to control the highlights independently of the tone computation (e.g., using a 1D highlight texture). However, once the properties of a highlight (e.g., size, softness, color) are chosen, they remain the same under any viewing condition. Thus, the method of Anjyo *et al.* only considers highlights with hard contours, a single color and a fixed size. Our approach offers the ability to correlate properties of the highlights with the tone value, in a way precomputing the response of the highlight to changing viewing conditions: highlights can change in softness, color and size dynamically. We compare our approach with an independent specification of highlights and tone in the accompanying video.

Anjyo's method does permit modifications to the shape of the highlight, thus controlling what we term its "shape detail." Achieving such an effect in a toon shader is an interesting avenue for future work (see Section 5).

Other methods manipulate the amount of shape detail conveyed in non-photorealistic renderings. The "graftal textures" of Markosian *et al.* [2000] manage small detail elements ("graftals") representing leaves, hair, etc., introducing and removing them according to a desired level of detail. Deussen and Strothotte [2000] do something similar for 3D models of trees rendered in a pen-and-ink style. The designer can choose the desired level of abstraction via a slider. Our goal is to have similar capabilities in a toon-shader.

3 Tone detail



We extend the classic 1D toon texture by adding a vertical "detail" axis (corresponding to LOA) to build a 2D toon texture. The horizontal axis corresponds to tone as in a classic toon texture – the texture coordinate along that axis is derived from a standard lambertian shading computation: $\mathbf{n} \cdot \ell$, where $\mathbf{n}$ is the unit surface normal and ℓ is the unit light direction. The vertical axis corresponds to tone detail: each value along this axis (labelled D for detail) corresponds to its own 1D toon texture. The whole 2D texture can thus be regarded as a stack of 1D toon textures with increasing "detail." The designer is free to create this 2D texture using any image processing or painting tools available. We generally found it convenient to start with a standard 1D toon texture at $D = 1$ and apply image processing transformations down the D axis to account for detail loss. In this paper and accompanying video we provide many example textures that have been created with this approach, and we demonstrate the resulting behavior of the extended toon shader.

Once the 2D texture is defined, the designer must choose an attribute (e.g., orientation or depth) that will control the tone detail, and provide functions called **attribute maps** that map the attribute to a detail value D at each location on a surface. This formulation is general in that any attribute can be chosen, depending on the application goals.

In this paper, we consider view-dependent attributes. In the next two sections we describe how to compute depth-based and orientation-based attribute maps to achieve LOAs, aerial perspective, depth-of-field, backlighting, and specular highlights.

3.1 Depth-based attribute mapping

We consider two ways to define the "depth" of a point in 3D: we can use its euclidean distance to the viewpoint, or its distance along the focal axis (see Figure 2). The latter assigns the same depth to all the points that lie in a plane parallel to the image plane. Depending on the intended effect, one computation or the other may be preferred. For LOAs and aerial perspective effects, we use distance along the focal axis; for depth-of-field effects, we prefer distance to the eye.

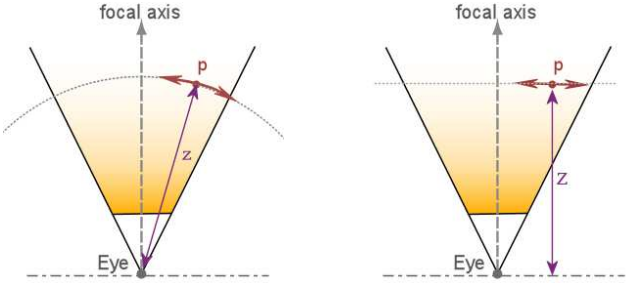


Figure 2: The depth of a point p relative to the eye can be calculated in two ways: (a) The distance between the eye and p or (b) the distance along the focal axis.

(a) (b) (c)

Figure 3: (a) The LOA attribute map used for the venus model, (b) the aerial perspective attribute map used to render the landscape model, and (c) the depth-of-field attribute map and the focus point (blue sphere) used to render the mechanical part model.

We derive a detail value $D \in [0..1]$ from the computed depth z via two user-specified parameters: z_{min}, the distance at which the detail starts decreasing, and $r > 1$, the scale factor that defines the coarsest detail (greatest abstraction) at distance $z_{max} = rz_{min}$. To account for perspective projection, we use the following formula for detail mapping[1]:

$$D = 1 - \log(z/z_{\min})/\log(z_{\max}/z_{\min})$$

$$D = 1 - \log_r(z/z_{\min})$$

For depth-of-field, we use a different formula for the detail computation: given a focus point c in 3D, we compute D as follows[2]:

$$D = \begin{cases} 1 - \log(z/z^-_{\min})/\log(z^-_{\max}/z^-_{\min}), & z < z_c \\ \log(z/z^+_{\max})/\log(z^+_{\min}/z^+_{\max}), & z > z_c \end{cases}$$

with $z^\pm_{\min} = z_c \pm z_{\min}$ and $z^\pm_{\max} = z_c \pm rz_{\min}$.

Depth-based attribute maps are shown in Figure 3. In our implementation, we compute D per vertex in a vertex shader. A pixel shader performs the 2D toon texture lookup using the value of D interpolated at each pixel. In an interactive session, the designer can thus experiment with the z_{min} and r parameters and set the point of focus, and then observe the resulting behavior of the X-Toon shader in real time.

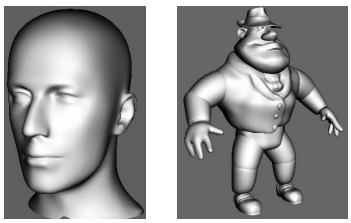
Figure 7 shows depth-based tone LOA effects using 2D textures created in three different ways: in (b) we progressively blurred an input 1D toon texture, in (c) we interpolated between three 1D toon textures with smooth transitions, and in (d) we shifted and lightened a 1D texture to create a "receding shadows" effect. The LOA attribute map is shown in Figure 3(a).

Figure 8 shows two examples of a focus-based detail map. In (a), we used a smooth two-tone texture that converges to a constant color with detail loss. In (b), we use a texture that decreases opacity and contrast with detail loss, and only shadows and highlights are fully opaque. We used these effects to focus attention on a specific depth range of a mechanical model. The depth-of-field attribute map is shown in Figure 3(c), along with the focus point depicted by a small blue sphere.

Finally, Figure 9 shows a landscape rendered with four different X-Toon implementations of aerial perspective. We created these textures using filters such as decreasing contrast (b), converging to a specific hue (c) and (e), and decreasing opacity (f). We did this independently for the main tone values (dark, intermediate, and highlights) so that we can correlate the change made by aerial perspective with a specific tone value (e.g., we make the intermediate colors more transparent with distance). The aerial perspective attribute map is shown in Figure 3(b).

[1] We assume $0 < z_{min} \le z \le z_{max}$.

[2] We assume $0 < z^-_{max} < z^-_{min} < c < z^+_{min} < z^+_{max}$.



(a) (b)

Figure 4: (a) The near-silhouette attribute map used for the face model. (b) The highlight attribute map used for the "Al" model. For these images, the detail scale is inverted, so that dark tones correspond to low detail values. These maps behave differently when the camera or lights are animated.

3.2 Orientation-based attribute mapping

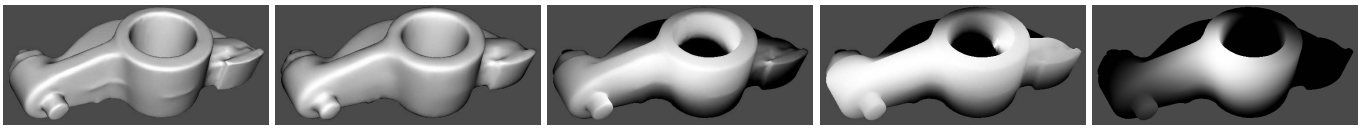
We next consider an attribute map based on the orientation of the surface with respect to the observer. With the appropriate choice of 2D toon texture and policy for computing D, we can achieve effects such as fading of near-silhouette regions, or brightening and coloring of near-silhouette regions to suggest a virtual "backlight." We define the near-silhouette attribute mapping as follows: $D = |\mathbf{n} \cdot \mathbf{v}|^r$, where $\mathbf{n}$ and $\mathbf{v}$ are the unit normal and view vector, respectively, and $r \ge 0$ is a user-defined parameter that controls the magnitude of the effect.

Specular highlights are another view-dependent effect: these depend on the angle between the viewing direction and the light reflection vector, and the specular properties of the material. Depending on the chosen 2D texture, we can model various material highlights intuitively (e.g., plastics or metals). This is better than just compositing a specular layer over the ordinary 1D toon shader, because the 2D texture lets us control the profile of the highlight (smoothness, width, alpha) so that it is correlated with the underlying tone. In our implementation, we use the Phong highlight model to map the highlight attribute values to detail: $D = |\mathbf{v} \cdot \mathbf{r}|^s$, where $\mathbf{r}$ is the light reflection vector at the current surface location and $s \ge 1$ is the "shininess" coefficient set by the designer to control the magnitude of the effect. Orientation-based attribute maps are shown in Figure 4.

In practice, it is straightforward to compute these values in a vertex shader. For accuracy reasons, however, we pass the relevant vectors to a fragment shader where they are interpolated before being used in the detail computation.

Figure 10 shows how we can use the orientation detail map for different purposes. In (b) we use it with a texture similar to the LOA examples in Figure 7 to abstract tonal detail in near-silhouette regions. In (c), we make the surface fade out in near-silhouette regions to yield a "fuzzy" appearance. A different 2D texture is used in (d) to produce an effect we call "backlighting". With the appropriate choice of 2D texture, the thickness and color of backlit regions can be made to depend on the tone value of the underlying 1D toon texture. The near-silhouette attribute map is shown in Figure 4(a).

Figure 11 shows three types of highlights. The first depicts a plastic-like shader, where we control the thickness and opacity of the highlight directly in the texture. The second shows a metallic-like shader, where the highlight has a quality of "glowing" suddenly when the camera moves to a favorable orientation. While the first supports dynamic variation in the size of the highlights, the second controls the smoothness of tone transitions view-dependently. A third example modifies the colors that appear near transitions to highlights. The highlight attribute map is shown in Figure 4(b).



(a) (b) (c) (d) (e)

Figure 5: For each figure, darker tones correspond to normals that are more tilted away from the camera. (a) The normals of the input mesh. (b) Smoothed normals (note the shape details that disappeared in the front of the object). (c) Elliptic normals whose distribution follows the principal direction of elongation of the object. (d) Cylindric normals, oriented in the direction of principal elongation. (e) Spheric normals, highly abstracted from the original model. Note that for every geometric map, the silhouettes are preserved.

4 Shape detail

We now describe how to further modify the shading to depict an abstracted version of the shape. Note that this extension is independent of the tone detail extension and is done view-independently. We achieve it by modifying the surface normal field used in shading computations with a **geometric map**: we map the input mesh to an abstracted shape that acts as a lower bound for shape detail. We ensure this abstracted shape has a direct correspondence with the input surface. The designer controls the "abstraction" of shape detail by setting a blending parameter that controls how the shader interpolates between normals of the input mesh and the abstracted shape. Note that while normal vectors used in shading calculations are modified, vertex locations are not. This has the important property of ensuring consistency when combined with other rendering passes.

We describe four types of abstracted shape: a smoothed version of the original mesh, and an enclosing ellipse, cylinder, and sphere. We show examples of these abstracted shapes in Figure 5, rendered using a simple lambertian shader with the light at the camera to better visualize the abstraction. The dimensions of the ellipse come from a bounding box of the input mesh. To map normals from the input mesh to the sphere, cylinder, or ellipse, we use straightforward analytic mapping techniques. We create the smoothed mesh using a simple normal smoothing technique, and the 1-1 correspondence between mesh vertices defines the mapping. These four types of abstracted shapes provide a great deal of expressiveness: they let us "flatten" the shading in a way that resembles some drawings, paintings and comics. Note that this manipulation of normals to abstract the shading need not be restricted to toon shading.

In our implementation, we precompute and store abstracted normals at each mesh vertex. A vertex shader is used to linearly blend between the input and abstracted normals using a single (global) weight provided by the designer. The resulting vector is renormalized in the pixel shader where the toon texture lookup is performed. While simple, this method works reasonably well, is easy to implement, requires just one extra normal per vertex, and provides interactive feedback when implemented on the GPU. Of course, for the shading to be coherent, the normals of the highly abstracted shape should not exhibit any degeneracies; however we consider this issue to be beyond the scope of this paper.

5 Discussion and future work

We have presented two independent extensions to the original toon shader, aimed at retaining its simplicity while allowing more general behaviors. An important property of the original toon shader is that it is fast. To compare our approach in terms of efficiency, we have made a set of measurements summarized in Table 1. The rendering time of X-Toon for typical scenes is only about 20% slower than the original toon shader, and can thus be used in interactive applications as well as for off-line rendering.

Model	*Size*	*Resolution*	*Toon*	*X-Toon*	*Ratio*
Mech part	$10,000 \circ$,	$640x480$	287	241	1.19
	$20,000 \triangle$	$1280x950$	268	222	1.21
David	$26,051 \circ$,	$640x480$	94	80	1.18
	$49,998 \triangle$	$1280x950$	90	76	1.18
Terrain	$105,152 \circ$,	$640x480$	33	28	1.18
	$208,962 \triangle$	$1280x950$	32	27	1.19

Table 1: Run-time performance (in frames per second) and ratio of toon to X-Toon frame rates for three different models ($\circ =$ vertices, $\triangle =$ triangles) at two different resolutions. These measurements were made on a Pentium 4 with ATI Radeon 9800 GPU.

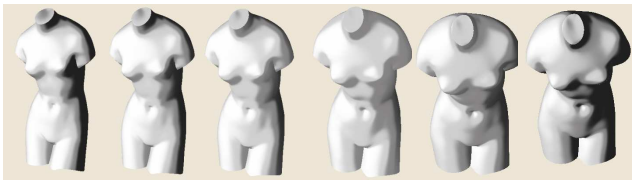


Figure 6: We show six frames of a small animation where coarser detail is automatically selected where the magnitude of optical flow is higher. We used the receding shadow texture from Figure 7.

Although we have focused in this paper on view-dependent attributes, our system can easily be extended to handle other attributes. For example, we can use optical flow to assign less detail to objects as they move faster in image space. To implement this method, we store the world-to-pixel matrix from the previous frame and use it to compute a per vertex displacement vector, measured in pixels. We use the length of this vector to assign a value to D. (See Figure 6.) In future work, we plan to investigate additional attribute detail maps, e.g. based on haloing, labeled importance, points of interest, cast shadows or reflections, and find a way to combine them efficiently.

Our geometric mapping is similar to the one developed by Ni *et al.* [2006], except that they interpolate normals *and* positions using a *local* detail map, and they support the use of multiple shapes (not just two) representing distinct levels of detail. Applying similar ideas in an X-Toon shader is one possibility for future work. Another possibility is to take inspiration from Anjyo *et al.* [2003], and use light maps to control the shape of highlights. We could also consider ways to control detail parameters over time (via keyframing, e.g.) as was done in the same paper.

We have only considered a single global weight to interpolate between the original and abstracted normals. A more flexible approach would be to control shape detail using a locally varying weight. This varying weight could be provided by the designer via a texture painted on the surface, or through an attribute map similar to those we have described for controlling tone detail. Controlling shape and tone detail via the same attribute maps may make sense in terms of providing more consistency in the resulting shading.

6 Conclusions

We have presented two methods for controlling the level of abstraction in toon shading. Our extended toon shader lets the designer correlate a chosen view-dependent attribute (e.g. depth or orientation) with tone detail by means of a 2D toon texture. Through the use of more or less abstracted normals, one can further abstract the shading from the shape it represents. By combining these extensions, a designer can easily create a wide range of effects that were difficult or impossible to achieve with previous approaches.

Acknowledgements

This research was supported in part by the NSF (CCF-0447883), and by a EURODOC grant (Région Rhône-Alpes).

References

ANJYO, K., AND HIRAMITSU, K. 2003. Stylized highlights for cartoon rendering and animation. *IEEE Computer Graphics and Applications 23*, 4, 54–61.

DEUSSEN, O., AND STROTHOTTE, T. 2000. Computer-generated pen-and-ink illustration of trees. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, 13–18.

GOOCH, A., GOOCH, B., SHIRLEY, P. S., AND COHEN, E. 1998. A non-photorealistic lighting model for automatic technical illustration. In *Proceedings of SIGGRAPH 98*, Computer Graphics Proceedings, Annual Conference Series, 447–452.

KLEIN, A. W., LI, W. W., KAZHDAN, M. M., CORREA, W. T., FINKELSTEIN, A., AND FUNKHOUSER, T. A. 2000. Non-photorealistic virtual environments. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, 527–534.

LAKE, A., MARSHALL, C., HARRIS, M., AND BLACKSTEIN, M. 2000. Stylized rendering techniques for scalable real-time 3D animation. In *NPAR 2000: First International Symposium on Non-Photorealistic Animation and Rendering*, 13–20.

MARKOSIAN, L., MEIER, B. J., KOWALSKI, M. A., HOLDEN, L. S., NORTHRUP, J. D., AND HUGHES, J. F. 2000. Art-based rendering with continuous levels of detail. In *NPAR 2000: First International Symposium on Non-Photorealistic Animation and Rendering*, 59–66.

MCCLOUD, S. 1994. *Understanding Comics*. Harper.

NI, A., JEONG, K., LEE, S., AND MARKOSIAN, L. 2006. Multi-scale Line Drawings from 3D Meshes. In *2006 ACM Symposium on Interactive 3D Graphics and Games*.

PRAUN, E., HOPPE, H., WEBB, M., AND FINKELSTEIN, A. 2001. Real-time hatching. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, 579–584.

SLOAN, P.-P., MARTIN, W., GOOCH, A., AND GOOCH, B. 2001. The lit sphere: A model for capturing NPR shading from art. In *Graphics Interface 2001*, 143–150.

WOOD, P. 1994. *Scientific Illustration*. Wiley.

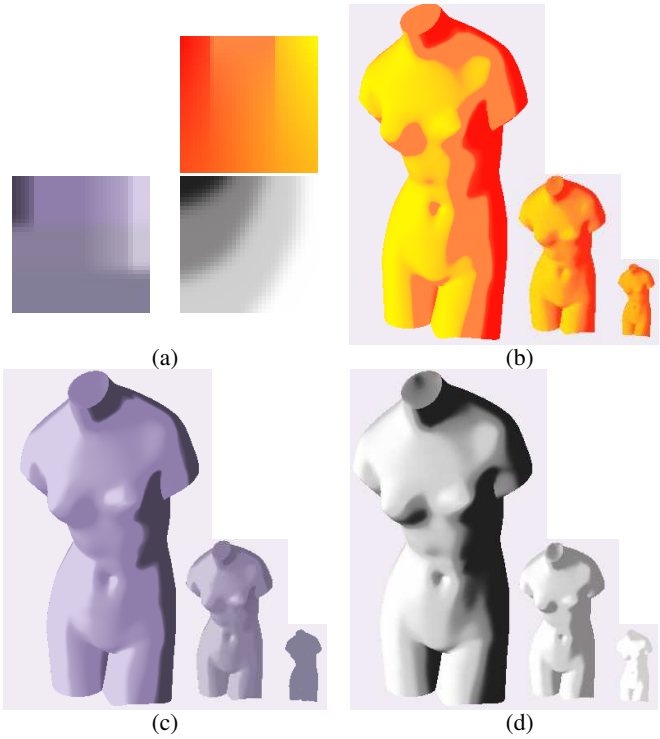


Figure 7: Some examples of the venus model rendered at various LOAs, with the corresponding toon textures in (a). In (b), we use a blurred texture to smooth out the shading with distance; in (c), we mimic a discrete LOA behavior using a texture with smooth steps along the detail axis, hence controlling the interpolation behavior; in (d), we make a receding shadow effect by sliding the darker tones to the left of the toon texture along the detail axis.

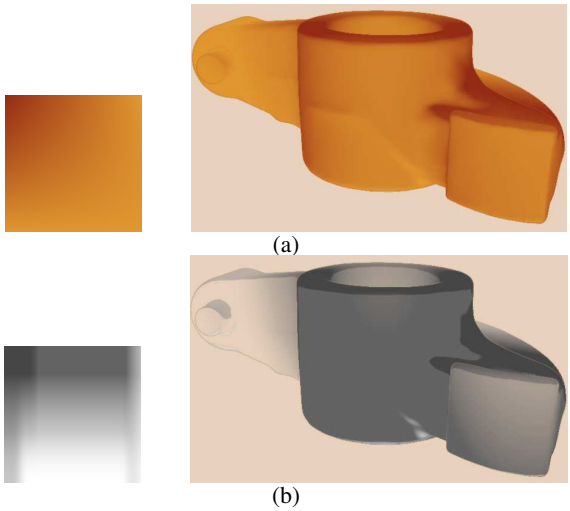


Figure 8: Some examples of depth-of-field shading: in (a), we use a blurred texture; in (b) we use a texture in which contrast and opacity are decreased for intermediate tones.

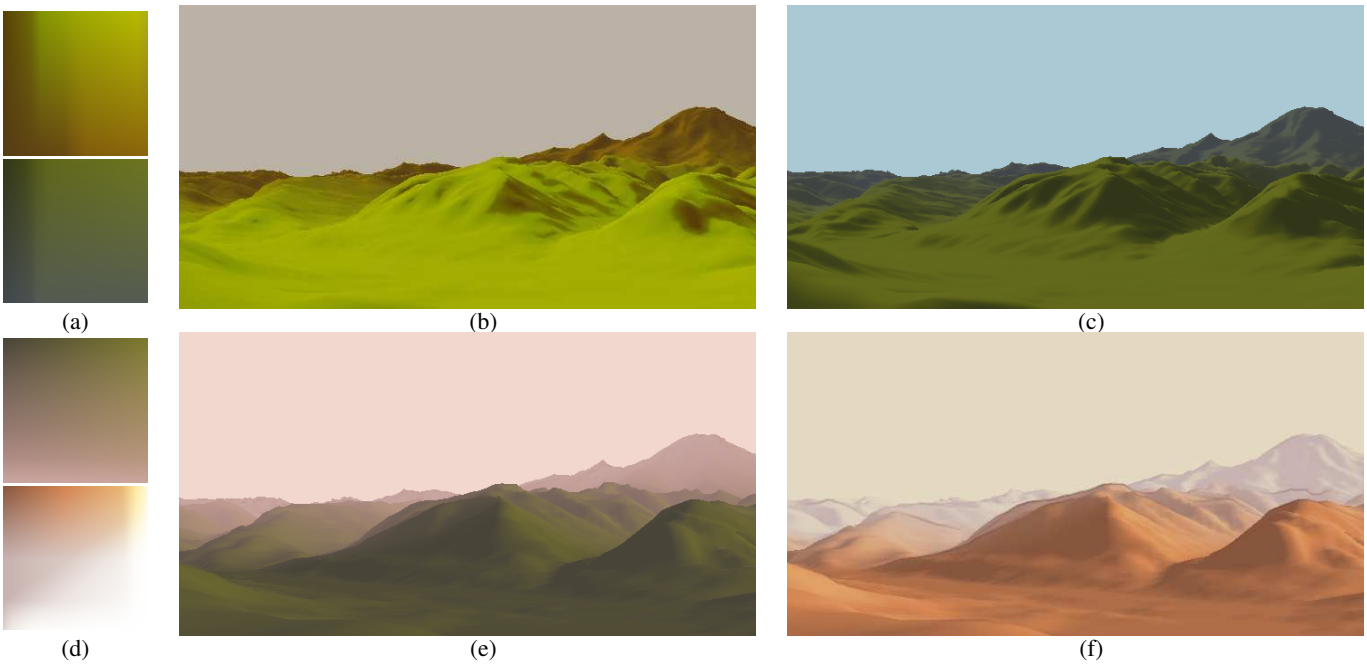


Figure 9: Aerial perspective effects. In (a) we show the two textures used to render the images in (b) and (c): the first one blends from a green-to-brown color ramp to a more uniform one consisting of brown tones; the second one applies a desaturation and shift toward blue. In (d) we show the two textures used to render the images in (e) and (f): the first one tends to a pink hue, with darker tones modified prior to lighter ones. The second texture decreases alpha and saturation, keeping only light gray shadows in the background.

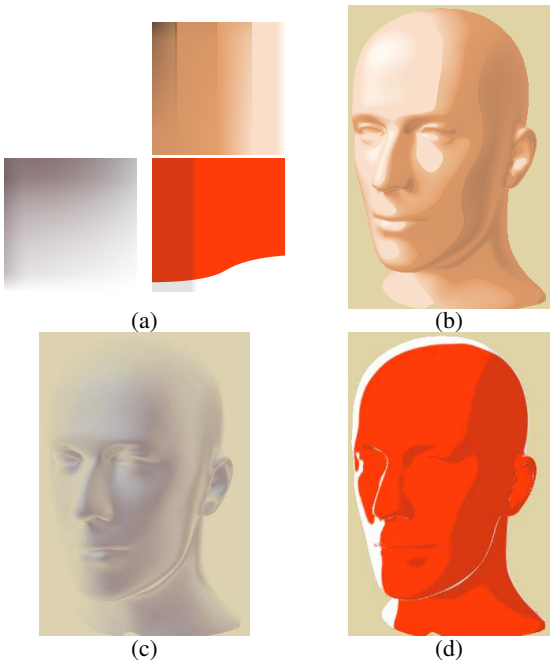


Figure 10: Near-silhouette abstraction and backlighting: (a) shows the textures used in (b), (c) and (d). In (b), we use a texture where intermediate tones are blurred prior to dark or light ones; this has the effect of smoothing out the shading in near-silhouette regions so that hard boundaries are only visible in regions facing the camera. In (c), the same approach is applied, this time to decrease opacity of the shading except for dark tones; this gives a fuzzy rendering of the model. In (d), we apply a simple white backlight that grows thinnner in darker tones; we can thus control the thickness of the backlighting by moving the light.

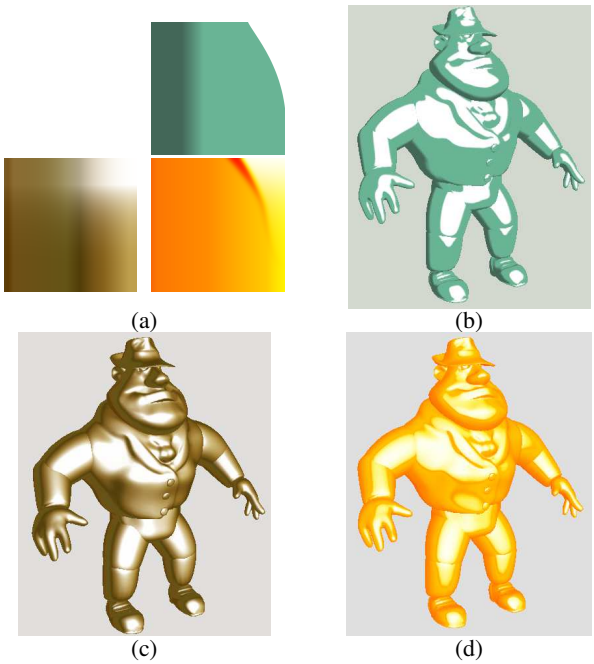


Figure 11: Plastic and metal highlights: (a) shows the textures used in (b), (c) and (d). In (b), we use a highlight texture that decreases in width, but is static with respect to color (white) and profile (hard boundaries); this gives a plastic toon highlight that varies in size depending on the viewing configuration. In (c), we use a texture that creates a “glowing” highlight, resembling a metallic material; this highlight is only present when the view direction is close to the reflected light direction, making the highlight appear suddenly, like a flare. In (d), we modify the color of the highlight, making it redder near its boundaries when it reaches a given scale.

Stroke pattern analysis and synthesis

paper 1072

Abstract

We present a synthesis technique that can automatically generate stroke patterns based on a user-specified reference pattern. Our method is an extension of texture synthesis techniques to vector-based patterns. Such an extension requires (a) an analysis of the pattern properties to extract meaningful pattern elements (defined as clusters of strokes) and (b) a synthesis algorithm based on similarities in the detected stroke clusters.
Our method is based on results from human vision research concerning perceptual organization. The resulting synthesized patterns effectively reproduce the properties of the input patterns, and can be used to fill both 1D paths and 2D regions.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Color, shading, shadowing, and texture I.3.4 [Computer Graphics]: Paint systems

1. Introduction

A particularly important class of non-photorealistic renderings is that of stroke-based images. Various styles such as etchings, pen-and-ink and oil painting renderings can be thought of as stroke-based styles as described in Hertzmann's survey [Her03]. The rendered strokes can be either used to fill in 2D regions, as in painterly rendering, or to annotate 1D paths, like with some hatching patterns; in both cases, the generation of appropriate stroke arrangements for these styles remains a difficult or tedious process to date. Since the individual style of each artist has to be conserved but is not easy to translate in an algorithmic representation, we can not simply rely on procedural methods to generate stroke patterns. Finding a compromise between automation and expressiveness is then crucial for such renderings to be used by artists.

Synthesis by example appears to be the best way to address this question. However, pixel-based texture synthesis is not well suited to stroke patterns, in part because each element of a stroke pattern is individually perceptible, in contrast to pixels. Organized stroke clusters such as those found in hatchings are difficult to extract and reproduce at the pixel level. Moreover, some variation in the reproduced pattern is desirable to avoid too much regularity, and it would be difficult to achieve such variation with pixel-based texture synthesis.

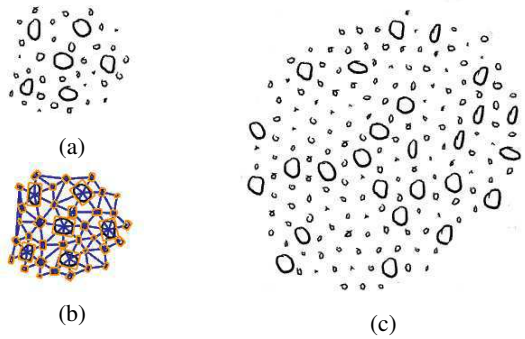


Figure 1: *Our method (a) takes as input a reference vectorized stroke pattern, then (b) analyses it to extract relevant stroke pattern elements and properties in order to (c) synthesize a similar pattern.*

We therefore propose to use a vector-based description of an input stroke pattern supplied by the user. This allows for greater expressiveness and higher-level analysis than would be afforded by a per-pixel approach. The stroke geometry is represented explicitly as connected vertices with attributes such as width and color. While this vector representation is typically less efficient to render, it has the important advantage that strokes can be controlled procedurally to adapt to changes in the depicted regions. Strokes can vary in opacity, thickness and density to depict an underlying tone, as in the WYSIWYG NPR system [KMM*02].

We target any kind of stroke patterns (stippling, hatching, brush strokes, small figures) with a quasi-uniform distribution of positions in 1D and 2D (i.e., along a path or inside a region). The strokes attributes can vary in non-uniform ways and the only parameter required from the user is the scale of the meaningful elements of the pattern. Then, in a manner analogous to texture synthesis techniques, we organise our method in two stages (see Figure 1). An analysis stage where we identify the relevant elements in terms of stroke patterns and their distribution, and a synthesis stage where these elements are placed in the image so as to reproduce an appearance similar to the reference pattern.

In the following we first review the previous work related to our goal, then explain the two stages of our method: analysis and synthesis, and last present our results and discuss future work.

1.1. Previous work

Related work can be found in two different research fields: stroke-based rendering methods that aim at reproducing various artistic styles, and texture synthesis methods that aim at generating a texture from a given sample pattern.

1.1.1. Stroke synthesis

Stroke pattern synthesis systems have been studied in the past, for example to generate stipple drawings [DHvOS00], pen and ink representations [SABS94, WS94], engravings [Ost99], and painterly rendering [Her98]. However, they have relied primarily on generative rules, either chosen by the authors or borrowed from traditional drawing techniques. We are more interested in analysing reference patterns drawn by the user and synthesizing new ones with similar perceptual properties in order to give the user as much freedom as possible in the choice of his own style.

Kalnins *et al.* [KMM*02] described an algorithm for synthesizing stroke "offsets" (deviations from an underlying smooth path) to generate new strokes with a similar appearance to those in a given example set. Hertzmann *et al.* [HOCS02], as well as Freeman *et al.* [FTP03] address a similar problem. Neither method reproduces the interrelation of strokes within a pattern. Jodoin *et al.* [JEGPO02] focus on synthesizing hatching strokes, which is a relatively simple case in which strokes are arranged in a linear order along a path. The more general problem of reproducing organized patterns of strokes has been adressed by the authors in the case of hatching and stippling using a statistical approach [Ano06]. (See the submitted version in supplemental material.) No neighborhood comparison was taken into account and the class of possible elements was reduced to single-colored points or lines. Here we target a wider range of patterns (including color patterns) and take into account the global organization of the pattern by means of neighborhood comparisons.

1.1.2. Texture synthesis

The idea of synthesizing textures, both for 2D images and 3D surfaces, has been extensively addressed in recent years. Previous work can be organized in two categories: parametric and non-parametric methods.

Parametric methods aim at giving a compact description of textures: they make use of statistical analysis to characterize an input texture by a set of parameters, and then try to synthesize similar textures in order to validate the parametric model. The reader can refer to the paper by Portilla and Simoncelli [PS00] for a good overview of parametric models. Our previous work on stroke pattern synthesis [Ano06] bears similarities to parameteric methods in that it performs a statistical analysis of properties of the input pattern (such as stroke positions, lengths, and orientations). Such methods are hard to extend to general patterns because the parameters depend heavily on the style and structure of the pattern.

On the other hand, non-parametric methods work exclusively from the reference texture [EL99, WL00, WL01, Tur01, Ash01]. Even if this representation is less compact, the synthesis results are usually more convincing and the type of synthesized textures more general. These iterative algorithms work with neighborhood comparisons between the reference and target textures. The size and shape of the neighborhoods vary from one technique to the other. Some methods work in scanline order, while others grow the texture from a central starting point; Synthesis is sometimes hierarchical. Still, one cannot directly apply pixel-based non-parametric texture synthesis to vector-based patterns, because stroke patterns are composed of individually perceived elements. We thus choose to adapt an existing non-parametric method [EL99] to vector data.

1.2. Contributions

To define a stroke pattern we draw upon research in human vision, more specifically in the field of perceptual organisation. Indeed, there is a common agreement that the human visual system, in the early stages of perception, structures 2D information into elements based on a set of criteria such as proximity, parallelism, and continuation [Pal99]. In the case of stroke patterns, this means that some sets of input strokes are perceived as single elements, and at a higher level, the distribution of elements defines the pattern.

Our first contribution is an analysis method that extracts an intermediate-level description of vector data, using perceptual organisation criteria applied to input strokes. In particular, we are able to analyse strokes of different styles: not only stippling and hatching strokes, but also brush strokes and small figures (see Figure 5 and Figure 6).

Our second contribution is a synthesis method analogous to texture synthesis, but operating on vector data. We propose a perceptually-based neighborhood matching algorithm that

allows comparisons between neighborhoods even when they have dissimilar connectivity.

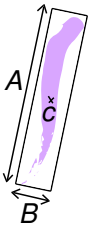
2. Analysis

The first step of our method aims at analyzing a reference stroke pattern to extract the meaningful elements that constitute the pattern, as well as their distribution. We define an element as a cluster of strokes that is perceived as a single feature by the user. Since we target a quasi-uniform distribution of elements, there is a characteristic scale of the pattern that gives the size of such elements. We believe that the choice of scale is context-dependent, and we let the user specify it. Note, however, that the whole analysis process is interactive, hence providing the user enough feedback to easily set a convenient scale. This allows us to target a wide range of patterns as shown in Section 4.

Once the scale is chosen the rest of the process is fully automatic and works as follows. We first fit an element to each input stroke; we then cluster elements iteratively using perceptual organisation criteria; finally, we relate elements to each other to derive properties about their distribution.

2.1. Element fitting

We define an element by its center and its two elongation axis. In order to fit an element to a stroke, the user can either choose to only consider the skeleton of the stroke (i.e. the gesture input by the user) or to also take into account its style (thickness, fading, tapering, etc). In the first case, we fit an element to the points that define the skeleton, while in the second case, we fit an element to the points of its contour.



An element E is constructed by fitting an oriented bounding box to the chosen set of points (skeleton or contour) that will prove useful for approximating geometric measures between elements. We first fit a gaussian distribution to the points to compute the two principal elongation directions given by the eigenvectors A and B of the points' covariance matrix. Each point is then projected onto each axis to compute the size and center c of the bounding box.

2.2. Element clustering

Now that each stroke is represented by its bounding box, we want to cluster them into elements at the chosen scale. For example, a hatch element can be made of several overlapping strokes, and a small figure (think of a flower) is often drawn using a small number of individual strokes.

To decide whether two elements can be clustered, we draw upon the work of Etemadi *et al.* [ESM*91] on perceptual line segment grouping, as we did in our previous work [Ano06], but this time extended to bounding box elements. Two elements are clustered if they meet a proximity constraint (e.g., for a flower), or if they meet a continuation constraint (e.g., for a hatch element). These tests are performed against the user-defined scale ε, that represents the minimum distance at which two elements are perceived separately. When two elements are clustered, we merge their respective strokes and fit a new element using the method explained in the previous section.

Clustering is performed by a greedy algorithm that processes the strokes in the order they have been drawn. The fitted elements are first placed into a queue. At each step, an element E^* is popped and every other element E_i in the queue is tested for clustering based on an element pair comparison. If the test is successful, we merge E_i into E^* and remove E_i from the queue. After all the elements of the queue have been tested, if any clustering occured, E^* is pushed back into the queue. Otherwise, E^* is added to the output list of clustered elements. The algorithm repeats until the queue is left empty.

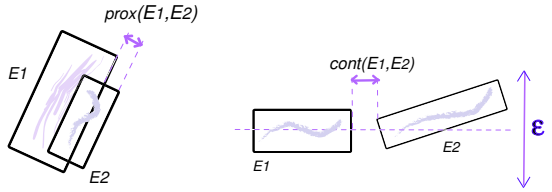


Figure 2: *Element clustering uses two perceptual measures: (a) Proximity and (b) Continuation.*

2.2.1. Proximity measure

The proximity between two elements E_1 and E_2 is computed using Hausdorff distances so that nested, or very close objects are clustered together (see Figure 2(a)):

$$prox(E_1,E_2) = \min(d_H(E_1,E_2),d_H(E_2,E_1))$$
$$d_H(E_1,E_2) = \max_{q_1\in E_1}\left(\min_{q_2\in E_2}(d(q_1,q_2))\right)$$

where $d_H(E_1,E_2)$ is the directed Hausdorff distance. In practice, it is computed using point-line distances between the bounding boxes. If $prox(E_1,E_2) < \varepsilon$, then E_1 and E_2 are clustered.

2.2.2. Continuation measure

Continuation has to be checked on all pairs of axes between the two elements E_1 and E_2. For each of the four configurations, we first have to ensure that the elements are near-collinear; then we compute a continuation measure. Without loss of generality, we only consider the measures of E_2 relative to the axis A_1 of E_1. E_2 is near-collinear to E_1 iff:

$$\forall p \in E_2, d(p,A_1) < \varepsilon/2$$

If the above condition is met, then a continuation error is computed as the gap between the projected points of E_1 and E_2 on A_1 (see Figure 2(b)):

$$cont(E_1,E_2) = \min_{p\in E_1^*, q\in E_2^*}(d(p,q))$$

where E_i^* is the set of points of E_i projected on A_1, $i = 1,2$. In practice, we also use the bounding boxes to speed up this computation. If $cont(E_1,E_2) < \varepsilon$ for any of the four configurations, then E_1 and E_2 are clustered.

2.3. Element distribution

Having identified the elements of our reference pattern, we can now extract connectivity information among them in order to characterize their distribution (see Figure 1(b)). We use the center position of each element. For 1D patterns, we extract the neighbors along a chain, while for 2D patterns, we extract a Delaunay triangulation and keep only edges that are part of at least one unskewed triangle (i.e. a triangle that do not has an angle greater than $\frac{2\pi}{3}$). The pattern input by the user is supposed to be uniform, but in practice, the distribution of elements is only close to uniform. We measure locally this variation by computing a shift vector S_{ref} that expresses the displacement between an element's position and the barycenter of its neighbors' positions. We will use those measurements to add variation to a synthesized pattern in Section 3.3.

3. Synthesis

Thus far, we analysed the reference stroke pattern in order to get a higher-level, perceptually meaningful description of it. In this section, we show how to take advantage of this knowledge during synthesis. Since we want to address the synthesis of texture-like patterns, it makes sense to take inspiration from the texture synthesis litterature. In our approach, we draw comparisons with Efros and Leung's pioneering paper [EL99]: like them, we use a causal synthesis procedure that starts with an element at the center of the pattern and expands it outward using neighborhood comparisons on the previously synthesized elements.

Our method exhibits some important differences though: contrary to the distribution of pixels on a grid, our element positions are not supposed to be aligned. This has an impact on the neighborhood comparison procedure that has to match relevant neighbors between the reference and target patterns. Moreover, elements are easily identifiable and perceived in isolation, thus their comparison should consider the whole set of their parameters: orientation, length, width and color. To do that, we draw inspiration from the field of perceptual organisation once again and show how a trade-off between variation and fidelity can be obtained. Algorithm 1 summarizes the synthesis process.

Algorithm 1 Stroke pattern synthesis

$D_{tar} \leftarrow$ InitialiseDistribution(D_{ref})
$E^s_{tar} \leftarrow$ GetCenterElement(D_{tar})
for each element E_{tar} in D_{tar} growing outward from E^s_{tar} **do**
  $E^*_{ref} \leftarrow$ FindBestMatch(E_{tar}, D_{tar}, D_{ref})
  $E_{tar} \leftarrow$ SynthesizeElement(E^*_{ref}, D_{tar}, D_{ref})
end for

We first present our neighborhood comparison in Section 3.1, before describing in Section 3.2 how it is applied iteratively to create the target pattern. Finally, in Section 3.3, we explain how the user can add variation to the synthesized pattern while keeping a strong similarity with the reference.

3.1. Synthesizing one element

Let E be an element in the synthesized pattern and assume for the moment that all elements in the pattern except for E are known. Let $\omega(E)$ be a neighborhood around E. To assign properties to E, as in [EL99], a set of neighborhoods Ω similar to $\omega(E)$ is extracted from the reference pattern using various perceptual measures described below. Then one of the neighborhoods in Ω is randomly chosen and the center element of the picked neighborhood is used for E.

Neighborhood comparisons are more complex for stroke pattern synthesis than for texture synthesis for two reasons: the number and position of neighbors vary in our distributions, and elements are more complex entities than pixels. The computation of the similarity between two neighborhoods ω_{ref} and ω_{tar} is thus performed in two steps. First, relevant elements of this pair of neighborhoods are found. Second, a set of perceptual organisation measures is tested against perceptual similarity thresholds for the whole candidate reference neighborhood.

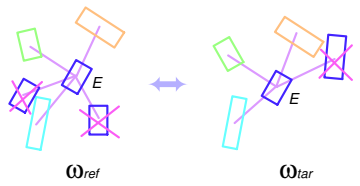


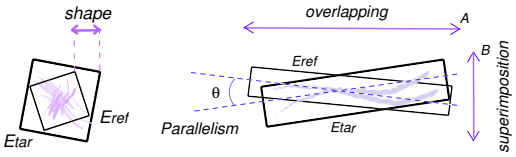
Figure 3: *Neighborhood comparison: Pairs of relevant elements are extracted based on their position.*

For the determination of relevant elements, we only consider pairs of closest reference and target elements by comparing their positions, see Figure 3. This heuristic locally matches the distribution of element positions between the reference and target patterns. We keep pairs of elements $E_{ref} \in \omega_{ref}$

and $E_{tar} \in \omega_{tar}$ such that

$$E_{tar} = \arg\min_{E \in \omega_{tar}} (d(E_{ref}, E))$$
$$E_{ref} = \arg\min_{E \in \omega_{ref}} (d(E_{tar}, E))$$

where $d(E_1, E_2)$ is the euclidean distance between the centers of E_1 and E_2.



(a) (b)

Figure 4: *Element neighborhood matching uses various perceptual measures: (a) isotropic elements; (b) anisotropic (elongated) elements.*

Once the elements are matched, we compute a set of four different perceptual measures that can be organized in two categories (see Figure 4): a *shape-matching* measure that compares two elements as point sets by computing a symmetric Hausdorff distance; and a set of three measures - *parallelism, overlapping and superimposition* - that compares elements using the higher-level description extracted during the analysis. The shape-matching measure is useful when there is an ambiguity between the principal and secondary axes of elongation of an element (e.g., a circle).

Each measure is computed independently on each pair of elements in their respective frame; i.e., the element centers are first aligned before the following measures take place:

$$shape(E_{ref}, E_{tar}) = \max(d_H(E_{ref}, E_{tar}), d_H(E_{tar}, E_{ref}))$$
$$par(E_{ref}, E_{tar}) = |\theta(A_{ref}, A_{tar})|$$
$$ov(E_{ref}, E_{tar}) = \max(\frac{|A_{ref}|}{|A_{tar}|}, \frac{|A_{tar}|}{|A_{ref}|})$$
$$sup(E_{ref}, E_{tar}) = ||B_{ref}| - |B_{tar}||$$

For the computation of the shape measure, we approximate the directed Hausdorff distance d_H using bounding boxes as previously. The parallelism measure is simply taken to be the norm of the angle θ between the two elements' principal axes. Overlapping is the maximum ratio of lengths between the target and reference principal axes. And superimposition is the difference in thickness (length of the secondary axis) between the target and the reference.

In addition to these geometric measures, any attribute can also be taken into account during the synthesis. We illustrate this ability with a simple color distance:

$$col(E_{ref}, E_{tar}) = d_{RGB}(C_{ref}, C_{tar})$$

where C_{ref} and C_{tar} are the colors of E_{ref} and E_{tar} in RGB.

All the measures are then averaged over the element pairs of ω_{ref} and ω_{tar} to give a set of perceptual measures between neighborhoods. They are then tested against a set of perceptual thresholds. These thresholds control the amount of selected candidate neighborhoods: they have to be sufficiently large to provide enough candidates, but small enough to avoid incoherences. In our experiments, we use $\sigma_{shape} = 0.1L$ where L is the average length of the reference elements, $\sigma_{par} = \frac{\pi}{20}$, $\sigma_{ov} = 1.5$, $\sigma_{sup} = 0.1L$ and $\sigma_{col} = 0.15$. We observed that our algorithm is robust to small variations in these thresholds. Two neighborhoods are then considered similar relative to a given measure m iff $m(\omega_{ref}, \omega_{tar}) < \sigma_m$.

The measures are finally combined to determine the similarity of the candidate reference neighborhood to the target one. If the colors or any other attribute does not match, we simply discard the matching. Otherwise, we test whether the neighborhoods match by considering them as sets of points or sets of elements. In our approach, we thus use the following combination

$$col \textbf{ and } (shape \textbf{ or } (par \textbf{ and } ov \textbf{ and } sup))$$

In Section 4, we show and comment examples that exhibit the role of each measure: hatching strokes are more discriminated by parallelism, overlapping or superimposition, while small figures rely mainly on the shape measure; color is independent of the shape of elements, but further refines the above measures.

3.2. Synthesizing a pattern

As mentioned previously, our synthesis algorithm begins at the center of a uniform distribution similar to that of the reference one. Hence, we first build a distribution of element positions that we call seeds, and connect seeds together to get neighborhood relationships. To this end, as in our previous method [Ano06], we use Lloyd's method [Llo82] in 1D and in 2D: it is an iterative algorithm that starts with a random distribution of seeds. Then, at each step, a Voronoi diagram of the seeds is computed, and each seed is moved to the center of its Voronoi region. It converges to a centroidal Voronoi tesselation, close to regular. The only parameter of the method is the number of seeds, that we set to $N_{tar} = N_{ref}.\mathcal{A}_{tar}/\mathcal{A}_{ref}$, where $\mathcal{A}_{ref}$ and $\mathcal{A}_{tar}$ are the areas of the reference and target regions respectively. Finally, when the algorithm has converged, for 1D patterns, we extract the neighbors along a chain, while for 2D patterns we extract a Delaunay triangulation and keep only the edges which are part of an unskewed triangle, in order to avoid degenerate edges at the border of the triangulation.

In the previous section, we have discussed a method of synthesizing an element when its neighborhood elements are already known. Unfortunately, this method cannot be used directly for synthesizing the entire pattern since for any element, only some of its neighbors will be known during the propagation. Like in Efros and Leung's approach, the

element synthesis algorithm must be modified to handle unknown neighborhood elements. This can be easily done by only matching on the known values of $\omega(E)$ and normalizing the error by the total number of known elements. This heuristic appears to provide good results in practice.

3.3. Adding variation

The patterns synthesized with our method exhibits strong similarities with the reference, since they consist of elements that have been copied from it. One might also wish to introduce some amount of variation relative to the reference pattern. In order to add such a variation, we developed a postprocessing mechanism that slightly changes the parameters of the synthesized elements and is controllable by the user via a slider. For each element, and for each of its parameters independently, we select a set of similar values in the reference pattern. E.g., we select a set of orientations close to the synthesized element's orientation. Then, we pick one value from this set and use it in place of the parameters of the synthesized element. This mechanism lets us exchange parameters between similar reference elements without producing elements that are too different from those of the reference pattern.

Another noticeable difference between our synthesized patterns and their reference is the distribution of positions: while the distribution of a synthesized pattern can be considered uniform, this is not the case of the pattern input by the user. The variation present in the input might be desired by the user, and we thus propose a heuristic to reintroduce variation in the distribution of element positions as a postprocess. For each synthesized element E_{tar} that has $n \geq k$ neighbors, we get the reference shift vector S_{ref} of the corresponding E_{ref}, computed during the analysis (see Section 2.3); Then we position all the E_{tar} in parallel at the barycenter of their neighbors, and translate them by $S_{tar} = (S_{ref}\mathcal{A}_{E_{tar}})/(n\mathcal{A}_{E_{ref}})$ where $\mathcal{A}_{E_{ref}}$ and $\mathcal{A}_{E_{tar}}$ are the areas of the neighborhoods of E_{ref} an E_{tar} respectivelly. In practice, we use $k = 2$ for 1D patterns and $k = 4$ for 2D patterns.

4. Results

We show here some results of our synthesis method using various types of elements in 1D and 2D. Computation times are of the order of a second for 1D patterns; and between 5 and 10 seconds for 2D patterns, depending on the neighborhood size.

Figure 5 shows 1D synthesis. A simple example is shown in Figure 5(a) where curved hatching strokes are drawn with sketchy gestures and are properly analysed and synthesized. Our postprocess that adds variation to both position and element parameters is illustrated in Figure 5(b) with a simple hatching pattern; Notice how the vertical positions of elements are reintroduced in the synthesized pattern. We then show how we can reproduce smooth variations in the parameters of the elements along the 1D path. In Figure 5(c), we use a 5-ring neighborhood synthesis to reproduce the smooth change in the orientation of elements along the path; Here the parallelism measure plays a major role in the synthesis. Figure 5(d) shows the influence of the neighborhood size on the quality of the result. Synthesised patterns with 1-ring, 3-ring and 5-ring neighborhoods are shown from top to bottom: the smooth change in stroke length is only well captured with the 5-ring neighborhood. Here, the overlapping measure is discriminant. Figure 5(d) shows a smooth change in element thickness captured with a 5-ring neighborhood, and made possible by our superimposition measure. Finally, we show at the bottom an example using brush strokes of alternating colors: the synthesized pattern exhibits the same alternance, thanks to our color measure.

Figure 6 shows 2D synthesis. Figure 6(a) shows an example of bars oriented in multiple directions. Here, our synthesis method is able to reproduce the complex relations among similar elements. It can also synthesize elements that are less similar, as in Figure 6(b) with water drops, small figures like the flowers of Figure 6(c) or alternating two different kinds of elements as in Figure 6(d). Finally, the variation of element positions is illustrated in Figure 6(e), where the distribution obtained with Lloyd's method is modified to be more similar to the input pattern.

5. Discussion and future work

5.1. Analysis

Our analysis method can extract a wide range of elements (stipples, hatches, brush strokes, small figures), but our element representation (a center and two axes) is too simple to correctly extract long curved strokes. Moreover, we do not target structured patterns, such as a brick wall, where the overall organization should be extracted along with each element. We thus plan to address these two issues in the future by modifying our element model and adding multiple levels of analysis to be able to capture more structured patterns. Another issue is the use of additional perceptual criteria such as closure or junctions in order to perform a deeper interpretation of the input pattern.

In our approach, the analysis stage is user-assisted in order to determine the scale of the pattern. Since the clustering of elements is dependent on this scale and is implemented with a greedy algorithm, it can produce flickering on rare occasions when the user interactively modifies the scale. However, this has no impact on the final synthesis result. On the other hand, if one wants to consider scanned drawings as input, it would make sense to extract both the elements and the scale automatically. The greedy nature of the clustering algorithm might then lead to problematic behaviors.

Figure 5: *1D synthesis results. (a) A simple hatching example that uses sketchy strokes; (b) another hatching example with a uniform distribution of elements (on top), and the same pattern after variation have been added (at bottom); (c) a smooth change of element orientation is analysed and synthesized with a 5-ring neighborhood; (d) a smooth change of element length is analysed and synthesized with increasing neighborhood sizes (from top to bottom: 1-ring, 3-ring and 5-ring neighborhoods); (e) a smooth change of element thickness is analyzed and synthesized with a 5-ring neighborhood; (f) the alternance of strokes colors is captured and reproduced in the synthesized pattern.*

5.2. Synthesis

Our synthesis method currently generates quasi-uniform distributions of elements via the Lloyd algorithm. In the future we will target non-uniform distributions that can take into account density variations within the pattern. We also plan to take into account the whole element shape rather than only its center position in the distribution definition. The heuristic we used for finding relevant neighbors also suffers from a limitation: it can happen that no pairing is found between the reference and target neighborhoods. However, in practice, the even distribution produced by Lloyd's method prevents this worst case scenario from happening.

Another interesting point is the ability to take into account attributes of the input strokes during synthesis. We only investigated color, but other attributes such as thickness, opacity or texture might give convincing results. Finally, our variation technique is tailored to stroke pattern synthesis and has thus no equivalent in texture synthesis. We plan to extend this ability to take into account the shape of the strokes.

5.3. Extension to synthesis on surfaces

Our system works in the picture plane and we plan to extend it to synthesis on surfaces in the future. However, stroke-based rendering raises several specific questions in terms of rendering. Indeed, the strokes need to be of roughly constant size in 2D if one wants to maintain the same style for every viewpoint. Therefore an LOD mechanism has to be defined, and we believe that this can be achieved with a dedicated synthesis algorithm. The automatic generation of mipmaps or Tonal Art Maps [PHWF01] would be a simple way to address synthesis on surfaces. However, we believe that direct synthesis on surfaces will open more interesting avenues: for example, the rendering could be done by varying the attributes of each synthesized stroke depending on the viewing and lighting conditions. By keeping the analogy with texture synthesis, it should be possible to devise such a method in the same way we did in the present paper.

6. Conclusions

We presented a novel approach to the analysis and synthesis of vector-based stroke patterns along a 1D path or inside a 2D region. Our method makes use of perceptual organisation findings to interpret a reference input pattern, and later synthesize a similar one.

The synthesis algorithm is directly inspired by texture synthesis algorithms. It makes use of neighborhood comparisons and is able to take into account additional attributes of the input strokes like color.

The synthesized patterns are very similar to the reference ones, and the user can also add variation to the results while keeping a reasonable fidelity to the reference pattern.

References

[Ano06] ANONYMOUS: Interactive hatching and stippling by example. In *submitted to GI* (2006). 2, 3, 5

[Ash01] ASHIKHMIN M.: Synthesizing natural textures. In *SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics* (New York, NY, USA, 2001), ACM Press, pp. 217–226. 2

[DHvOS00] DEUSSEN O., HILLER S., VAN OVERVELD C., STROTHOTTE T.: Floating points: A method for computing stipple drawings. *Computer Graphics Forum 19*, 3 (2000). 2

[EL99] EFROS A. A., LEUNG T. K.: Texture synthesis by non-parametric sampling. In *IEEE Int. Conf. on Computer Vision* (1999), pp. 1033–1038. 2, 4

[ESM*91] ETEMADI A., SCHMIDT J., MATAS G., ILLINGWORTH J., KITTLER J.: Low-level grouping of straight line segments. In *British Machine Vision Conf.* (1991), pp. 119–126. 3

[FTP03] FREEMAN W. T., TENENBAUM J. B., PASZTOR E. C.: Learning style translation for the lines of a drawing. *ACM Trans. Graph. 22*, 1 (2003), 33–46. 2

[Her98] HERTZMANN A.: Painterly rendering with curved brush strokes of multiple sizes. In *Proceedings of SIGGRAPH 98* (1998), pp. 453–460. 2

[Her03] HERTZMANN A.: A survey of stroke-based rendering. *IEEE Computer Graphics and Applications 23*, 4 (July/August 2003), 70–81. Special Issue on Non-Photorealistic Rendering. 1

[HOCS02] HERTZMANN A., OLIVER N., CURLESS B., SEITZ S. M.: Curve analogies. In *Rendering Techniques 2002: 13th Eurographics Workshop on Rendering* (June 2002), pp. 233–246. 2

[JEGPO02] JODOIN P.-M., EPSTEIN E., GRANGER-PICHÉ M., OSTROMOUKHOV V.: Hatching by example: a statistical approach. In *Proceedings of NPAR 2002* (2002), pp. 29–36. 2

[KMM*02] KALNINS R., MARKOSIAN L., MEIER B., KOWALSKI M., LEE J., DAVIDSON P., WEBB M., HUGHES J., FINKELSTEIN A.: WYSIWYG NPR: Drawing strokes directly on 3D models. *ACM Trans. on Graphics 21* (July 2002), 755–762. 1, 2

[Llo82] LLOYD S. P.: Least squares quantization in pcm. *IEEE Trans. on Information Theory 28*, 2 (1982), 129–137. 5

[Ost99] OSTROMOUKHOV V.: Digital facial engraving. In *Proceedings of SIGGRAPH 99* (1999), pp. 417–424. 2

[Pal99] PALMER S.: *Vision Science : Photons to Phenomenology.* MIT Press, 1999. 2

[PHWF01] PRAUN E., HOPPE H., WEBB M., FINKELSTEIN A.: Real-time hatching. In *Proceedings of ACM SIGGRAPH 2001* (2001), pp. 579–584. 8

[PS00] PORTILLA J., SIMONCELLI E. P.: A parametric texture model based on joint statistics of complex wavelet coefficients. *Int. J. Comp. Vision 40*, 1 (2000), 49–70. 2

[SABS94] SALISBURY M. P., ANDERSON S. E., BARZEL R., SALESIN D. H.: Interactive pen-and-ink illustration. In *Proceedings of SIGGRAPH 94* (1994), pp. 101–108. 2

[Tur01] TURK G.: Texture synthesis on surfaces. In *Proceedings of ACM SIGGRAPH 2001* (2001), pp. 347–354. 2

[WL00] WEI L.-Y., LEVOY M.: Fast texture synthesis using tree-structured vector quantization. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2000), ACM Press/Addison-Wesley Publishing Co., pp. 479–488. 2

[WL01] WEI L.-Y., LEVOY M.: Texture synthesis over arbitrary manifold surfaces. In *Proceedings of ACM SIGGRAPH 2001* (2001), pp. 355–360. 2

[WS94] WINKENBACH G., SALESIN D. H.: Computer-generated pen-and-ink illustration. In *Proceedings of SIGGRAPH 94* (1994), pp. 91–100. 2

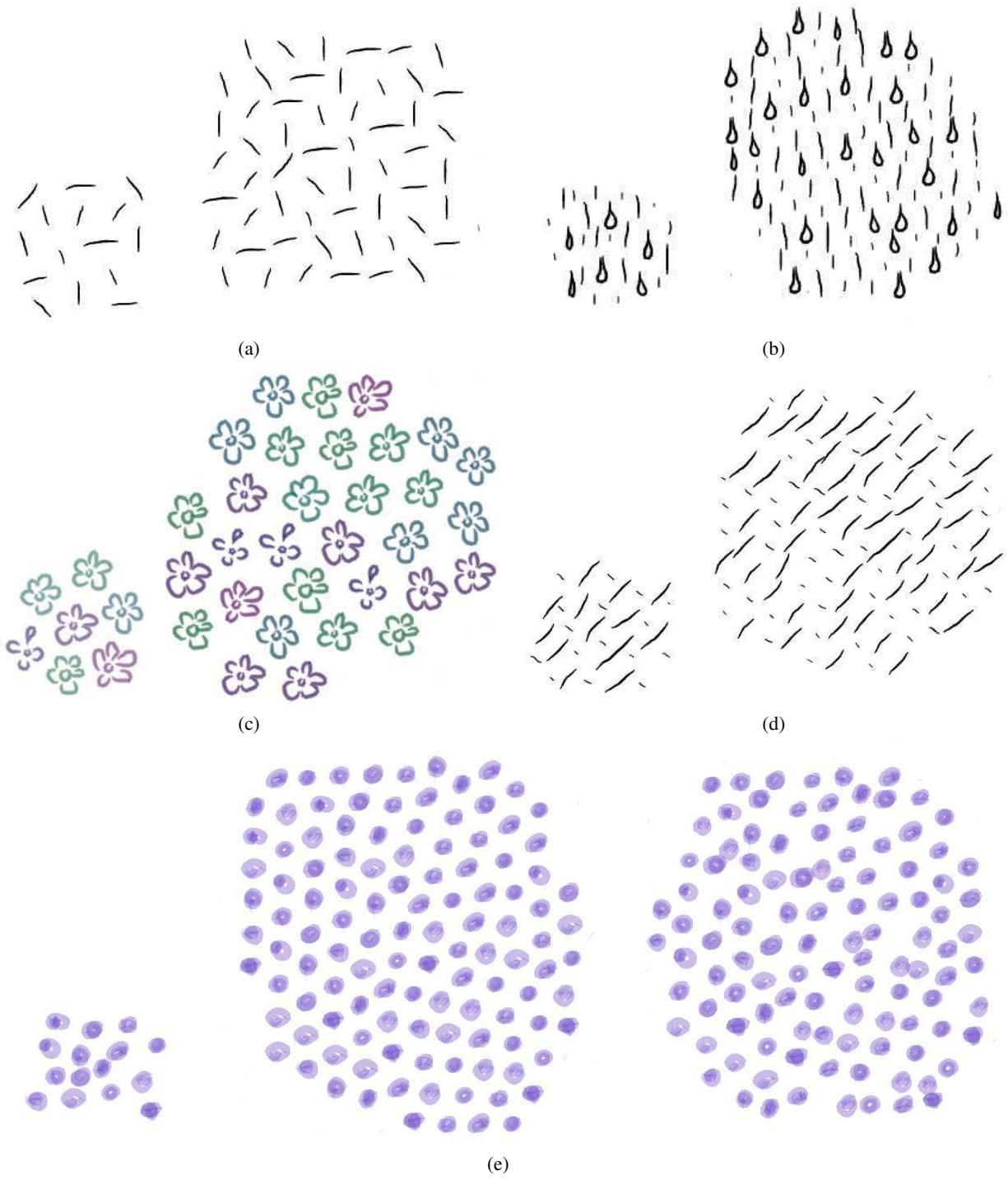


Figure 6: *2D synthesis results. (a) A pattern of hatching strokes in multiple directions is synthesized using a 3-ring neighborhood; (b) elements of various nature (water drops, hatches) are analysed and synthesized with a 3-ring neighborhood; (c) small figures like flowers of different colors can be analyzed and synthesized by our method using a 2-ring neighborhood; (d) a pattern composed of two different types of hatching strokes is synthesized with a 2-ring neighborhood; (e) the addition of variation in the position of elements is able to break the uniform distribution of Lloyd's method.*

Eurographics Symposium on Rendering (2007)
Jan Kautz and Sumanta Pattanaik (Editors)

Dynamic point distribution for stroke-based rendering

David Vanderhaeghe, Pascal Barla, Joëlle Thollot and François X. Sillion

ARTIS - INRIA Grenoble University

Abstract

We present a new point distribution algorithm that is well adapted to stroke-based rendering systems. Its main characteristic is to deal efficiently with three conflicting constraints: the distribution of points should retain a good repartition in 2D; their motion should tightly follow the target motion in the underlying scene; and as few points as possible should be added or deleted from frame to frame. We show that previous methods fail to meet at least one of these constraints in the general case, as opposed to our approach that is independent of scene complexity and motion. As a result, our algorithm is able to take 3D scenes as well as videos as input and create non-uniform distributions with good temporal coherence and density properties. To illustrate it, we show applications in four different styles: stippling, pointillism, hatching and painterly.

1. Introduction

Creating animations by hand requires the artist to paint or draw each frame independently. In complex styles such as painterly, watercolor, airbrush, stippling or hatching, this process does not take into account frame-to-frame coherence of "brush strokes". This can lead to vibrating, evocative, and rich visual experiences when the technique is mastered (see the work of George Schwizgebel, Alexander Petrov, Frederic Back or Bill Plimton for instance). But this noise may also become annoying for the viewer, and artists may want to avoid it while keeping the richness of a complex style. However, ensuring frame-to-frame coherence is a very tedious and difficult task to achieve by hand.

One of the goals of non-photorealistic rendering is precisely to assist the user in complicated and tedious tasks, and automate most of the non-creative process. It then found a natural application in the creation of animations in various styles. Among these, stroke-based rendering [Her03] has received much attention, since it consists in assisting the artist in creating animations where images are composed of many small 2D drawing primitives; a task that is typically time-consuming when done by hand. Previous techniques rendered animations in specific styles, either from an input animated 3D scene or from a video. In this paper, as we consider both types of input, we will refer indiscriminately to a 3D scene or a video as the *underlying scene*.

By considering only small drawing primitives, a natural approach, taken by most of previous work, is to attach them to anchor points in the underlying scene. Hence, when put in motion, these anchor points will guide the primitives in the picture and give a sense of movement. While the definition of a primitive might vary depending on the chosen style, defining the anchor point distribution from frame-to-frame is common and essential to any stroke-based system. It also raises multiple, conflicting constraints: the distribution of anchor points should retain a good repartition in 2D; their motion should tightly follow the target motion in the underlying scene; and as few points as possible should be added or deleted from frame to frame, a property often referred to as *temporal coherence*.

Naive distributions cannot satisfy all of the above requirements. For instance, if one distributes new drawing primitives at each frame, the first constraint might be easily satisfied, but since no motion is matched, and no coherence is ensured, the remaining two constraints are violated. Another approach would be to use a single fixed distribution for the whole animation (good repartition and of course perfect coherence), but it will obviously violate the motion constraint. More sophisticated approaches have been proposed in the literature, but as explained in Section 2, they fail to satisfy all the constraints in general cases. This motivates the development of a more flexible distribution method for the creation of stroke-based animations.

© The Eurographics Association 2007.

Figure 1: *Our algorithm allows to distribute points on arbitrary scenes with non-uniform density and temporal coherence.* ***(a)*** *A rigid 3D model.* ***(b)*** *a 3D scene composed of different objects.* ***(c)*** *A deformable model.*

In this paper, we present a new distribution technique that works for any kind of input, and achieves interactive frame-rates *independently* of the scene complexity. Regarding the afore-mentioned constraints, our algorithm, presented in Section 3, provides the following contributions:

- Generation of non-uniform Poisson-disk distributions with blue-noise properties, necessary for applications such as stippling or pointillism;
- Matching of any motion (rigid- or soft-body deformations, vector flows, occlusions), making the approach tractable for arbitrary 3D scenes and videos;
- Control of temporal coherence via the management of coarse-to-fine transitions when anchor points need to be added or deleted.

To achieve these goals, we sample stroke positions in the image plane and project them to the underlying scene to apply the scene motion. At each frame, previous samples points are reprojected to the image plane and updated with respect to an importance function. New points are inserted if needed. Our main results, presented in Section 4, show the advantages of our distribution over previous approaches to primitive distribution. However, for the purpose of illustration, we also show in Section 5 how our distribution algorithm can be used to render animations in various, albeit simple, styles.

2. Previous work

As our goal here is to compare with other distribution techniques, we first present classic methods employed in the sampling literature, before reviewing previous approaches encountered in stroke-based rendering.

Sampling techniques usually rely on an importance map to create non-uniform point distributions (with more points distributed in areas of higher importance). When applied to time-varying importance maps, either for video environment map sampling [HSK*05] or for primitive distribution [SHS02], the samples follow the motion of the peaks of the time-varying importance. This approach is well-suited for environment map sampling for instance, but in the case of primitive distribution, it results in points floating over the surface as the viewpoint or importance change.

In contrast, we are interested in having points follow a given motion, while their distribution matches a distinct importance map: for instance in stippling, points should follow object motion, with more points in dark regions of the image. The recent tile-based distribution method of Kopf et al. [KCODL06] shows such a behavior: they propose a real-time blue-noise point distribution technique that matches very accurately any importance map. However, because their method relies on pre-computed tileable point distributions, it can only directly deal with 2D rigid motions (panning and zooming inside an image). Adapting the method to more general motions, for instance via non-uniform warping, is not trivial. In this paper, we thus take an alternative approach that does not rely on any precomputation.

Unlike sampling techniques, stroke-based rendering systems precisely focused on ways to convey motion. Different techniques have been proposed, depending on whether a 3D scene or a video is given as input. Because of the amount of strokes we seek to distribute, we only focus here on *automatic* distributions. In comparison, in the work of Kalnins *et al.* [KMM*02] or Daniels [Dan99], the user has to manually place each stroke onto 3D surfaces and provide levels-of-detail for each of them.

For *3D scenes*, anchor points are usually directly distributed onto 3D object surfaces and projected in the picture plane to guide primitives motion. Introduced by Meier [Mei96] for painterly rendering using a static 3D distribution (hence limited in the range of possible viewpoints), the method has been extended by various authors to adapt to viewpoint changes [CRL01, PFS03, NS04, HS04]. They all first construct a hierarchy of 3D anchor points over an object's surface in pre-process. Then they choose a "cut" in the hierarchy at runtime so that the resulting projected points best match the required distribution (e.g. specified via an importance map). While this approach has the advantage of accurately following objects motion with a very good temporal coherence, it suffers from limitations that make it impractical for general scenes. First, the hierarchy construction, built in pre-process, might be time-consuming for large models and some knowledge of "how close" any object can come to the point of view is needed to control its precision. Second, as each object is considered individually, distribu-

© The Eurographics Association 2007.

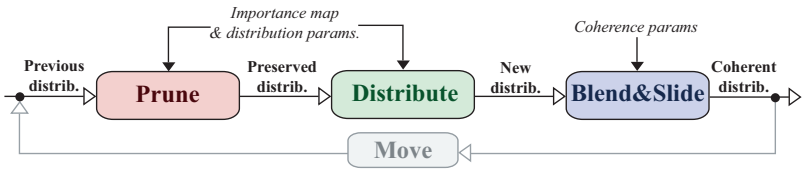


Figure 2: *The different steps of our dynamic distribution algorithm: pruning, distribution, blending&sliding. A coherent non-uniform distribution is dynamically updated from frame to frame regardless of how the scene moved: 3D objects can be deformed, viewpoint motion is not restricted, and motion in video can be provided via its optical flow.*

tions cannot overlap multiple objects (although it might be necessary, for instance with distant objects); they can only follow individual object motion with restricted deformations (no topology change, nor vector flows over surfaces for instance); and they become increasingly time- and memory-consuming with scene complexity. Finally, the repartition constraint cannot be ensured since the points are only guaranteed to appear roughly in the middle of neighbor points when projected to the picture plane [Pas03]. Moreover, distributions obtained this way sometimes present artifacts such as patterns, accumulation or grouping motifs (especially at grazing angles). As a result, 3D-based techniques, while very efficient at ensuring temporal coherence, fail to fully satisfy motion and distribution constraints.

For *videos*, on the other hand, since there are no objects to rely on, no pre-process is needed. Rather, as in the compelling painterly renderer of Hays and Essa [HE04], anchor points follow video optical flow. Provided the optical flow is a good approximation of motion, the resulting anchor point distributions satisfy the motion and temporal coherence constraints. However, this approach needs to be adapted to work with 3D scene input, as it does not take into account occlusions: for instance, if we use Hays and Essa's system on a motion flow extracted from a 3D scene, anchor points may move from one object to the other. Moreover, their method for dealing with distributions assumes a thick, opaque paint medium: multiple uniform distributions are painted in sequence, from coarse to fine. Not only this approach makes the system very slow (around 80 seconds per frame), but it will fail to adapt to other styles such as stippling or pointillism, since there are no continuous variations of anchor points density, and distributions layers are built independently so that two anchor points from different layers may be arbitrary close to each other.

The "animosaics" method of Smith *et al.* [SLK05] provide a distribution approach similar to [HE04], even if used in another context: they take a vector based animation as input and mainly focus on packing mosaics with a uniform distribution. Therefore, regarding our goals, the approach shares the same drawbacks as Hays and Essa's work. Video-based methods hence have important limitations concerning motion and distribution constraints.

To overcome the limitations of previous work we make use of an hybrid approach that combines advantages of 3D- and video-based techniques, while having distribution properties sufficiently close to the ones obtained with sampling techniques, at least in the case of stroke-based rendering.

3. Distribution algorithm

Our goal is to distribute drawing primitives in screen-space in such a way that they follow a given motion in the depicted scene without exhibiting temporal artifacts. To permit non-uniform densities, we also allow the user to specify an importance map I, a function that assigns to any point p in screen-space an importance value $I(p) \in [0,1]$. This function, which typically evolves during the animation, can either be defined by 3D scene properties (object IDs, depth, painted textures, etc) or screen-space properties (luminance, optical flow magnitude, etc).

We choose to ground our dynamic distribution on the static Poisson-disk distribution approach of McCool and Fiume [MF92]. As argued in Section 3.1, it is best adapted to the particular constraints of stroke-based rendering. We then explain in Section 3.2 how we extend this algorithm to deal with dynamic settings. The underlying idea, summarized in Figure 2, is that at each step of our algorithm, points are moved to the next frame according to available scene motion; they are then pruned using a Poisson-disk criterion, and holes in the distribution are filled. The main contribution of this method is that through this process, the updated distribution retains the properties of the static algorithm.

3.1. Static distribution

We first present the guidelines of the Poisson-disk distribution technique of McCool and Fiume [MF92], also called relaxation dart throwing. Standard dart throwing [Coo86] randomly distributes points in the image plane and keeps a point only if no other point is found within a disk of radius r around it. McCool and Fiume extended this algorithm with a hierarchical formulation: initially, points are randomly distributed with a large radius $r = r_{\max}$; once no more space has been found after n trials the radius is multiplied by $\alpha \in (0,1)$. The algorithm stops as soon as $r \leq r_{\min}$. The quality of the distribution can be controlled by the parameters n and α, which allow us to run our algorithm at interactive frame rates for previewing ($n = 100$, $\alpha = 0.9$), or create high quality animations for off-line rendering ($n = 1000$, $\alpha = 0.99$).

© The Eurographics Association 2007.

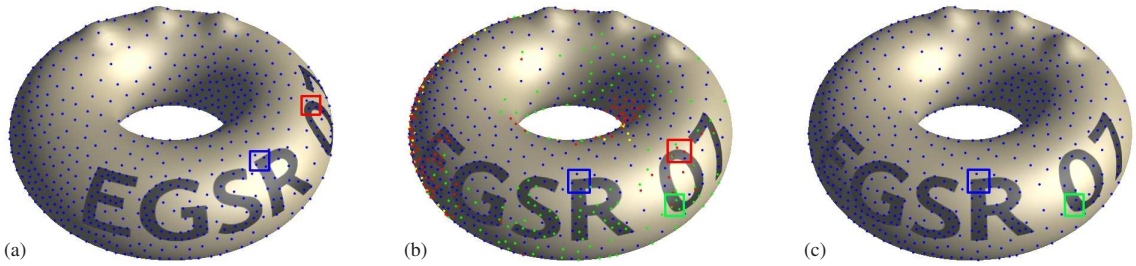


Figure 3: ***(a)*** *Points are initially distributed in screen-space, here using luminance to drive density. They are then projected onto the 3D object.* ***(b)*** *After they have been moved to the next frame, points are reprojected to screen-space. Red points are rejected based on a Poisson-disk criterion; blue points from the previous frame are still valid; green points are added to fill in the blanks; and yellow points are hidden.* ***(c)*** *The distribution after it has been updated. The red, green and blue frames locate corresponding points that have been respectively removed, added or preserved.*

Apart from having good statistical properties (see the survey on Poisson-disk distributions by Lagae and Dutre [LD06]) and being simple to implement, the algorithm is well suited to our problem: it enables non-uniform densities, provides a hierarchical distribution and can be built incrementally. In particular, the point density can be made to follow an arbitrary importance map I defined in screen-space, via the use of an importance criterion: p is kept only if $1-\frac{r(p)-r_{\min}}{r_{\max}-r_{\min}} > I(p)$, with $r(p) \in [r_{\min}, r_{\max}]$ the radius used to distribute it. This way, the user is able to finely control the distribution of points by manipulating $r_{\min}$ and $r_{\max}$.

3.2. Dynamic distribution

Up until now, we explained how points are distributed for a single frame. We start any animation with such a point distribution (Figure 3(a)). Then, we *move* points to the next frame. This produces clusters of points that we clean out by *pruning* unwanted points; but it also creates holes that we fill by *distributing* new points.

Depending on the input given to our system, motion is performed in different ways. To deal with complex 3D scene motions, we project distributed points from screen-space onto object surfaces, move objects (with any kind of motion, including skinning and deformations), and project points back to screen-space. In practice, we take as input triangle meshes, and the projection onto a surface is done by rendering at each frame buffers that hold mesh IDs, face IDs, and triangle barycentric coordinates; then for each point we obtain its corresponding IDs and coordinates simply by reading the buffers. In the next frame, the point is located in 3D and perspectively projected. This approach allows points to stick to their triangle even if they are severely distorted. For videos we simply translate points along an optical flow provided in input (we rely on a classical gradient based method available in Adobe® After Effects®).

Another important aspect of motion is occlusion: while for videos, points are considered hidden only when they fall out of the image dimensions, they can disappear behind other surfaces when dealing with 3D scenes. We perform a simple depth test to detect occlusions, and tag hidden points that will be treated specifically in the next step. Moreover, points falling onto the background in 3D scenes are considered hidden. Figure 3(b) shows points after they have been moved to a subsequent frame, with hidden points in yellow.

In both videos and 3D scenes, points can be either visible or hidden after they have been moved to the next frame. In the following, we only consider points that are still visible. In order to clean out clutters, we re-insert them in the order they have been previously distributed using a Poisson-disk criterion to reject unwanted samples. For each radius r_L corresponding to a level L of the hierarchical distribution, we check *all* the previously inserted points against the importance criterion of Section 3.1. Accepted points are re-inserted with radius r_L and we try to re-insert the remaining points at level $L-1$. The process stops as soon as all the points have been re-inserted or the finest level has been checked, usually rejecting a subset of the previous points. These rejected points are tagged as deleted (red points in Figure 3), while accepted points (in blue) are preserved so that the distribution is coherent from frame to frame.

Finally, holes still remain in the distribution due to appearing surfaces or optical flow dilation. We thus need to distribute new points to fill these holes with the same properties as in the static case of Section 3.1. Considering *all* the re-inserted points as part of the current distribution, we reuse McCool and Fiume's algorithm: each level L of the hierarchical distribution is processed in a top-down approach; points are randomly distributed with a radius r_L, until no more space has been found for n trials, discarding points that do not meet the importance criterion. The algorithm stops after the finest level, corresponding to $r_L \leq r_{\min}$, has been processed. This approach allows inserted points to appear in the highest possible level of the hierarchy, hence resulting in a better repartition.

© The Eurographics Association 2007.

3.3. Temporal coherence

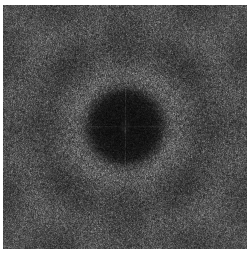
Using the algorithm described in Section 3.2 guarantees that most of the previous points are kept from frame-to-frame. However, we cannot avoid inserting or deleting *some* points, as motion and density are usually conflicting constraints. Hence, in order to further improve the temporal coherence of our approach, we propose two mechanisms: blending and sliding.

Blending: Inserting or deleting points as soon as this is requested by our distribution algorithm often results in disturbing popping artifacts. As in previous work [PFS03, HE04, KCODL06], we rather blend points in or out using smooth transitions. A blending duration is assigned to each inserted and deleted point, that is incremented or decremented at subsequent frames. Therefore, the distribution algorithm only tags points for being inserted or deleted, and leaves the task of performing the operation to the blending mechanism.

The main difference in our approach compared to previous work is that we only focus on blending *duration*, so that the way blending is actually performed can make use of this value, but vary depending on the application (see Section 5). Moreover, we propose to distinguish between two different cases where blending occurs: density and occlusion events. For occlusion and dis-occlusion events, points are deleted (resp. inserted) directly. In practice, detecting which points become occluded in the current frame is easy (using the z-buffer), but finding out what points are dis-occluded requires to store some more information (z-buffer and transformation matrices from the previous frame). For points that appear or disappear by density, we developed another mechanism that assigns blending durations proportional to the radius of the inserted or deleted point. In practice, it produces smooth transitions where "details" are added progressively, from coarse to fine.

Sliding: Another problem may appear when we remove a point due to the Poisson-disk constraint, and it leaves enough place to add another point in its neighborhood: even with blending enabled, it can result in a flickering artifact during animation. We avoid this problem by matching a pair $\{p_d, p_i\}$ of deleted and inserted points, where p_d is the closest deleted points to p_i in screen-space, using the radius of p_i as a threshold. Then instead of removing p_d, we allow it to *slide* linearly towards the position of the newly inserted one. We only perform sliding for density events though, otherwise hidden points would tend to slide across occlusion boundaries.

In practice, we track the original points p_d and p_i and linearly interpolate between their projected positions. This results in a projected distance $d_p(t)$ between the sliding point and its target position, with $t \in [0, t_s]$ where t_s is a sliding duration set by the user. While this would work for simple motions, more complex movements such as rotations can stretch out the projected pair, hence increasing $d_p(t)$; which is opposite to the goal of the sliding mechanism. Therefore, we also clamp $d_p(t)$ to the distance that would be obtained with a straightforward 2D linear interpolation, $d_p(0)(t - t_s)/t_s$. This way, the interpolated position gets progressively closer to the inserted point, while still conveying object motion.



Relative radius statistics		
α/n	RDT	ours
0.9/100	0.69	0.67
0.99/1000	0.80	0.74

Figure 4: ***Left:*** *Power spectrum of our point distribution method obtained by averaging periodograms for each frame of a test animation (central peak removed).* ***Right:*** *Relative radius statistics for static relaxation dart throwing (RDT) and our dynamic point distribution method.*

4. Results

Figure 1 shows stills from dynamic distributions created with our algorithm, either from 3D scenes or video input. The full animations can be viewed in the accompanying video (available at `http://artis.imag.fr/Publications/2007/VBTS07a/`). The point distribution tightly matches required motion and density in all cases, while exhibiting good temporal coherence.

The sampling times are independent of scene complexity, and our method requires no pre-process. The performance of our distribution algorithm thus only depends on image dimensions and the choice of the n and α parameters that drive points density. Note that as in [MF92], we use a uniform grid to speed up distance tests during distribution. For interactive manipulation ($n = 100/\alpha = 0.9$) we get frame rates between 5 and 10 fps, while for off-line rendering ($n = 1000/\alpha = 0.99$), we are able to render point distributions at approximately 1 fps. These parameter settings are the ones advocated by McCool and Fiume [MF92] and in both cases, we render images at 720×540 resolution, and get performances similar to them.

We now present statistical and visual evaluations of our distribution algorithm in comparison with previous work.

4.1. Statistical evaluation

We first want to check that our dynamic distribution has qualities similar to that of McCool and Fiume's original algorithm [MF92]. A study by Lagae and Dutré [LD06] proposed to use two measurements to evaluate Poisson-disk distributions: relative radius that measures the amount of "packing" in the distribution; and power spectrum that gives infor-

© The Eurographics Association 2007.

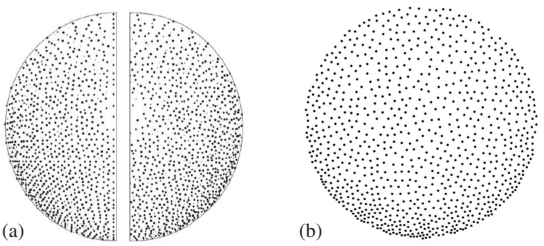


Figure 5: *Comparison with 3D-based distributions.* ***(a)*** *The technique of Pastor et al. [PFS03] (figure taken from their paper) cannot ensure a good repartition (left half), even after randomization (right half).* ***(b)*** *Our approach, in comparison, produces distributions with blue-noise properties.*

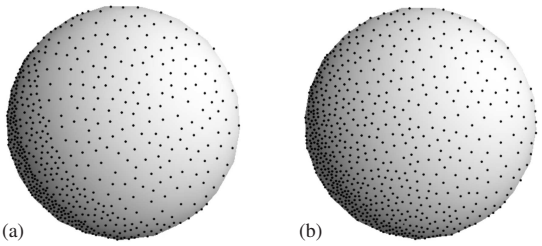


Figure 6: *Non-uniform density.* ***(a)*** *Using* $\alpha = 0.5$ *results in a quantification effect while* ***(b)*** *with* $\alpha = 0.9$ *the importance gradation is faithfully reproduced.*

mation about its frequency content. Regarding these measurements, that are only valid for *uniform* distributions, McCool and Fiume's static distribution method is one of the best available techniques. We found no similar evaluation tool for non-uniform distributions.

We thus measured the power spectrum and relative radii for the frames of a test animation using the torus model of Figure 3, and a dynamic *uniform* distribution that fills the image (using $r_{\min} = 2$, $r_{\max} = 10$, $\alpha = 0.99$ and $n = 1000$). We expected that our approach, by introducing coherence of points distribution, would result in worse relative radii, or reveal regularities in the power spectrum. However, it did not show significant influence (see Figure 4): the spectrum clearly exhibits blue-noise properties (we removed the central peak for clarity as in [LD06]) and relative radii results are only slightly better in the static case. Note that as our distribution is not periodic, horizontal and vertical lines appear in the spectrum.

While these measures do not outperform static Poisson-disk distribution methods, they show that our technique has comparable statistical qualities while it satisfies additional constraints, i.e. scene motion and temporal coherence.

4.2. Visual evaluation

The previous section compared our approach to previous techniques using uniform distribution statistics. However, such comparison is sometimes infeasible, either because no statistics are available, or because we deal with non-uniform distributions. In this section, we thus resort to visual evaluation, to compare our approach with object-based methods and illustrate its behavior with non-uniform densities.

The pioneering work of Meier [Mei96] produced remarkably stable object-based distributions: this is because her distribution is static, and done once in pre-process. A similar image-based approach, obtained using a static 2D distribution, would introduce the so-called "shower-door effect". Thus the object-based approach of Meier gives much better results, albeit for the limited range of viewpoints where the distribution is dense enough, or not too packed.

But as soon as we need to insert or remove points to allow camera and object motions to be unconstrained, we lose stability and need blending. This is a problem common to object- and image-based approaches: in the first case, points are distributed on a surface and selected/deselected depending on a refinement function; while in our case, points are added/deleted using a Poisson-disk criterion. While both approaches have similar problems, object-based techniques have the additional burden of finding a good refinement function. And as shown in Figure 5(a), there is no simple way to get a good 2D distribution starting from a 3D distribution, even after introducing randomization as in Pastor et al. [PFS03]; while our method, shown in Figure 5(b), simply avoids this problem by directly distributing points in the picture plane.

Non-uniform distributions created with our approach depend on the values of $r_{\min}$, $r_{\max}$ and α. While $r_{\min}$ and $r_{\max}$ control the density in regions where the importance map is equal to 1 and 0 respectively, the value of α influences the continuity of the non-uniform distribution. As shown in Figure 6, a low value for α has an effect similar to quantization, while a reasonable value results in smooth gradations. In practice, we found that using $\alpha > 0.9$ gives very good results in most cases.

5. Applications

We now present some applications of our distribution algorithm that work with 3D scenes or video input (with optical flow). Each of them target a different visual style: stippling, pointillism, hatching or painterly. Note that we are concerned here with a very low-level notion of style: stipples are dots of varying size, pointillism introduces color, hatching needs to deal with orientation (e.g. for cross-hatching), and painterly refers to the use of small paint strokes of varying color and orientation. There is much more research to do to characterize how a style can be specified, but this falls out of the scope of this paper.

Our stroke-based rendering prototype system is similar in spirit to previous painting systems [Mei96, DOM*01, HE04, PFS03] in that it separates low-level strokes drawing from high-level style control. At each frame, stroke parameters are

© The Eurographics Association 2007.

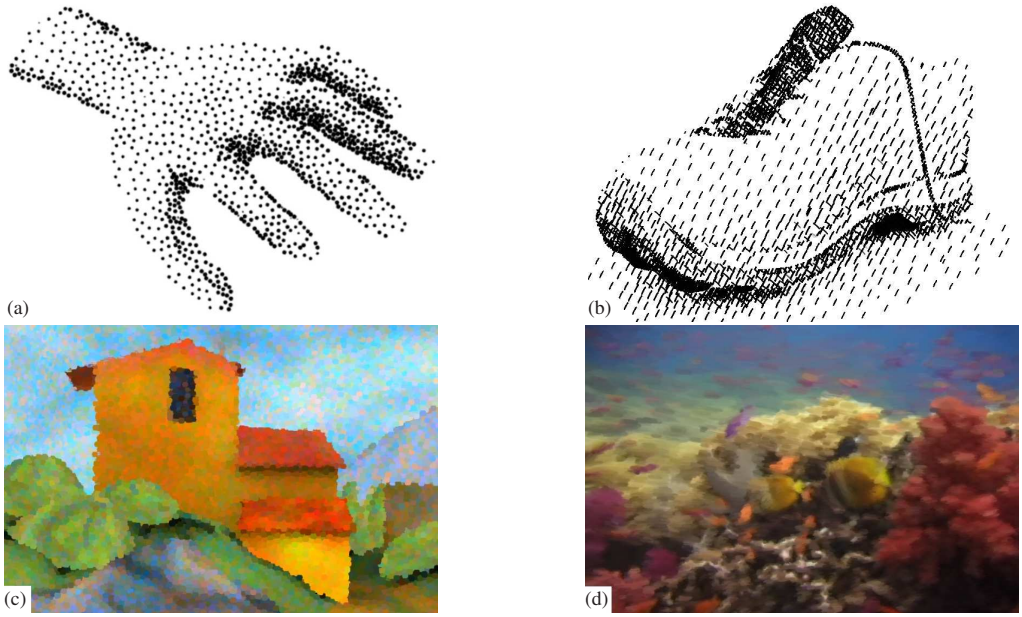


Figure 7: *Our distribution can be used as a basis for various styles:* ***(a)*** *Stippling rendition of a deformable hand model,* ***(b)*** *a hatching style applied to a rigid shoe model,* ***(c)*** *pointillist rendering of a scene consisting in multiple objects and* ***(d)*** *a painterly rendering of a video that follows optical flow.*

computed for the whole image and stored in a style buffer. Stroke anchor points are then distributed using our dynamic algorithm. And finally, primitives are drawn by taking their parameters at their anchor point location in the style buffer. Note however that nothing is done to make stroke parameters evolve coherently, hence stroke orientation, length or thickness may change suddenly in the animations. Such temporal artifacts are specific to every new style and we plan to study temporal coherence for individual styles in future work.

Stroke density is a special parameter that deserves more attention since it is common to all styles. In our system, it is controlled by a time-varying importance map that is input to the distribution algorithm at each frame. As it plays an important role in anchor points distribution, we implemented a variety of importance mappings in the form of GPU shaders, so that new mappings can be added easily. Each style uses a different importance mapping, as shown in Figure 7 and in the accompanying video.

Stippling: For stippling, we use an approach similar to Pastor et al. [PFS03]: stipples are simply black dots of varying size; their density follows the luminance information coming from the scene (our importance map); and stipples are blended in or out by making them grow or shrink.

Pointillism: Pointillism is a variant of stippling where we make use of color information to color the stipples, and the point distribution is close to uniform. The points are added or removed by alpha blending. Moreover, as in [Mei96], we jitter each point's color in the Lab color space.

Hatching: Hatching introduces two new parameters: strokes length and orientation. In order to make the distribution visible in our results, we intentionally chose a simple style: stroke length and orientation are uniform across the image. To create cross-hatching, we simply create two independent distributions of strokes with different orientations. Moreover, as in previous work [HP00, HE04], we cut strokes at object boundaries, that we measure as depth discontinuities.

Painterly: Finally, painterly rendering combines all the characteristics of previous styles at once (color, length and orientation) and adds another parameter: stroke thickness. We again set the parameter values to uniform to better show the underlying distribution. Paint strokes are blended using the half-toning technique of Durand et al. [DOM*01].

6. Discussion and Future Work

The work presented in this paper proposes an new approach to the dynamic distribution of anchor points for stroke-based rendering: it consists in a hybrid method that combines the advantages of distributing points in image space and the richness of object-space motions.

In the near future we plan to adapt our algorithm to work with more complex motions, such as specular reflexions motion, or natural phenomena (e.g. fluid, smoke, etc). Indeed, the only requirement in our case is to be able to project a point to the scene, and after it has been moved, to project it back to the picture plane.

© The Eurographics Association 2007.

Another area of research resides in using more complex strokes, such as long paint strokes, or mosaics. Already, when dealing with stippling, we cannot ensure an accurate tone reproduction, even if we get satisfying results in practice. While with hatching or painterly styles, one interesting question would be to adapt the distribution to primitives direction (e.g. using "Poisson-ellipses"). But this will not be enough as soon as long strokes will be involved. In this case, one possibility would be to use our approach as an underlying structure, and attach long strokes to multiple anchor points. And in the case of mosaics, another issue is raised: we need to meet boundary constraints between primitives. Our approach already offers good packing properties, but additional mechanisms are needed to make mosaic primitives "touch" each other without inter-penetrating.

Finally, while our distribution, by construction, makes use of the coherence of points from frame to frame, we additionally proposed two temporal coherence mechanisms: blending and sliding. Our simple assumption being that by reducing the number of insertions and deletions, and making the remaining ones less noticeable, we effectively enhance coherence from frame to frame. Interestingly, while temporal coherence is a recurrent problem in non-photorealistic rendering, there is no standard way to measure it to our knowledge, making a formal study an interesting avenue of future work.

Acknowledgments

We would like to thank Florent Moulin and Laurence Boissieux for the 3D models; and Lionel Baboud, Cyril Soler, Kartic Subr and Kaleigh Smith for their suggestions.

References

[Coo86] COOK R. L.: Stochastic sampling in computer graphics. *ACM Trans. Graph. 5*, 1 (1986), 51–72.

[CRL01] CORNISH D., ROWAN A., LUEBKE D.: View-dependent particles for interactive non-photorealistic rendering. In *Proceedings of Graphics Interface 2001* (2001), pp. 151–158.

[Dan99] DANIELS E.: Deep Canvas in Disney's Tarzan. In *SIGGRAPH 99 Conference Abstracts and Applications* (1999), p. 200.

[DOM*01] DURAND F., OSTROMOUKHOV V., MILLER M., DURANLEAU F., DORSEY J.: Decoupling strokes and high-level attributes for interactive traditional drawing. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques* (2001), pp. 71–82.

[HE04] HAYS J., ESSA I.: Image and video based painterly animation. In *NPAR'2004: international symposium on non-photorealistic animation and rendering* (2004), pp. 113–120.

[Her03] HERTZMANN A.: A survey of stroke-based rendering. *IEEE Computer Graphics and Applications 23*, 4 (July/August 2003), 70–81. Special Issue on Non-Photorealistic Rendering.

[HP00] HERTZMANN A., PERLIN K.: Painterly rendering for video and interaction. In *NPAR'2000: international symposium on non-photorealistic animation and rendering* (2000), pp. 7–12.

[HS04] HALLER M., SPERL D.: Real-time painterly rendering for MR applications. In *GRAPHITE '04* (2004), pp. 30–38.

[HSK*05] HAVRAN V., SMYK M., KRAWCZYK G., MYSZKOWSKI K., SEIDEL H.-P.: Importance Sampling for Video Environment Maps. In *Rendering Techniques 2005 (Proceedings of the Eurographics Symposium on Rendering)* (2005), pp. 31–42,311.

[KCODL06] KOPF J., COHEN-OR D., DEUSSEN O., LISCHINSKI D.: Recursive wang tiles for real-time blue noise. *ACM Transactions on Graphics 25*, 3 (2006).

[KMM*02] KALNINS R. D., MARKOSIAN L., MEIER B. J., KOWALSKI M. A., LEE J. C., DAVIDSON P. L., WEBB M., HUGHES J. F., FINKELSTEIN A.: WYSIWYG NPR: Drawing Strokes Directly on 3D Models. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2002) 21*, 3 (July 2002), 755–762.

[LD06] LAGAE A., DUTRÉ P.: *A Comparison of Methods for Generating Poisson Disk Distributions*. Report CW 459, Department of Computer Science, K.U.Leuven, Leuven, Belgium, August 2006.

[Mei96] MEIER B. J.: Painterly rendering for animation. In *SIGGRAPH'96* (1996), pp. 477–484.

[MF92] MCCOOL M., FIUME E.: Hierarchical poisson disk sampling distributions. In *Proceedings of the conference on Graphics interface '92* (1992), pp. 94–105.

[NS04] NEHAB D., SHILANE P.: Stratified point sampling of 3d models. In *Eurographics Symposium on Point-Based Graphics* (June 2004), pp. 49–56.

[Pas03] PASTOR O. E. M.: *Frame-Coherent 3D Stippling for Non-photorealistic Computer Graphics*. PhD thesis, Otto-von-Guericke University of Magdeburg, Germany, October 2003.

[PFS03] PASTOR O. M., FREUDENBERG B., STROTHOTTE T.: Real-time animated stippling. *IEEE Computer Graphics and Applications 23*, 4 (2003).

[SHS02] SECORD A., HEIDRICH W., STREIT L.: Fast primitive distribution for illustration. In *Rendering Techniques 2002 (Proceedings of the Eurographics Symposium on Rendering)* (2002), pp. 215–226.

[SLK05] SMITH K., LIU Y., KLEIN A.: Animosaics. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation* (2005), pp. 201–208.

© The Eurographics Association 2007.

Structure-preserving manipulation of photographs

Alexandrina Orzan Adrien Bousseau Pascal Barla Joëlle Thollot
ARTIS - INRIA Grenoble University

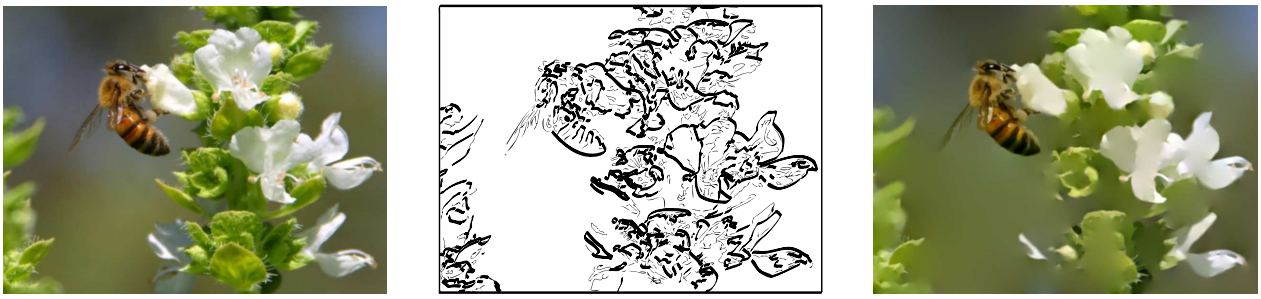


Figure 1: *Our approach takes as input an image (left), and allows a user to manipulate its structure in order to create abstracted or enhanced output images. Here we show a line drawing with line thickness proportional to their structural importance (middle), and a reconstruction of color information that focuses on the bee and removes detail around it (right).*

Abstract

Visual content is often better communicated by simplified or exaggerated images than by the "real world like" images. In this paper, we offer a tool for creating such enhanced representations of photographs in a way consistent with the original image content. To do so, we develop a method to identify the relevant image structures and their importance. Our approach (a) uses edges as the basic structural unit in the image, (b) proposes tools to manipulate this structure in a flexible way, and (c) employs gradient domain image processing techniques to reconstruct the final image from a "cropped" gradient information. This edge-based approach to non-photorealistic image processing is made feasible by two new techniques we introduce: an addition to the Gaussian scale space theory to compute a perceptually meaningful hierarchy of structures, and a contrast estimation method necessary for faithful gradient-based reconstructions. We finally present various applications that manipulate image structure in different ways.

CR Categories: I.4.3 [Image Processing and Computer Vision]: Enhancement; I.3.3 [Computer Graphics]: Picture/Image Generation;

Keywords: Visual communication, multiscale analysis, image processing, image reconstruction.

1 Introduction

Effective visual communication is not always best achieved by the "real-world like" images. Simplified objects or exaggerated features can often improve perception and facilitate comprehension. This observation led researchers to investigate new non-photorealistic image processing techniques, as in the work of DeCarlo and Santella [2002] that explicitly aims at better conveying a message visually by means of perceptual considerations. In particular, such an image manipulation approach offers a tool for transmitting effective visual information by grabbing visual attention.

However, to modify images in a way consistent with the information content of the image, one must first identify the relevant *structures* and their *importance* and then ensure they are preserved through manipulations. The main goal of this paper is precisely to give insights into "what structure means" when we have **no a priori** about semantics, and to provide the user with image manipulation tools to create enhanced or abstracted representations of photographs in accordance with their structural information.

Taking a look at Figure 2 that represents a hand-made scientific illustration, it is clear that the main subject of the image is the butterfly: it is depicted with many details, while plants around are more or less suggested. However, while abstracted, secondary elements of the image retain their look and are easily identified; in other words, their relevant structural components are preserved through the abstraction process. Similarly, our approach is to provide high-level *structural* information to guide user image manipulations. This is a significant step compared to previous approaches.

Gaussian scale space theory has been developed by the computer vision community to deal with structure identification in images with no a priori information. This theory models the first stages of human vision (front-end vision) and extracts features that are perceptually important. A relevant structure is defined as an element that is *invariant* to various viewing conditions; other elements can be considered "accidental", and of less importance. Out of many invariants in an image (edges, corners, ridges, curvatures, etc), *edges* contain most of the visually important information, because the human visual system is very sensitive to contrast variations [Palmer 1999]. Moreover, an edge-based representation of images provides a flexible and simple way of merging multi-scale information. We therefore use edges as the structural unit in our image manipulation and derive their importance from the scale space analysis.

Edges together with their importance form a hierarchy of structures that can be easily manipulated by the user to reflect what is semantically important to her. We then build on existing gradient domain manipulation techniques to reconstruct the final image from the modified edge structure.

Figure 2: *"Le Papillon" (The Butterfly), watercolor by Eric Alibert. From "Leman, mon ile", © 2000 by Editions Slatkin. As seen in the guidebook of scientific illustration [2003].*

The major contribution of this paper is to combine these two existing techniques: scale space analysis and gradient domain image editing, to provide new *structure-oriented* image manipulation methods. Moreover, the successful adaptation of these techniques to this new domain has required extending them via two specific technical contributions. For scale space analysis, we propose a new approach to extract image structure that identifies meaningful edges according to their importance in the scale space. For gradient domain image manipulation, we propose a novel reconstruction method from sparse edge fields that carefully reintroduces blur and contrast information.

Regarding our contributions, we would like to emphasize that this work is *not* about a new stylization technique. It is rather a starting point for any subsequent stylization. However, we show applications, that differ in the way they manipulate image structure. We provide three types of manipulation: levels-of-detail, shape simplification, and importance-based line drawing. We hope that our research will motivate the development of other applications that take advantage of image structure.

The paper is organized in six sections. After presenting previous work (Section 2), we give quick overviews of Gaussian scale-space image analysis and gradient domain image manipulation (Section 3). We then present our method (Section 4) and various applications along with results (Section 5) and implementation details (Section 6). We finally discuss limitations, and propose possible extensions as future work (Section 7).

2 Previous work

A number of previous techniques focused on creating enhanced or abstracted renderings from arbitrary photographs.

Generally the previous methods manipulate an image globally without using the image structure [Winnemöller et al. 2006], or rely on the user to define what is important [Wang et al. 2004; Kang et al. 2006; Wen et al. 2006]. As a result, the content either cannot be controlled, or its control involves tedious user interactions. We are interested in automatically extracting the relevant structural information to enrich automatic systems or assist the user in her task.

Previous work made use of Gaussian scale space [Hertzmann 1998] or saliency maps [Collomosse and Hall 2005; Collomosse and Hall 2003] in order to guide painterly stylizations. However, saliency maps identify image regions that already grab visual attention in the original image, and using them to guide stylization will only preserve these attention-grabbing regions. In contrast, our goal is to extract a structure that allows the user to *intentionally* manipulate the image, possibly modifying its attention focus (i.e. changing its subject, see Figure 1 - right), and hence conveying a particular message.

DeCarlo and Santella [2002; 2004] were the first to use a meaningful visual structure in photo abstraction. They use color regions as structural units and create their hierarchy of regions from a pyramid of down-sampled versions of the image. But for coarser-level regions the shape simplifies and the borders move slightly. There is therefore no perfect overlap between finer and coarser regions. When mixing different levels of detail in the same image, this becomes problematic because one doesn't know how to unify information at different scales.

Bangham et al. [2003] extend DeCarlo and Santella's work by improving the region segmentation. Their region hierarchy is based on a morphological scale-space and has the advantage of preserving region shapes. But since only the region size is considered, and not its contrast, they tend to eliminate visually important cues that have a high contrast but small size.

In general, multi-scale region approaches have the inconvenience of associating a solid color to each region. This fact introduces visible color discontinuities between regions and demands to explicitly treat color mixing when two regions merge together. The end result is a poster-like effect in the final rendering. In contrast, our structural hierarchy is an edge-based one that allows us to avoid the problems generated by region-based methods. In particular, our edges are not required to be closed contours, as opposed to region boundaries, and hence they do not create erroneous color discontinuities.

Edge representation of images has been used in previous work of course, although not with the same purpose. Elder et al. [2001] use the edge domain to ease image editing operations (crop, delete, paste), but have no concept of edge importance. Perez et al. [2003] suggest using gradient information only at edge locations as input for a Poisson solver, in order to obtain a texture flattening effect. We improve on this method with the aim of manipulating an image for abstraction and/or enhancement purposes by (a) giving insights into how image structure can be manipulated, and (b) by providing a new reconstruction method that extends [Pérez et al. 2003].

3 Background

In order to manipulate images in a structure-preserving way, our method relies on two image processing tools: Gaussian scale space and gradient domain image manipulation. In the following, we give a quick overview of both tools and we provide the reasons for choosing them for our purpose. The bottom line is that Gaussian scale space will be responsible for extracting the structure of edges, while gradient domain processing will be used for reconstruction.

3.1 Gaussian scale space

Scale space methods base their approach on representing the image at multiple scales, ensuring that fine-scale structures are successively suppressed and no new elements are added (the so-called "causality property" [Koenderink 1984]).

The motivation for constructing scale-space representations originates from the basic fact that real-world objects are composed of different structures at different scales of observation. Hence, if no prior information is available about the image content, the state-of-the-art approach for deriving the image structure is to use the successive disappearance of scale features to create a hierarchy of structures [Romeny 2003].

Gaussian scale space is the result of two different research directions: one looking for a scale-space that would fit the axiomatic basis stating that "we know nothing about the image" and the other searching for a model for the front-end human vision [Fischler and Firschein 1987; Wandell 1995; Romeny 2003]. Since our purpose is to define a human-vision-like importance measure for an image content we have no a priori on, this scale-space fits our needs.

A scale-space is a stack of images of increasing scales. The basic Gaussian scale space is thus a stack of images convolved by Gaussian kernels of increasing variance [1]. In the general case, Gaussian derivatives of any order can be used to build the stack, allowing one to create scale-spaces of edges, ridges, corners, laplacians, curvatures, etc.

Edge representations, as discontinuities in image brightness, retain important data about objects in the image (shape, surface orientation, reflectance) [Lindeberg 1998]. We thus settle on studying the image structures represented by a hierarchy of edges in the Gaussian scale space. As edges are defined by gradient information, we only need to convolve the original image with Gaussian derivatives of order 1, one for each image dimension. These Gaussian derivatives G_x and G_y are computed as follows:

$$\begin{aligned} G_x(x,y;\sigma) &= g(y)\cdot g'(x) \text{ and} \\ G_y(x,y;\sigma) &= g(x)\cdot g'(y) \end{aligned}$$

with

$$g(i) = \frac{e^{-\frac{i^2}{2\sigma^2}}}{\sqrt{2\pi}\sigma} \quad \text{and} \quad g'(i) = -\frac{e^{-\frac{i^2}{2\sigma^2}} i}{\sqrt{2\pi}\sigma^3}$$

where the width σ of the kernel corresponds to scale and $i \in \{x, y\}$. Given an input image I, we thus build two different scale spaces: an horizontal gradient $I_x = I \otimes G_x$ and a vertical gradient $I_y = I \otimes G_y$.

3.2 Gradient domain image manipulation

Many recent works introduced gradient manipulations as an efficient tool for image processing. The main reason is that gradient represents the image variations independently of the original colors, allowing more flexibility in image manipulations. Handling directly the image variations makes possible applications such as seamless image editing [Pérez et al. 2003] and image fusion [Agarwala et al. 2004; Raskar et al. 2004]. Gradient domain is also an intuitive representation for image contrast [Fattal et al. 2002].

We propose to combine the flexibility of gradient domain manipulations to the high level control provided by the gradient scale space. This allows us to seamlessly mix information from multiple scales.

Working in the gradient domain implies one can reconstruct an image I from its gradient field $\mathbf{w} = (\mathbf{w}_x, \mathbf{w}_y)$. As a manipulated gradient is unlikely to be conservative and integrable, a common approach is to compute an estimation of the image whose gradient field best fits $\mathbf{w}$ in a least-square minimization sense:

$$\arg\min_I \int_\Omega (\nabla I - \mathbf{w})^2 d\Omega$$

[1] For numerical stability, one usually starts with a variance $\sigma_0 = 1$

This estimation corresponds to the unique solution of the Poisson equation $\Delta I = \text{div}\mathbf{w}$, where Δ and div are the Laplace and divergence operators [Pérez et al. 2003; Fattal et al. 2002].

4 Our approach

Figure 3 illustrates our approach: we first apply Gaussian scale-space analysis to the input image I to get gradient values at multiple scales $(I_x, I_y)_\sigma$; then we manipulate this rich information in a way that preserves the structure of the image, giving rise to a gradient field $\mathbf{w} = (\mathbf{w}_x, \mathbf{w}_y)$; finally, the output image O is built from $\mathbf{w}$ using Poisson reconstruction. When using a color image as input, we use only its luminance during the multi-scale analysis.

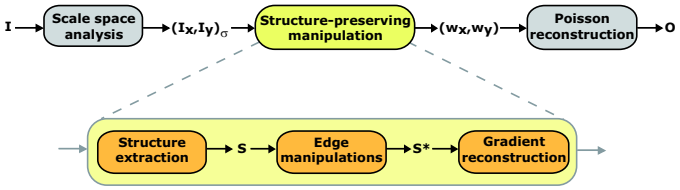


Figure 3: *Overview of our method.*

Our structure-preserving manipulation represents the heart of the approach and is composed of three steps:

1. **Structure extraction:** starting from the raw multi-scale gradient values, we extract the image structure S corresponding to the edges, their importance and profile.
2. **Edge manipulations:** We then use the structure S as a high-level control for user-defined image manipulations, and output a manipulated structure S^*.
3. **Gradient reconstruction:** We finally reconstruct a gradient field from the set of manipulated edges with their profile.

In the following section we mainly present the two technical steps of the method: structure extraction and gradient reconstruction. Several edge manipulation techniques are presented in Section 5.

4.1 Structure extraction

4.1.1 Edge extraction

From the first-order Gaussian derivative scale spaces, we want to build a hierarchy of edges holding structural importance. Before defining what we mean by "structural importance", we first extract edges at all the available scales in order to get the richest possible information. For this task we use a Canny edge detector [Canny 1986]: it is a state-of-the-art edge detection method that processes the Gaussian derivative information at each scale to give thin, binary edges. Its main quality resides in using hysteresis thresholding that results in long connected paths and avoids small noisy edges (see Figure 4).

After applying the Canny detector, we are left with a multi-scale binary mask C_σ that indicates at each scale the edges locations. Figure 5 illustrates such a typical edge scale-space for a simple 1D example. Due to the nature of Gaussian scale-space, three different cases can occur: (a) an edge exists and suddenly drops off at a higher scale; (b) two edges are coming toward each other and collapse at a higher scale; (c) some "blurry" edges only appear at a higher scale. To simplify further computations, we "drag" edges corresponding to case (c) down to the minimum scale and note C^*_σ the resulting multiscale edge mask.

Figure 4: *Edge importance. (a) The input image. (b-d) Canny edges at increasing scales. (e) The lifetime measure reflects the importance of edges: "older" edges correspond to more stable and important structures.*

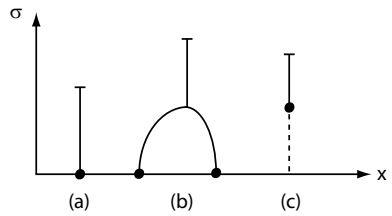


Figure 5: *Three different events in a 1D Gaussian scale-space: (a) an edge drops off at a high scale; (b) two edges collapse ; (c) a blurry edge is created. In the last case, we drag the edge down to the finest scale for convenience.*

4.1.2 Edge importance

As shown in Figure 5, there is a great deal of coherence along the scale dimension in the multi-scale edge representation. The main idea behind scale-space techniques is to try to extract this coherent *deep structure*, by linking edges at different scales. In particular, because of the causality property of Gaussian scale-space, an edge that disappears at a given scale will not reappear at a higher scale; hence an important measure of structure along scale is *lifetime*, as edges that live longer will correspond to more stable structures.

Unfortunately, extracting an edge lifetime is not trivial, since edges move in Gaussian scale-space (this corresponds to Figure 5 case (b)). This motivated edge focusing techniques, that track edges at increasing scales. In this paper, we take an alternative approach which revealed simpler to implement: instead of considering each pixel p belonging to an edge, we consider its projected point $\mathcal{P}_\sigma(p)$ onto the closest edge at scale σ (we use a distance field for this purpose). We can then define the membership of any pixel $m_\sigma(p)$ as the binary function that indicates whether p can be considered to belong to an edge at scale σ:

$$m_\sigma(p) = \begin{cases} 1 \text{ if } ||\mathcal{P}_\sigma(p) - p|| < T_\sigma \\ 0 \text{ otherwise} \end{cases}$$

The choice of the threshold distance T_σ is essential to get a good approximation for our membership function. Bergholm [1987] proved that the edge shifting is less than a pixel when the scale σ varies by less than 0.5. Therefore, we increase our σ values by $\Delta\sigma = 0.4$ at each scale and use $T_\sigma = \sigma/\Delta\sigma$. This approach is similar in spirit to the morphological linking method of Papari et al. [2007].

Finally, using membership for linking purpose, we compute the lifetime $L(p)$ at each edge pixel p in the finest scale by summing up membership values. Considering the successive scale values $\sigma_i, i \in 1..N$, where N is the size of our scale-space stack, we write lifetime as:

$$L(p) = \arg\min_i \{\sigma_i | m_{\sigma_i}(p) = 0\}$$

This can be seen as a simpler, easier-to-manipulate version of Lindeberg edge strength measure [Lindeberg 1998]. We can now use lifetime as a measure of structural importance to manipulate edges in a structure-preserving way, as shown in Section 5.

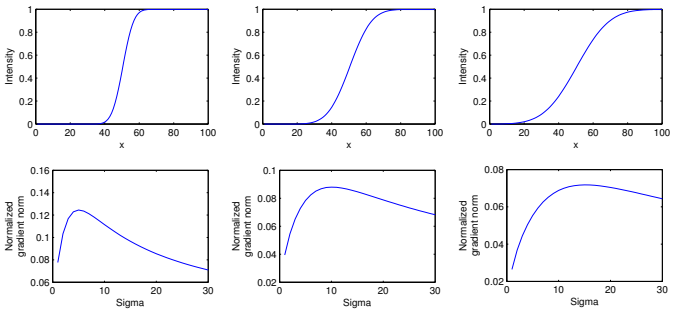


Figure 6: *Best scale estimation. Top: 1D edges blurred with* $\{\sigma_i\} = \{5, 10, 15\}$. *Bottom: normalized gradient magnitude scale space proposed by Lindeberg. The best-scale measures (the local maxima) are at the* $\{\sigma_i\}$ *used for blurring, hence they represent well each edge profile.*

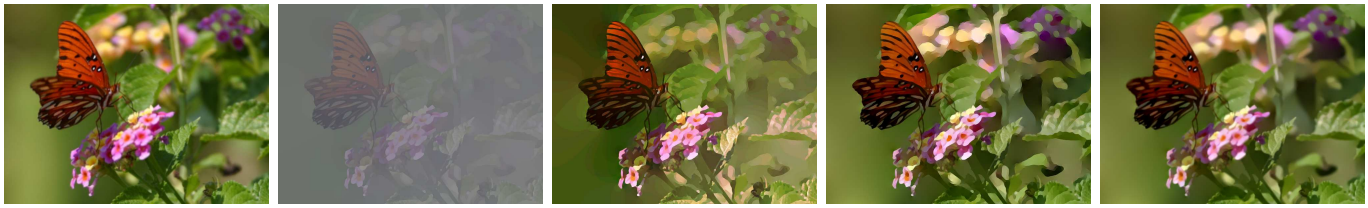
4.1.3 Edge profile

In the previous section, we mainly relied on edge locations and their persistence along scale. Another concern is to deal with their *profile* (contrast value and degree of blur) . In this paper, as in previous work [Lindeberg 1998; Elder and Goldberg 2001], we rely on a simple assumption: the profile of an edge gradient is modeled as the convolution of a Dirac (its location and contrast) with a spatially varying Gaussian kernel (its blur). For instance, in a photograph with depth-of-field, out-of-focus edges are blurry (with a wide profile) while in-focus edges are sharp (with a thin profile).

Our second measure of structure then consists, for each edge, in finding the *best scale* that locally corresponds to its blur. This is also the scale where we measure the contrast.

The best scale search is another form of *deep structure* that has been studied by Lindeberg [1998]. Following his approach, we first compute a normalized gradient magnitude scale-space by $||\nabla I|| = \sqrt{\sigma(I_x^2 + I_y^2)}$. The best scale $B(p)$ at an edge pixel p is then identified as the one which gives the first local maxima along the scale axis in this normalized gradient magnitude stack. But as with lifetime computation, we need to link "moving edges" at different scales using the projection operator $\mathcal{P}_\sigma$ again: $||\nabla I(p)|| = ||\nabla I(\mathcal{P}_{\sigma_i}(p))||$. Figure 6 shows how best scales can be well estimated for edges of increasing blur.

We are now able to "re-blur" the edges using the best scale. Moreover, we will also make use of this measure to find a correct contrast in order to get edge profiles back into the output image.



(a) (b) (c) (d) (e)

Figure 7: *Gradient reconstruction. (a) Input image. (b) Reconstructed image using only the original gradient values at edge positions. (c) Reconstructed image with histogram equalization. Note the quantization artefacts. (d) Reconstructed image using contrast correction. Note that blurry edges become sharp if the profile is not taken into account. (e) Full reconstruction using contrast correction and re-blurring.*

4.2 Edge manipulations

The multi-scale Canny edges, together with their lifetime and best scale finally constitute the structure $S = \{C^*_\sigma, L, B\}$ we extracted from the input image. This structure can be manipulated in various ways depending on the application (see Section 5). The main idea is to select a subset E of the multi-scale Canny edges C^*_σ according to lifetime L. After manipulation, we are thus left with a new, simpler structure $S^* = \{E, B\}$.

4.3 Gradient reconstruction

We wish to reconstruct the corresponding image by solving a Poisson equation, i.e. we want to build a vector field $\mathbf{w}$ that corresponds to our new edges.

We propose to use the scale space information to estimate the original gradient profiles and correctly reproduce the contrast and blur of the input image. However, taking the original gradient values at edge locations as suggested by Perez et al. [2003] results in a gradient field that does not capture the whole original contrast, nor the original blur (Figure 7, (a) and (b)). This is because we only consider the central value of the profile, loosing all its surrounding informations.

A simple solution to the contrast problem would be to apply a histogram equalization on the reconstructed image to match the original contrast. However the very low dynamic range of the reconstructed image leads to strong quantization artifacts (Figure 7(c)).

We thus need to take into account our knowledge of edge profiles to compute the correct contrast. Our model of an edge represents blurry edges that appear in the input image I as the convolution of a step function H by a 2D Gaussian kernel G_B, where B is the local best scale. When we measure I_x (resp. I_y) at scale B on edge locations, we get the following contrast values:

$$I_x = H \otimes G_B \otimes \frac{\partial G_B}{\partial x} = H \otimes \frac{\partial G_{B_2}}{\partial x} = \frac{\partial H}{\partial x} \otimes G_{B_2}$$

with $B_2 = \sqrt{2B^2}$. However, to recover the original contrast value of the profile, we are precisely interested in the value of $\frac{\partial H}{\partial x}$. This corresponds to the deconvolution of I_x (resp. I_y) by G_{B_2}. Unfortunately, deconvolution is known as an ill-posed problem, particularly sensitive to noise and quantization [Romeny 2003]. To avoid this problem, we propose to simplify our model for the sake of contrast correction: we replace the 2D Gaussian derivative by a 1D Gaussian derivative $\widetilde{G}_x = g'(x)$. This way, we can derive an analytical solution for the correction problem (see appendix): for each edge pixel p, we only need to multiply the gradient value found in I_x (resp. I_y) by $2B(p)\sqrt{\pi}$. This correction gives a final contrast close to the original one, and we find that our approximation works well in practice, with no visible artefacts (see Figure 7(d)).

(a) (b) (c) (d)

Figure 8: *Detail removal: (a) original image, and (b-d) several levels-of-detail automatically generated by our method.*

Finally, even if using edge locations and correcting their contrast does give a convincing result, blurry edges become sharp in the reconstructed image. Therefore, we also re-blur the edges, as seen in Figure 7(e). This process remains optional as the sharp result provides an interesting cartoon style.

5 Applications

Most of the image manipulations presented in this paper can be seen as variations of recently proposed methods that take advantage of the flexibility of the gradient domain. Our contribution is to use the high-level structural information provided by our approach to guide these gradient manipulations.

5.1 Detail removal

We use the lifetime information as a threshold value to seamlessly remove details while keeping important structures. Such image editing operations are similar to the seamless cut and paste operations proposed by Perez et al. [2003] and Elder et al. [2001], except that we provide a high level control to the user, who has only to select the desired level of detail (Figure 8).

5.2 Multi-scale shape abstraction

We propose a shape abstraction method that adapts the level of abstraction to the scale of the features in order to preserve the informative content of the picture. In practice, we select for each edge its last available version in the scale space using lifetime. As

Figure 9: *Shape abstraction: (a) original image, and (b) our shape abstraction result. Notice how the thin details are kept, while shapes of bigger objects are abstracted (e.g. the poles).*

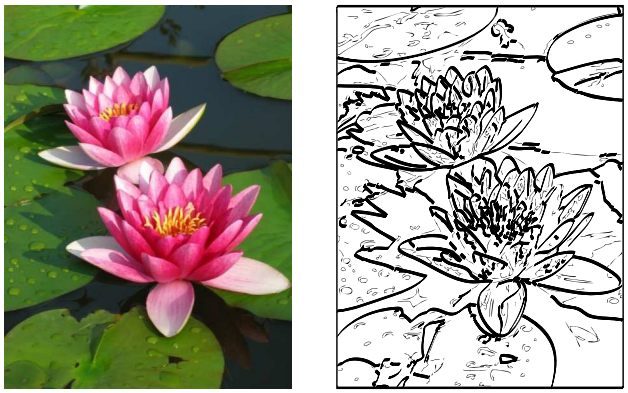


Figure 10: *Vectorized edges, with a larger width for relevant structures (i.e. those having greater lifetime).*

shapes become more and more smoothed along scales due to the Gaussian filter, relevant structures will have increasingly rounded shapes while details will keep their original silhouettes.

In opposition to previous approaches [DeCarlo and Santella 2002] that remove texture details and abstract shapes at the same time, our approach selects for each edge (including edges belonging to texture details or other small elements) the shape of its last scale. Hence, our approach still keeps most of the meaningful structural information, while simplifying its shape, as seen in Figure 9.

This application can be seen as a fusion of multi-scale images, similar in spirit to other image fusion methods like the ones of Agarwala et al. [2004] and Raskar et al. [2004].

5.3 Line drawing

The edge lifetime information offers a powerful high-level parameter for any line drawing algorithm. Figure 10 presents the rendering of vectorized edges with a different width to enhance important structures from details. Figure 1 (middle) also shows an example of this application.

5.4 Local control

In order to offer a local control to the user, each image manipulation can be weighted by a gray-level map indicating the desired amount of abstraction (Figure 12). This mechanism is essential to be able to focus on a given zone of the input image, and efficiently grabs visual attention. We take advantage of the Poisson reconstruction to obtain seamless transitions between regions of different weight.

6 Implementation

In our approach, we clearly did not focus on performance, but rather on how to extract and use image structure: our current implementation[2] is in Matlab, with performance times of approximately 10 minutes for the whole process of our approach, considering an 800×600 input image and a scale-space depth of $N = 30$. However, most of this time is spent in the structure extraction, and the Poisson reconstruction takes only about 2 seconds; once structure has been computed, it can be manipulated rather efficiently.

To solve the Poisson equation on the manipulated gradient field, we use the sine transform based Poisson solver of Simchony et al. [1990] with Dirichlet conditions. We use the Matlab implementation provided by Agrawal et al.[3]

7 Discussion and future work

7.1 Discussion

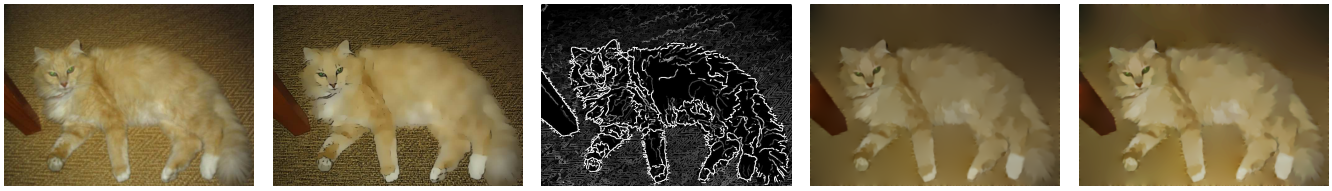
In our exploration of deep structure, we mainly took inspiration from Lindeberg [1998]; indeed, he has a notion similar to lifetime, and the best scale measure is directly borrowed from his approach. One alternative for measuring the best scale is the method introduced by Elder et al. [1998; 2001], based on local signal-to-noise ratios. But this approach is not easy to combine with our importance measure, making Lindeberg's method better suited to our purposes. However, there is a main difference between Lindeberg's work and ours: we separate the importance of edges from their contrast and profile, while he deals with all this information at once. Our approach has the advantage of being easier to manipulate: one can modify any property without affecting the others.

This is well illustrated in Figure 11: here we show a failure case of Winnemöller et al.'s abstraction approach [2006]. Although their method gives convincing results in many cases, this specific example shows how they cannot get rid of high-contrast texture lines without abstracting the cat too far; in contrast, our approach allows us to simply remove detail edges regardless of their contrast. Another interesting example comes up when we compare our results to DeCarlo et al. [2002] as in Figure 13: while their method couple simplification of shape with detail suppression, ours allows to remove details *without* necessarily simplifying shapes.

Another choice we made is to use Poisson reconstruction methods. This body of techniques have an advantage over other diffusion approaches: it is independent from the input image. For instance, while a diffusion method will try to blur an unwanted detail, a Poisson approach will simply ignore it in the reconstruction. This is again well illustrated by the example in Figure 11, since the texture lines simply do not appear in our image. Another advantage is that it gives smooth results: when compared to the stylized image of DeCarlo et al. in Figure 13, we can clearly see that our method better represents color variations and avoids the introduction of arbitrary color discontinuities. However, these advantages come at a cost: it is hard to reconstruct an image with a correct contrast. This is the reason why we introduced our contrast correction method. We can also perform histogram equalization as a post-process, as shown in Figure 11 (d) and (e).

[2] http://artis.imag.fr/Publications/2007/OBBT07/

[3] http://www.umiacs.umd.edu/users/aagrawal/software.html



(a) (b) (c) (d) (e)

Figure 11: *Comparison with the failure case of Winnemöller et al. [2006]. (a) Original picture. (b) Winnemöller et al. abstraction failure: note how the carpet details are preserved while the fur is abstracted away. (c) Our lifetime map. (d) Our detail removal abstraction preserves the cat structure and abstract the carpet. (e) We apply histogram equalization as a post-process to fine tune contrast.*

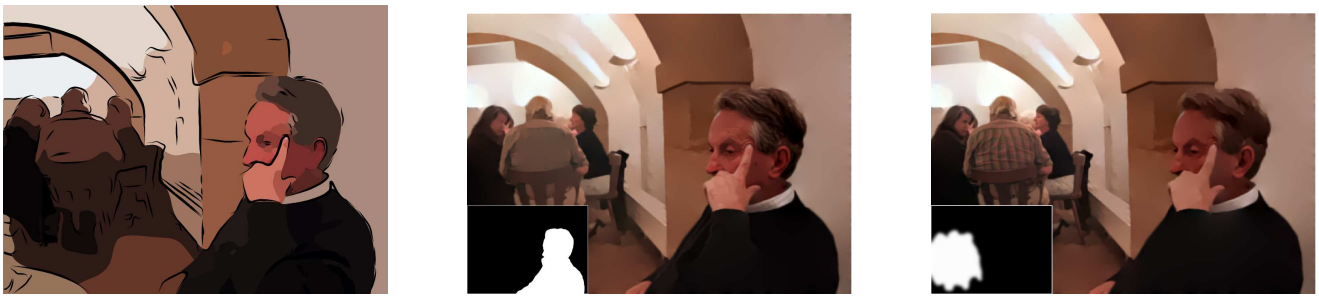


Figure 12: *Local control: original image of DeCarlo et al. [2002] and our results for two different user-specified control maps.*

Finally, one may wonder why we have not used Elder et al.'s contour domain approach [Elder and Goldberg 2001] instead of the Poisson reconstruction. Although their approach could be used for most of the stages of our method, the fact that they need to handle colors on both side of edges makes the manipulation stage less flexible.

7.2 Future work

Our approach extracts structure from a luminance image, but uses this structure to reconstruct color images. This approach works in most cases, since luminance carries a lot of the structural information. However, when iso-luminant color regions "touch" each other in the input image, our method tends to fail to reconstruct correct colors, as shown in Figure 14. We envision two ways to solve this problem in future work: one would be to use color-to-gray methods [Gooch et al. 2005] that would introduce color discontinuities in our original luminance image; the other one would require to extract edges for different channels, but it raises the problem of combining this disparate information in the end.

Another avenue for future work resides in the design of an intuitive and efficient user interface. For instance, we plan to develop free-hand tools to manipulate directly edges at different scales: when selecting an edge at a coarse scale, the user could manipulate many details at a time, while selecting an edge at a fine scale would allow her to edit details in subtle ways. Applying our approach to process videos is also an exciting area of future work: Gaussian scale-space and Poisson reconstruction approaches have been already applied successfully to videos, and the flexibility of our structural measures would make them tractable for manipulating image sequences.

Finally, as we see our approach as a starting point for any subsequent stylization, we are also interested in developing such stylized renditions that take advantage of structural information. As an example, we created two preliminary results, shown in Figure 15: a drawing, and a watercolor. There are many connections to establish between style parameters and structure information, and we hope this work will motivate future research along this direction.

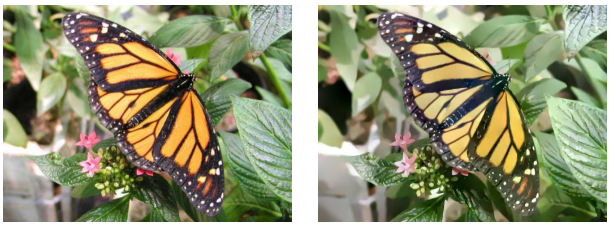


Figure 14: *Failure in the image reconstruction due to the iso-luminance of the pink flowers and green leaves, which leads to a greenish butterfly.*

Figure 15: *Different stylizations obtained from our abstracted images, in a drawing and watercolor style.*

Acknowledgments

Alexandrina Orzan is supported by a grant from the European Community under the Marie-Curie project VISITOR MEST-CT-2004-008270.

(a) (b) (c)

Figure 13: *Comparison with the DeCarlo et al. [2002]. (a) Original picture. (b) DeCarlo et al. results exhibit flat color regions with shape simplification (c) Our result simplifies the image while keeping smooth color variations and original shapes.*

Appendix

We model a directional edge gradient $I_{\{x,y\}}$ as the 1D convolution of a step function H of amplitude A by a Gaussian kernel g_σ and a Gaussian derivative g'_σ, resulting in:

$$\begin{aligned} I_x(0) &= \big(H \otimes g_\sigma \otimes g'_\sigma\big)(0) = \big(H \otimes g'_{\sqrt{2\sigma^2}}\big)(0) \\ &= \int_{-\infty}^{+\infty} H(t) \cdot g'_{\sqrt{2\sigma^2}}(-t)dt = \int_{0}^{+\infty} A \cdot g'_{\sqrt{2\sigma^2}}(-t)dt \\ &= A \cdot g_{\sqrt{2\sigma^2}}(0) = \frac{A}{2\sigma\sqrt{\pi}} \end{aligned}$$

References

AGARWALA, A., DONTCHEVA, M., AGRAWALA, M., DRUCKER, S., COLBURN, A., CURLESS, B., SALESIN, D., AND COHEN, M. 2004. Interactive digital photomontage. *ACM TOG (Proceedings of SIGGRAPH 2004)*, 294–302.

BANGHAM, J., GIBSON, S., AND HARVEY, R. 2003. The art of scale-space. In *British Machine Vision Conference*, 569–578.

BERGHOLM, F. 1987. Edge focusing. *IEEE Transactions on Pattern Analysis and Machine Intelligence 9*, 6, 726–741.

CANNY, J. 1986. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence 8*, 6, 679–698.

COLLOMOSSE, J. P., AND HALL, P. M. 2003. Cubist style rendering from photographs. *IEEE Transactions on Visualization and Computer Graphics 9*, 4, 443–453.

COLLOMOSSE, J. P., AND HALL, P. M. 2005. Genetic paint: A search for salient paintings. In *Proc. of EvoMUSART*, 437–447.

DECARLO, D., AND SANTELLA, A. 2002. Stylization and abstraction of photographs. *ACM TOG (Proceedings of SIGGRAPH 2002) 21*, 3, 769–776.

ELDER, J. H., AND GOLDBERG, R. M. 2001. Image editing in the contour domain. *IEEE Transactions on Pattern Analysis and Machine Intelligence 23*, 3, 291–296.

ELDER, J. H., AND ZUCKER, S. W. 1998. Local scale control for edge detection and blur estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 20*, 7, 699–716.

FATTAL, R., LISCHINSKI, D., AND WERMAN, M. 2002. Gradient domain high dynamic range compression. *ACM TOG (Proceedings of SIGGRAPH 2002)*, 249–256.

FISCHLER, M. A., AND FIRSCHEIN, O. 1987. *Intelligence: The Eye, the Brain and the Computer*. Addison-Wesley.

GOOCH, A. A., OLSEN, S. C., TUMBLIN, J., AND GOOCH, B. 2005. Color2gray: salience-preserving color removal. *ACM TOG (Proceedings of SIGGRAPH 2005) 24*, 3.

HERTZMANN, A. 1998. Painterly rendering with curved brush strokes of multiple sizes. In *SIGGRAPH '98*, 453–460.

HODGES, E. R. S. 2003. *Guild Handbook of Scientific Illustration*. John Wiley & Sons, Inc.

KANG, H. W., CHUI, C. K., AND CHAKRABORTY, U. K. 2006. A unified scheme for adaptive stroke-based rendering. *The Visual Computer (Proceedings of Pacific Graphics 2006) 22*, 9, 814–824.

KOENDERINK, J. J. 1984. The structure of images. *Biological Cybernetics 50*, 5, 363–370.

LINDEBERG, T. 1998. Edge detection and ridge detection with automatic scale selection. *International Journal of Computer Vision 30*, 2, 117–154.

PALMER, S. 1999. *Vision Science : Photons to Phenomenology.* MIT Press.

PAPARI, G., CAMPISI, P., PETKOV, N., AND NERI, A. 2007. A biologically motivated multiresolution approach to contour detection. *EURASIP Journal on Advances in Signal Processing 2007*.

PÉREZ, P., GANGNET, M., AND BLAKE, A. 2003. Poisson image editing. *ACM TOG (Proceedings of SIGGRAPH 2003)*.

RASKAR, R., ILIE, A., AND YU, J. 2004. Image fusion for context enhancement and video surrealism. In *Proceedings of NPAR 2004*, 85–152.

ROMENY, B. T. H. 2003. *Front-End Vision and Multi-Scale Image Analysis*. Kluwer Academic Publishers.

SANTELLA, A., AND DECARLO, D. 2004. Visual interest and npr: an evaluation and manifesto. In *Proceedings of NPAR 2004*, 71–150.

SIMCHONY, T., CHELLAPPA, R., AND SHAO, M. 1990. Direct analytical methods for solving poisson equations in computer vision problems. *IEEE Transaction on Pattern Analysis and Machine Intelligence 12*, 5, 435–446.

WANDELL, B. A. 1995. *Foundations of Vision*. Sinauer Associates.

WANG, J., XU, Y., SHUM, H.-Y., AND COHEN, M. F. 2004. Video tooning. *ACM TOG (Proceedings of SIGGRAPH 2004) 23*, 3, 574–583.

WEN, F., LUAN, Q., LIANG, L., XU, Y.-Q., AND SHUM, H.-Y. 2006. Color sketch generation. In *Proceedings of NPAR 2006*.

WINNEMÖLLER, H., OLSEN, S. C., AND GOOCH, B. 2006. Real-time video abstraction. *ACM TOG (Proceedings of SIGGRAPH 2006) 25*, 3, 1221–1226.

Video Watercolorization using Bidirectional Texture Advection

Adrien Bousseau[1,3] Fabrice Neyret[2] Joëlle Thollot[3] David Salesin[1,4]
[1]Adobe Systems [2]LJK/IMAG-INRIA [3]INRIA Grenoble University [4]University of Washington
{Adrien.Bousseau|Fabrice.Neyret|Joelle.Thollot}@imag.fr salesin@adobe.com

Figure 1: A watercolorized video. *Original video (left), and its watercolorization (right).*

Abstract

In this paper, we present a method for creating watercolor-like animation, starting from video as input. The method involves two main steps: applying textures that simulate a watercolor appearance; and creating a simplified, abstracted version of the video to which the texturing operations are applied. Both of these steps are subject to highly visible temporal artifacts, so the primary technical contributions of the paper are extensions of previous methods for texturing and abstraction to provide temporal coherence when applied to video sequences. To maintain coherence for textures, we employ texture advection along lines of optical flow. We furthermore extend previous approaches by incorporating advection in both forward and reverse directions through the video, which allows for minimal texture distortion, particularly in areas of disocclusion that are otherwise highly problematic. To maintain coherence for abstraction, we employ mathematical morphology extended to the temporal domain, using filters whose temporal extents are locally controlled by the degree of distortions in the optical flow. Together, these techniques provide the first practical and robust approach for producing watercolor animations from video, which we demonstrate with a number of examples.

Keywords: Non-photorealistic rendering, abstract stylization, animated textures, temporal coherence.

1 Introduction

Watercolors have two essential characteristics that make them especially beautiful and evocative. They have distinctive textures — including those due to separation and granulation of pigments, as well as the texture of the paper itself, which shows through on account of watercolors' transparency. And they have an ability to suggest detail through abstract washes of color. In recent years, researchers in computer graphics have found ways to reproduce much of watercolors' characteristic appearance, taking an image as input and producing a watercolor-like illustration as output in a process known as "watercolorization" [Curtis et al. 1997; Lum and Ma 2001; Van Laerhoven et al. 2004; Bousseau et al. 2006]. However, the same characteristics that give watercolors their distinctive charm are nonetheless quite difficult to reproduce in any temporally coherent fashion, so the analogous process of taking a *video* as input and producing a watercolor-like *animation* as a result has remained a longstanding and elusive goal. In this paper, we show how this goal can be achieved to a large extent.

We use two different approaches to achieve temporal coherence. To preserve coherence of the watercolor texture itself, we advect a set of pigmentation textures according to an optical flow field computed for the video. And to produce a temporally coherent abstraction of the video, we use 3D mathematical morphology operations to create spatially and temporally smooth (i.e., non-flickering), simplified areas to render with watercolor textures.

The major technical contribution of the paper is the combination and application of these two existing algorithmic techniques, *texture advection* and *mathematical morphology*, to a new domain.[1] Moreover, the successful adaptation of these techniques to this new domain has required extending them in a number of ways. For advection, we propose a new scheme that involves simultaneously advecting two different texture fields, in forward and reverse directions through the video, and optimizing their combination on a per-pixel basis so as to yield minimal texture distortion — even in areas of disocclusion, which are notoriously difficult to handle with previous approaches. For abstraction via mathematical morphology, we extend traditional approaches by also filtering over time, and by taking into account errors and distortions in optical flow to control the temporal extent of the 3D filters. Together, these contributions provide the first practical and reasonably robust approach for producing realistic watercolor animations from video sequences.

In the rest of this paper, we will briefly survey related work (Section 2), describe our approach to texture advection in detail (Section 3), present our extensions of mathematical morphology for abstraction (Section 4), demonstrate our results (Section 5), and finally discuss areas for future work (Section 6).

[1]Advection has previously been used for visualizing fluids and flow, while mathematical morphology has typically been used for image denoising.

2 Related work

A significant body of research has been devoted to creating watercolor rendering for static images [Small 1991; Curtis et al. 1997; Lum and Ma 2001; Lei and Chang 2004; Van Laerhoven et al. 2004; Johan et al. 2005; Luft and Deussen 2006; Bousseau et al. 2006]. Commercial tools, like Photoshop^{TM}and The GIMP, also provide ways of creating watercolor-like renderings from images.

In general, the difficulty with extending non-photorealistic rendering (NPR) techniques from static to moving images is that, without careful consideration to temporal coherence, the resulting animations exhibit one of two problems: either the illustration effects remain fixed in place across the image, an unwanted artifact that has become known as the "shower door" effect; or the illustration effects exhibit no temporal coherence whatsoever, randomly changing in position and appearance from frame to frame, which can be even more visually distracting.

Several techniques have been developed to combat these problems in various NPR styles, although most of this work concerns the problem of animated 3D scenes, in which geometry information is available. For watercolorization, these approaches rely on texture mapping in 3D [Lum and Ma 2001], or on the distribution of discrete 2D primitives over surfaces [Luft and Deussen 2006; Bousseau et al. 2006]. Similarly, most of the work on video stylization has been in the realm of primitive-based rendering, where discrete paint strokes [Litwinowicz 1997; Hertzmann and Perlin 2000; Hays and Essa 2004; Collomosse et al. 2005] or other primitives [Klein et al. 2002] are applied to create the stylization. Applying such methods to watercolor raises the difficult question of finding a set of 2D primitives that, once combined, produces a continuous texture (typically pigments or paper). In that spirit, Bousseau et al. [2006] has proposed to use a Gaussian kernel as a primitive for watercolor effects, but was restricted to Perlin noise. We rather evolve the texture globally, allowing us to deal with any scanned texture.

A related problem has been addressed by Cunzi et al. [2003], who use a "dynamic canvas" to preserve the appearance of a 2D background paper texture while rendering an interactive walkthrough of a static 3D scene. Our work takes inspiration from theirs to perform a similar kind of dynamic adaptation of our watercolor textures, while at the same time tracking dynamic object motions in a changing video scene. Our work also shares some similarity to Fang's "RotoTexture" [Fang 2006], in that both attempt to provide textures that track dynamic scenes; however, Fang's work is concerned with maintaining the appearance of 3D textures rather than 2D. Other work in scientific visualization [Max and Becker 1995; Jobard et al. 2001], fluid simulation [Stam 1999; Neyret 2003], and artistic image warping [Sims 1992] shares the goal of evolving texture along a 2D vector field. Our work builds on the advection approach that these schemes introduced.

A significant body of research has been concerned with the issue of abstraction of video as well. Winnemöller et al. [2006] presented a method to smooth a video using a bilateral filter, which reduces the contrast in low-contrast regions while preserving sharp edges. We use a similar approach to produce large uniform color areas, and we go a step further in simplifying not just the color information but the 2D scene geometry as well. In "video tooning," Wang et al. [2004] use a mean-shift operator on the 3D video data to cluster colors in space and time. To smooth geometric detail, they employ a polyhedral reconstruction of the 3D clusters followed by mesh smoothing to obtain coarser shapes. Collomosse et al. [2005] use a similar type of geometric smoothing involving fitting continuous surfaces to voxel objects. Such high-level operations are usually computationally expensive and may sometimes require user input to produce convincing results. Our approach, by contract, uses simple low-level image processing for geometric as well as color simplification. In essence, we extend the approach of Bousseau et al. [2006], who use a morphological 2D filter to abstract the shapes of a still image and mimic the characteristic roundness of watercolor, by extending the morphological filter to the 3D spatiotemporal volume in a way that provides temporal coherence as well.

Finally, many image filtering operations have been extended to the spatiotemporal domain to process video: the median filter [Alp et al. 1990], average filter [Ozkan et al. 1993], and Wiener filter [Kokaram 1998] are all examples. A motion compensation is usually applied before filtering to avoid ghosting artifacts in regions of high motion. Recently, Laveau and Bernard [2005] proposed orienting a morphological structuring element along an optical flow path in order to apply these operators on videos. However, their filters have been developed in the context of noise removal, which requires filters of small support. Our application is more extreme in that it targets the removal of significant image features larger than a single pixel. To this end, we propose a type of adaptive structuring element that smoothes the appearance and disappearance of significant image features.

3 Texture advection

The "granulation" of watercolor pigments provides a large share of the richness of continuous tone techniques. Granulation is caused by pigment deposition on paper or canvas: the more pigments are deposited, the more the color is saturated. The effect is seen in both wet techniques like watercolor and dry techniques like charcoal or pastel. In addition, watercolors are transparent, and their transparency allows the texture of the paper itself to show through.

To achieve these texture effects, we follow the approach of Bousseau et al. [2006], who showed that for computer-generated watercolor, convincing visual results are obtained by considering a base color image C (e.g., a photograph) that is modified according to a grey-level pigment texture $P \in [0,1]$ at each pixel to produce a modified color C' according to the equation

$$C' = C\big(1-(1-C)(P-0.5)\big) \tag{1}$$

However, in order to create effective watercolor animations, this texture P must satisfy two competing constraints: On the one hand, it must maintain its appearance in terms of a more or less homogeneous distribution and frequency spectrum. On the other hand, it must follow the motion in the scene so as not to create a "shower door" effect.

To resolve this conflict we build on previous work on advected textures [Max and Becker 1995; Neyret 2003], classically used to depict flow in scientific visualizations. The general idea of such methods is to initialize the texture mapping on the first frame of an animation, and then evolve the mapping with the motion flow. In these methods the texture mapping is reinitialized whenever the statistical spatial properties of the current texture become too dissimilar to the original one.

We employ this same basic idea to our situation of applying texture to video, substituting optical flow for fluid flow. One significant complication, which arises quite frequently for videos but not for continuous fluid flows simulations, is the occurrence of *disocclusion boundaries*: places where new pixels of the background are revealed as a foreground object moves across a scene. Optical flow fields at disocclusions are essentially undefined: there is no pixel in the prior frame corresponding to these disoccluded boundary regions. Classical advected textures are designed for handling only continuous space-time distortions. In the absence of continuity, they tend to fail quite badly as shown Figures 3 and 4.

In order to handle disoccluded boundaries effectively, we introduce the notion of *bidirectional advection*: simultaneously advecting two texture layers in opposite directions in time — from the first frame of the video to the last, and from the last frame to the first. We use a combination of these two texture layers weighted at each pixel by the local quality of the texture from each direction. In the rest of this section we describe our algorithm and its properties.

3.1 Advection computation

In the following, we will use $\mathbf{x} = (x, y)$ for screen coordinates, and $\mathbf{u} = (u, v)$ for texture coordinates. An advected texture relies on a field of texture coordinates $\mathbf{u}(\mathbf{x}, t)$, which is displaced following a vector field $\mathbf{v}(\mathbf{x}, t)$: for any frame t, the vector $\mathbf{u}$ defines the location within the texture image P_0 to be displayed at position $\mathbf{x}$, i.e. the *mapping*. For simplicity we assume that $\mathbf{x}$ and $\mathbf{u}$ coordinates are normalized on the interval $[0, 1]$ and that $\mathbf{u}(\mathbf{x}, 0) = \mathbf{x}$. Therefore in Equation 1 the composited pigment texture will be $P(\mathbf{x}, t) = P_0(\mathbf{u}(\mathbf{x}, t))$. We will see in the following that several such layers will be combined to obtain the final result. Our vector field is obtained from an optical flow extracted from the video (we rely on a classical gradient-based method available in Adobe After Effects™). The vector $\mathbf{v}$ indicates for each frame t where the pixel at position $\mathbf{x}$ came from within frame $t-1$: $\mathbf{v}(\mathbf{x}, t) = \mathbf{x}_{t-1} - \mathbf{x}_t$.

The purpose of advection is to "attach" the texture P_0 to the moving pixels of the video, which is theoretically done by displacing the texture coordinates according to the vector field:[2] $\mathbf{u}(\mathbf{x}, t) = \mathbf{u}(\mathbf{x} + \mathbf{v}(\mathbf{x}, t), t - 1)$. However, this backward mapping is problematic wherever the optical flow is poor or ill-defined, as at disocclusion boundaries. We will discuss how these problematic cases are handled momentarily.

3.2 Controlling the distortion

With this basic approach, the distortion of the advected texture increases with time. To combat this distortion, Max and Becker's [1995] and Neyret's [2003] approaches blend two or three phase-shifted texture layers, respectively (See Figure 2). In both schemes, the distortion is reset periodically (i.e., $\mathbf{u}(\mathbf{x}, t + \tau) = \mathbf{u}(\mathbf{x}, t)$), allowing the original texture to be used again. The regeneration period τ is chosen via a user defined delay [Max and Becker 1995] or a dynamic estimation of the distortion [Neyret 2003]. The fact that the regeneration occurs periodically guaranties that our system can handle videos of arbitrary length.

Like Max and Becker, we rely on two texture layers, but we combine one "forward" mapping $\mathbf{u}_f$, advected from the beginning to the end of the video sequence, with one "reverse" mapping $\mathbf{u}_r$, advected from the end to the beginning. The final advected pigment texture P' is a combination of these two fields, calculated on a per-pixel basis by taking into account the local distortions ω_f and ω_r of the forward and reverse mappings $\mathbf{u}_f$ and $\mathbf{u}_r$, respectively, within a given frame:

$$P'(\mathbf{x}, t) \;=\; \omega_f(\mathbf{x}, t) P_0\big(\mathbf{u}_f(\mathbf{x}, t)\big) \;+\; \omega_r(\mathbf{x}, t) P_0\big(\mathbf{u}_r(\mathbf{x}, t)\big)$$

We will show precisely how the distortion is measured and how ω_f and ω_r are computed later on. For now, to understand the intuition behind this approach, it suffices to note that texture distortion gradually increases in the direction of advection. Since $\mathbf{u}_f$ is advected forward, its distortion increases with time. However, since $\mathbf{u}_r$ is advected backwards through the video sequence, its distortion *decreases* with time. The combination of the two textures can therefore be used to create a less distorted texture. Moreover, a disocclusion in the forward sequence becomes an occlusion in the reverse, which is no longer a source of distortion; thus, a well-defined texture can always be used. (A similar observation was noted by Chuang et al. [2002] in their work on video matting.)

[2] In order to manage boundary conditions properly, we assume that the texture P_0 is periodic, and that $\mathbf{u}(\mathbf{x}, t) \approx \mathbf{u}(\mathbf{x}, t-1) + \frac{\partial \mathbf{u}}{\partial \mathbf{x}} \cdot \mathbf{v}(\mathbf{x}, t)$ whenever $\mathbf{x} + \mathbf{v}(\mathbf{x}, t)$ seeks outside $[0, 1]^2$, in the spirit of [Max and Becker 1995].

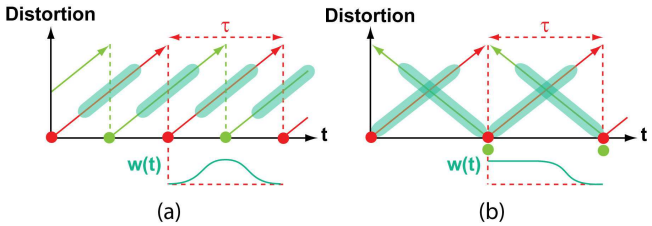


Figure 2: Approaches to control the distortion of the advected texture. (a) *Max and Becker scheme, which uses two phase-shifted, overlapping texture layers at any given time.* (b) *Our scheme, which also uses two texture layers at a time but advected in opposite time directions. In the two diagrams, the green solid areas represents, roughly, the relative contribution of each advected layer to the final texture's appearance, which corresponds to the weight $w(t)$. Note that in prior work* (a)*, in order to maintain temporal coherence, the advected layers do not begin contributing significantly until they are already somewhat distorted. In our method* (b)*, by contrast, textures contribute maximally where they are least distorted. Blending textures advected in opposite time directions also allows for less distortion everywhere, since distortion is decreasing in one texture just as it is increasing in the other. Finally, bidirectional advection handles disocclusion boundaries much better since the disocclusion leaves one of the two layers unaffected, with the unaffected layer contribution most to the texture's final appearance.*

The technical challenges of this approach are to quantify the visual distortions and to choose a clever way of combining the two advected fields. Three competing goals have to be taken into account: (1) minimizing the distortion; (2) avoiding temporal discontinuities; and (3) limiting visual artifacts such as contrast variation.

In the rest of this section, we detail our method: how we quantify the distortion (Section 3.3), and how we adjust the weights of the two advected fields (Section 3.4).

3.3 Distortion computation

We need a way to estimate the visual quality of a distorted texture at each pixel. Various methods exist to compute the distortion of a non-rigid shape. The general principle is to compute a deformation tensor and to choose an appropriate norm to compute the distortion.

All deformation tensors are based on the deformation gradient tensor F, corresponding to the Jacobian matrix of the shape coordinates (in our case the texture coordinates): $F_{ij}(\mathbf{x}, t) = \partial \mathbf{u}_i(\mathbf{x}, t) / \partial x_j$. As in previous work, we do not wish to consider translations and rotations to be distortions because they do not alter the visual properties of the texture. Instead, it is typical to rely on a strain tensor, which cancels antisymmetric components. However, unlike Neyret [2003], we wish to deal with large displacements, so we cannot use the infinitesimal strain tensor, which is the classical approximation. We therefore choose the Cauchy-Green tensor: $G = F^T F$ (one can verify that multiplying F by its transpose cancels the rotations). The eigenvectors of G give the principal directions of deformations, and the tensor's eigenvalues λ_i give the squares of the principal deformation values: an eigenvalue $\lambda_i > 1$ corresponds to a stretch, whereas an eigenvalue $\lambda_i < 1$ corresponds to a compression.

We want to derive an estimation of the visual quality of the *distortion* ξ as a scalar in $[0, 1]$ with 0 representing no distortion, and 1 representing distortion that is intolerable. We assume that compres-

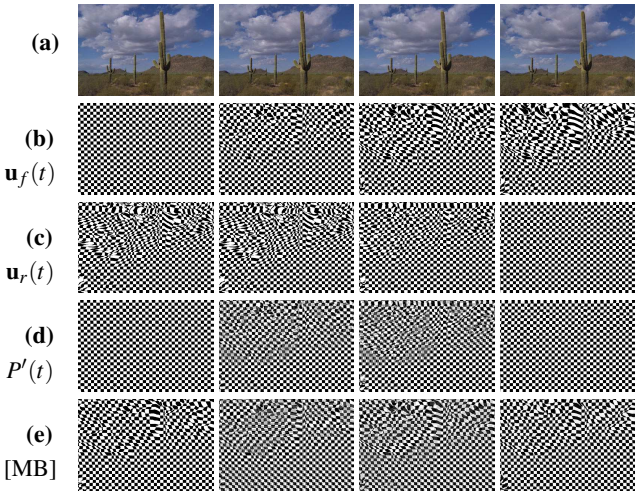


Figure 3: Analysis of our bidirectional advected texture. (a) *Original video sequence.* (b) *A checkerboard texture (for illustration purposes), advected forward using optical flow.* (c) *The same texture, advected in the reverse time direction using reverse optical flow.* (d) *The combined, bidirectional advected texture.* (e) *Advected texture using the previous approach of Max and Becker. Note how the bidirectional advected texture in* (d) *shows much less distortion in all frames than the previous approach in* (e).

sion and stretching are equally bad from a visual standpoint. We therefore define the *visual deformation* in the two principal directions σ_1 and σ_2 as: $\sigma_i(\mathbf{x},t) = \max\left(\sqrt{\lambda_i(\mathbf{x},t)}, 1/\sqrt{\lambda_i(\mathbf{x},t)}\right)$.

We define the visual distortion ξ as the quadratic mean of both deformations, normalized to $[0,1]$:

$$\xi(\mathbf{x},t) \quad = \quad \frac{\xi'(\mathbf{x},t) - \xi_{min}}{\xi_{max} - \xi_{min}}$$

where $\xi'(\mathbf{x},t) = \sqrt{\sigma_1^2(\mathbf{x},t) + \sigma_2^2(\mathbf{x},t)}$ is an unnormalized scalar measure of the visual distortion; $\xi_{min} = \sqrt{2}$ is the minimum value of ξ' (representing no distortion); and ξ_{max} is a maximum bound over which the distortion is considered too high, measured experimentally. In practice we use $\xi_{max} = 5$.

3.4 Adjusting weights

Given the distortions ξ_f and ξ_r of each advected mapping $\mathbf{u}_f$ and $\mathbf{u}_r$, we must now find the appropriate set of weights ω_f and ω_r at each pixel in order to minimize the final distortion.

Our weights must satisfy a number of properties: They should lie on the interval $[0,1]$; sum to 1 at every pixel; be inversely related to the texture distortion; gradually decrease to 0 at regeneration events in order to maintain overall temporal continuity; and vary smoothly, both in space and in time.

We choose the following definition for the weights, which satisfy all of these properties:

$$\omega_f(\mathbf{x},t) = \frac{\omega'_f(\mathbf{x},t)}{\omega'_f(\mathbf{x},t) + \omega'_r(\mathbf{x},t)} \qquad \omega_r(\mathbf{x},t) = \frac{\omega'_r(\mathbf{x},t)}{\omega'_f(\mathbf{x},t) + \omega'_r(\mathbf{x},t)}$$

with

$$\omega'_f(\mathbf{x},t) = g_f(\mathbf{x},t)\, h_f(t) \qquad \omega'_r(\mathbf{x},t) = g_r(\mathbf{x},t)\, h_r(t)$$

Here, g_f and g_r are measures of the distortions of the forward and reverse textures relative to the other:

$$g_f(\mathbf{x},t) = \frac{1 - (\xi_f - \xi_r)}{2} \qquad g_r(\mathbf{x},t) = \frac{1 - (\xi_r - \xi_f)}{2}$$

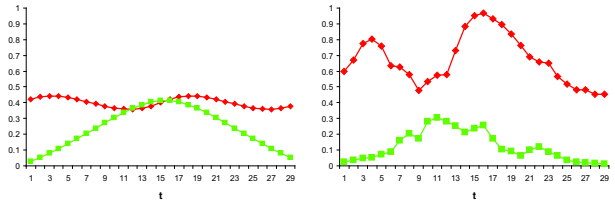


Figure 4: Comparison of distortion rates for our method (green squares) vs. Max and Becker (red diamonds). *Shown are resulting distortions for a theoretical case, in which distortion increases at a constant rate* (left)*; and a real case, measured at a disoccluded pixel (event occurs at* $t = 12$*)* (rigth). *Note how our method is able to cancel large distortions almost entirely.*

and h_f and h_r are the temporally decaying weighting functions shown in Figure 2:

$$h_f(t) = \cos^2\left(\frac{\pi}{2}\,\frac{t \bmod \tau}{\tau}\right) \qquad h_r(t) = \sin^2\left(\frac{\pi}{2}\,\frac{t \bmod \tau}{\tau}\right)$$

where τ is the delay between two regenerations.

Figure 3 shows the resulting advected texture for a test scene, and Figure 4 shows the resulting distortion values for two different pixels. If the distortion rate is regular, our blending behaves much like that of Max and Becker [1995]; however, when the distortion rate is high, for example at disocclusion boundaries, our blending results are much better.

3.5 Limiting contrast oscillation and tuning τ

The blending of the two advected layers produces a cycle between frames with one single texture ($\omega_f = 1$ or $\omega_r = 1$) and frames with two blended textures (for all other values of ω_f and ω_r). Contrast is reduced, especially as ω_f and ω_r approach the same value of $1/2$, because high frequencies are dimmed. To reduce the perception of the resulting oscillation of contrast, we include a second pair of advected layers with a regeneration cycle halfway out of phase. The average of these two advected textures gives a nearly constant amount of contrast. As two out-of-phase identical textures moving in the same direction may produce spatial correlation (i.e., ghosting), we use a visually similar but distinct texture image for this pair. Thus, our complete model relies on four layers.

In previous methods, the regeneration period τ is chosen as a trade-off between texture distortion and loss of motion perception (if texture is regenerated too often there is no longer advection). As pointed out by Neyret [2003], the ideal value of τ is generally not the same in each region of the image, depending on the local velocity. Thus, Neyret proposes an adaptive regeneration scheme, which consists of advecting several texture layers with increasing regeneration periods, and locally interpolating between layers to minimize the distortion. In this way, fast regeneration occurs only in regions of high distortion. Our method already has some spatial adaptation. Still, the same idea is also applicable in our case since velocity varies in the optical flow. Being able to choose between various periods also helps minimize the distortion due to disocclusion. In practice, we found that two pairs of layers were sufficient in most cases. One fast layer ($\tau = 15$ frames) allows for good correction of disocclusions, while a slower layer ($\tau = 30$ or 60 frames) provides good motion perception in slow-motion areas.

4 Shape abstraction

One of the distinctive characteristics of watercolor is the medium's ability to suggest detail with abstract washes of color. In this section, we examine how raw video footage can be abstracted into

shapes that are more characteristic of watercolor renderings, while remaining temporally coherent.

To this end, we generalize the work of Bousseau et al. [2006], who use mathematical morphology filters to create more regular image regions. While relatively little known, such filters have long been described as having the ability to "simplify image data, preserving their essential shape characteristics and eliminating irrelevancies" [Haralick et al. 1987]. In the following we briefly present the generalized versions of the morphological operators for continuous-tone images, before extending them to temporally coherent operators adapted to video stylization. For further details on morphological filtering, see the overview of Serra and Vincent [1992] or the work of Haralick et al. [1987].

4.1 Morphological operators

Figure 5: Mathematical morphology operators, generalized for continuous-tone images. (a) *Original image.* (b) *Erosion.* (c) *Dilation.* (d) *Opening (erosion followed by dilation).* (e) *Closing (dilation followed by erosion).* (f) *Closing followed by opening: our chosen morphological filter.*

Let I be an image and B a *structuring element*, that is an array that contains the relative coordinates of a pixel's neighborhood. The *morphological dilation* δ of I by B at a pixel $\mathbf{x}$ and its dual, the *morphological erosion* ε, are defined by

$$\delta_B(I)(\mathbf{x}) = \max_{b\in B}\{I(\mathbf{x}-b)\} \quad \varepsilon_B(I)(\mathbf{x}) = \min_{b\in B}\{I(\mathbf{x}+b)\}$$

The dilation spreads the light features of the image whereas the erosion spreads the dark features. (See Figure 5.)

The *morphological opening* is then defined as a sequence of one erosion followed by one dilation, $\delta_B \circ \varepsilon_B(I)$, and the *morphological closing* as one dilation followed by one erosion $\varepsilon_B \circ \delta_B(I)$. Opening removes light features of the image, whereas closing removes dark features.

An effective filter for removing both dark and light features, used by Bousseau et al. [2006], is the sequence of one closing followed by one opening. In this case, the size of the morphological structuring element defines the minimum size of the features preserved by the filtering, and the shape of the structuring element defines the shape of the filtered objects in the resulting image. For simplicity, the operators are applied on the three color channels separately. Although this independent filtering of the channels produces color ghosting on dilation and erosion (see Figure 5(b,c)), it becomes unnoticeable when these dual operators are applied in sequence (see Figure 5(d,e,f)).

4.2 Spatiotemporal morphological operators

Applying morphological filtering on each image separately produces a great deal of flickering, as many features appear and disappear at every frame (See Figure 6.) Moreover, as each feature is at least as large as the 2D structuring element, the features' appearances and disappearances produce noticeable "popping" artifacts. To reduce temporal artifacts, we extend the structuring element to the temporal domain. The resulting 3D structuring element can be thought of as a stack of neighborhoods in k successive frames. The 3D element reduces flickering as it guarantees that each new feature remains visible during all k frames. However, care must be taken to correctly orient the filter. Indeed, moving objects correspond to spatiotemporal regions that are not necessarily perpendicular to the time axis. As in previous work [Ozkan et al. 1993; Kokaram 1998], we compensate for this motion by translating the neighborhood at each pixel according to the optical flow for each frame. The resulting filtering produces uniform temporal regions, resulting in a higher temporal coherence.

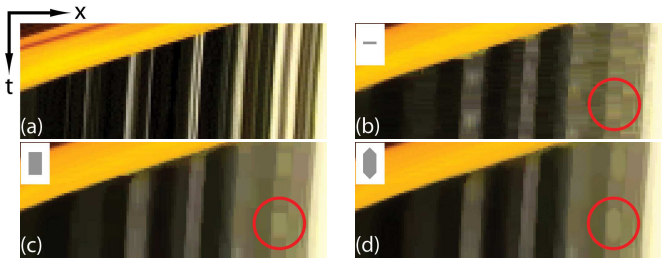


Figure 6: A visualization of the effects of various filters on the spatiotemporal volume (filter represented in gray). *These figures show a portion of a single horizontal scanline of video as a function of time, which appears on the vertical axis.* (a) *Original video.* (b) *The results of a 2D morphological filter (note the vertical noise, which corresponds to flickering in the filtered sequence).* (c) *The results of a constant-width spatiotemporal filter (note the sudden onset of features, corresponding to popping in the filtered sequence).* (d) *The results of our tapered filter (note how features appear and disappear gradually rather than suddenly).*

Unlike previous methods in video denoising, which used small filter kernels (usually $3\times3\times3$ pixels), we would like to use kernels with much larger spatiotemporal support (generally, $7\times7\times9$ pixels) to abstract away significant details. In order to reduce the popping produced by the abrupt appearance or disappearance of these details, we design the structuring element in such a way that visibility events occur gradually. As such, we taper the structuring element at its extremities (see Figure 7). The features in the resulting video appear and disappear gradually, as shown in Figure 6(d). Indeed, their shapes in the spatiotemporal volume, visualized in this figure, mirror precisely the shape of the tapered 3D structuring element that we use.

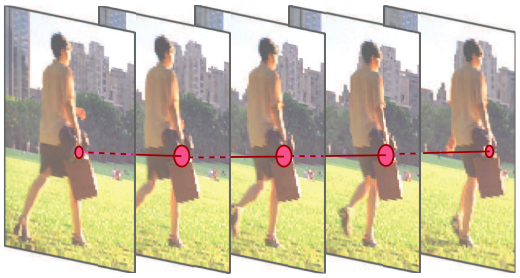


Figure 7: A visualization of how our tapered morphological filters follow the optical flow from frame to frame.

Finally, proper spatiotemporal filtering assumes that every pixel has a correct motion trajectory throughout the temporal support of each filter. In practice, this assumption fails for several reasons: First, a pixel can become occluded by another object in the video. Second, optical flow is usually just an approximate estimation of the motion flow, especially in disocclusion areas where no real correspondence

can be found between two successive frames. A common approach to increase the robustness for spatiotemporal filtering in such cases is to make the filter adaptive [Ozkan et al. 1993]: the idea is to ignore the contribution of outliers in the filtered value. However, a direct application of this method to morphological operators would create holes in the structuring element, which would have a direct impact on the shapes in the resulting image. Instead, we simply truncate the structuring element in the temporal domain as soon as an "outlier" is detected. A pixel is flagged as an outlier when its estimated motion accuracy is below a defined threshold. Similarly to Chuang et al. [2002], we estimate the accuracy of our optical flow assuming a constant intensity along motion trajectories. This is done by comparing the mean color on a 3×3 neighborhood of a pixel between two frames. In practice we use the L_2 norm of the distance in RGB.

5 Results and discussion

Following Bousseau et al. [2006], we produce the final watercolor rendering as the composite of the abstracted video with the advected pigmentation texture using Equation (1) (see Figure 8). We also add a similar edge darkening pass on the abstracted video. The spatiotemporal filter we have described can either be used alone or after first applying a bilateral filtering that simplifies colors, as in Winnemoeller et al. [2006].

We have implemented our method as an Adobe After Effects plug-in. Our performance statistics are dependent on the whole system, and our algorithms have not been optimized. Currently, advection takes 5 seconds per frame and morphological filtering about 30. The advection computation could be sped up considerably using graphics hardware, as the current bottlenecks are mainly texture access and bilinear interpolation. Morphological filtering operations are notoriously expensive. However, some of the clever implementation strategies devised for 2D processing [Weiss 2006] may be generalizable to 3D, which could greatly improve the performance.

Our results are best viewed in the accompanying videos.[3] The "cactus" sequence shows how our new advection scheme handles strong motion boundaries. The "waves" sequence gives an example of complex motions that are well depicted by the advection, without introducing distortions.

In all of the results, the quality of the texture advection is limited by the quality of the optical flow. Errors in the optical flow lead to "swimming" artifacts in the pigment texture. Such errors are especially visible in relatively unstructured areas of the scene, and near occlusion boundaries. Other kind of motion representations, such as motion layers [Wang and Adelson 1994] could correct these artifacts in some situations, but we believe that the vector field representation can handle motions that other representations cannot (e.g., zoom or large parallax on the same object). We have also applied our advection scheme on a computer-generated sequence, which shows how the advection can accurately depict complex motions and occlusions giving a correct motion flow, as shown in the accompanying video material.

As far as abstraction is concerned, our filtering is stable over time and offers large color regions that appear and disappear gradually over the animation. All the examples presented in our video have been abstracted with a $7 \times 7 \times 9$ structuring element. Increasing the spatial extent of the structuring element generally requires increasing its temporal extent as well, in order to allow time for the width of the structuring element to increase and decrease gradually.

[3] Videos are available on the project webpage
http://artis.imag.fr/Publications/2007/BNTS07

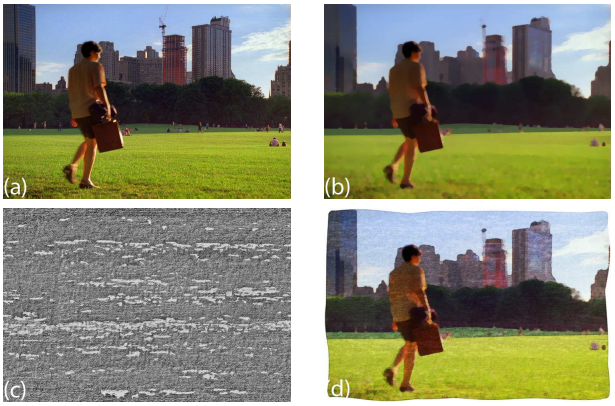


Figure 8: Watercolor compositing. *The final watercolor rendering* (d) *is obtained from an input video* (a) *as the composite of the abstracted video* (b) *with the advected pigmentation texture* (c)*, using Equation (1).*

Figure 9: Final results. *Note the absence of distortions, even in the presence of motion boudaries* (left) *or complex motions* (right).

6 Conclusions

We have presented an approach to video watercolorization that provides good temporal coherence given good optical flow. By combining texture advection and mathematical morphology operations, and by extending them to handle the types of distortions commonly found in video sequences, we are able to produce computer-generated watercolor animations that are convincing for most scenes.

We see a number of areas for future work. First, we would like to investigate whether motion estimation methods based on a post-optimization of the optical flow, such as Particle Video [Sand and Teller 2006], could be used to obtain a better vector field from which we could perform our texture advection. It might also be worthwhile to explore other domains in which high-quality optical flow fields may be more readily available, such as watercolorizations of 3D pans and zooms across real-world scenes derived using structure-from-motion estimations [Horn 1986].

We would also like to look at speeding up our computations, perhaps through a GPU implementation of our texture advection approach, or adapting techniques for optimizing morphological filtering to 3D [Weiss 2006].

In addition, we would like to extend this work to other styles of illustration beyond watercolor, such as charcoal or pastel. In our experiments so far, we have found that the more "structured" the appearance of the medium's effects, the more objectionable any distortions due to incorrect optical flow appear. However, these artifacts might be ameliorated by decreasing the frame rate, and/or alternating between different versions of texture, as in, for example, much of Bill Plympton's animation.[4]

Finally, we are interested in seeing how our extensions could now be re-adapted to improve other domains. Our new advection method is applicable to scientific flow visualization, thus overcoming some

[4] http://www.plymptoons.com/

limitations of previous methods. Similarly, our new abstraction method can be used as a pre-processing step for any type of video stylization, improving existing stroke-based rendering techniques.

Acknowledgments

All the videos used in this work are from the Artbeats Digital Film Library. Thanks to Laurence Boissieux for the computer-generated sequence, to Mira Dontcheva for her help on the submission video, and to David Simons, Dan Wilk, and Bruce Bullis for their help on Adobe After Effects™. Thanks also to the anonymous reviewers for their constructive comments.

References

ALP, B., HAAVISTO, P., JARSKE, T., OISTAMO, K., AND NEUVO, Y. A. 1990. Median-based algorithms for image sequence processing. In *SPIE Vol. 1360, Visual Communications and Image Processing*, 122–134.

BOUSSEAU, A., KAPLAN, M., THOLLOT, J., AND SILLION, F. X. 2006. Interactive watercolor rendering with temporal coherence and abstraction. In *International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, 141 – 149.

CHUANG, Y.-Y., AGARWALA, A., CURLESS, B., SALESIN, D. H., AND SZELISKI, R. 2002. Video matting of complex scenes. *ACM Transactions on Graphics (Proc. SIGGRAPH 2005) 21*, 3 (July), 243–248.

COLLOMOSSE, J. P., ROWNTREE, D., AND HALL, P. M. 2005. Stroke surfaces: Temporally coherent artistic animations from video. *IEEE Transactions on Visualization and Computer Graphics 11*, 5 (Sept.), 540–549.

CUNZI, M., THOLLOT, J., PARIS, S., DEBUNNE, G., GASCUEL, J.-D., AND DURAND, F. 2003. Dynamic canvas for immersive non-photorealistic walkthroughs. In *Graphics Interface*, 121–130.

CURTIS, C. J., ANDERSON, S. E., SEIMS, J. E., FLEISCHER, K. W., AND SALESIN, D. H. 1997. Computer-generated watercolor. In *SIGGRAPH 97*, 421–430.

FANG, H. 2006. Rototexture: Automated tools for texturing raw video. *IEEE Transactions on Visualization and Computer Graphics 12*, 6, 1580–1589.

HARALICK, R. M., STERNBERG, S. R., AND ZHUANG, X. 1987. Image analysis using mathematical morphology. *IEEE Trans. Pattern Anal. Mach. Intell. 9*, 4, 532–550.

HAYS, J., AND ESSA, I. 2004. Image and video based painterly animation. In *International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, 113–120.

HERTZMANN, A., AND PERLIN, K. 2000. Painterly rendering for video and interaction. In *International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, 7–12.

HORN, B. K. P. 1986. *Robot Vision*. MIT Press. ISBN 0-262-08159-8.

JOBARD, B., ERLEBACHER, G., AND HUSSAINI, M. Y. 2001. Lagrangian-eulerian advection for unsteady flow visualization. In *VIS '01: Conference on Visualization '01*, 53–60.

JOHAN, H., HASHIMOTA, R., AND NISHITA, T. 2005. Creating watercolor style images taking into account painting techniques. *Journal of the Society for Art and Science 3*, 4, 207–215.

KLEIN, A. W., SLOAN, P.-P. J., FINKELSTEIN, A., AND COHEN, M. F. 2002. Stylized video cubes. In *ACM-SIGGRAPH/EG Symposium on Computer Animation (SCA)*, 15–22.

KOKARAM, A. C. 1998. *Motion Picture Restoration: Digital Algorithms for Artefact Suppression in Degraded Motion Picture Film and Video*. Springer-Verlag. ISBN 3-540-76040-7.

LAVEAU, N., AND BERNARD, C. 2005. Structuring elements following the optical flow. In *Mathematical Morphology: 40 Years On, Proc. of the 7th International Symposium on Mathematical Morphology*, Springer-Verlag, Ed., 43–52.

LEI, E., AND CHANG, C.-F. 2004. Real-time rendering of watercolor effects for virtual environments. In *IEEE 2004 Pacific-Rim Conference on Multimedia*, 474–481.

LITWINOWICZ, P. C. 1997. Processing images and video for an impressionist effect. In *SIGGRAPH 97*, 407–414.

LUFT, T., AND DEUSSEN, O. 2006. Real-time watercolor illustrations of plants using a blurred depth test. In *International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, 11–20.

LUM, E. B., AND MA, K.-L. 2001. Non-photorealistic rendering using watercolor inspired textures and illumination. In *Pacific Graphics*, 322–331.

MAX, N., AND BECKER, B. 1995. Flow visualization using moving textures. In *Proc. of the ICASW/LaRC Symposium on Visualizing Time-Varying Data*, 77–87.

NEYRET, F. 2003. Advected textures. In *ACM-SIGGRAPH/EG Symposium on Computer Animation (SCA)*, 147 – 153.

OZKAN, M. K., SEZAN, M. I., AND TEKALP, A. M. 1993. Adaptive motion-compensated filtering of noisy image sequences. *IEEE transactions on circuits and systems for video technology 3*, 4, 277–290.

SAND, P., AND TELLER, S. 2006. Particle video: Long-range motion estimation using point trajectories. In *CVPR*, 2195 – 2202.

SERRA, J., AND VINCENT, L. 1992. An overview of morphological filtering. *Circuits Syst. Signal Process. 11*, 1, 47–108.

SIMS, K. 1992. Choreographed image flow. *The Journal of Visualization and Computer Animation 3*, 1, 31–43.

SMALL, D. 1991. Modeling watercolor by simulating diffusion, pigment, and paper fibers. In *SPIE*, vol. 1460, 140–146.

STAM, J. 1999. Stable fluids. In *SIGGRAPH 99*, 121–128.

VAN LAERHOVEN, T., LIESENBORGS, J., AND VAN REETH, F. 2004. Real-time watercolor painting on a distributed paper model. In *Computer Graphics International*, 640–643.

WANG, J. Y. A., AND ADELSON, E. H. 1994. Representing Moving Images with Layers. *The IEEE Transaction on Image Processing Special Issue: Image Sequence Compression 3*, 5, 625–638.

WANG, J., XU, Y., SHUM, H.-Y., AND COHEN, M. 2004. Video tooning. *ACM Transactions on Graphics (proc. of SIGGRAPH 2004) 23*, 3, 574 – 583.

WEISS, B. 2006. Fast median and bilateral filtering. *ACM Transactions on Graphics (proc. of SIGGRAPH 2006) 25*, 3, 519–526.

WINNEMOLLER, H., OLSEN, S. C., AND GOOCH, B. 2006. Real-time video abstraction. *ACM Transactions on Graphics (proc. of SIGGRAPH 2006) 25*, 3, 1221 – 1226.

Dynamic 2D Patterns for Shading 3D Scenes

Simon Breslav, Karol Szerszen, Lee Markosian
University of Michigan

Pascal Barla, Joëlle Thollot
INRIA Grenoble University

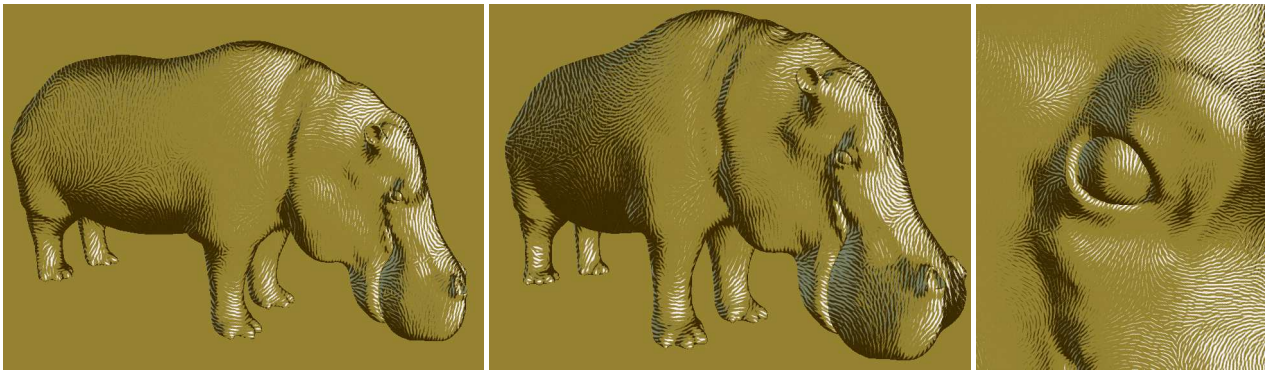


Figure 1: *A dynamic 2D pattern that scales, rotates and translates in 2D to best match the apparent motion of the underlying 3D shape.*

Abstract

We describe a new way to render 3D scenes in a variety of non-photorealistic styles, based on patterns whose structure and motion are defined in 2D. In doing so, we sacrifice the ability of patterns that wrap onto 3D surfaces to convey shape through their structure and motion. In return, we gain several advantages, chiefly that 2D patterns are more visually abstract – a quality often sought by artists, which explains their widespread use in hand-drawn images.

Extending such styles to 3D graphics presents a challenge: how should a 2D pattern move? Our solution is to transform it each frame by a 2D similarity transform that closely follows the underlying 3D shape. The resulting motion is often surprisingly effective, and has a striking cartoon quality that matches the visual style.

1 Introduction

A variety of methods have been proposed for rendering 3D scenes in non-photorealistic styles based on patterns of brush strokes or other 2D marks. These methods are typically designed to produce images that resemble traditional drawings, prints, and paintings in a range of styles. Examples include the work of Winkenbach and Salesin [1994], Meier [1996], Hertzmann and Zorin [2000], Webb *et al.* [2002], Freudenberg *et al.* [2002], and Kalnins *et al.* [2002].

In nearly all such work to date, the 2D marks are organized to follow the features of the 3D shapes that make up the scene. This helps convey a better sense of 3D shape to the viewer [Girshick et al. 2000]. Indeed, the same strategy is used as well in many hand-drawn images: strokes are often aligned with features of the shapes being depicted.

In quite a few cases, however, artists choose a different strategy – stroke directions appear independent of the underlying 3D shapes (see for example Figure 2).

Such "2D patterns" of strokes are popular because they provide greater visual abstraction. Visual abstraction takes many forms, and is of fundamental importance in non-photorealistic images because it lets the artist "stress the essential rather than the particular" [Brassard 1998]. Depending on the chosen style, better visualization of the details of a shape is not always desirable. This is especially true for less-important objects such as those in the background.

Given the prevalence of 2D patterns in hand-drawn images, it is natural to consider using them to render animated 3D scenes with computers. This presents a challenge: how should the patterns move? A trivial solution is to keep the pattern fixed in the image, but this leads to a disturbing disconnect between the movement of the pattern and the underlying object – a condition known as the "shower door" effect. On the other hand, any motion that exactly matches the image-space motion of the 3D object will result in distortions in the shape of the 2D pattern.

Our solution is to compute a shape-preserving 2D "similarity transform" combining translation, rotation, and uniform scaling to match as closely as possible the apparent motion of the surface (see Figure 1 and the accompanying video). The scene can be divided into separate objects, or patches within an object, with a separate similarity transform computed for each object or patch.

We describe a working system based on these dynamic 2D patterns to achieve a variety of styles: multi-color halftoning, hatching and stippling, and a painterly style. We address LOD transitions that prevent the 2D pattern from growing too large or small. The computation of the similarity transform is straightforward and efficient, and the system leverages programmable GPUs to render at interactive rates.

Transformed patterns can still exhibit sliding, but the method works surprisingly well for many motions, and the 2D movement has a striking cartoon quality that can match the visual style.

Figure 2: *Left to right: details from drawings by Joe Sacco, Bill Plympton and Richard Sala. Stroke patterns in many hand-drawn images like these are laid out in image space and do not follow the 3D shapes depicted by them.*

2 Related work

The "dynamic canvas" method of Cunzi *et al.* [2003] addresses a closely related problem and proposes a similar solution. The problem is how to reduce the shower door effect by transforming the background "canvas" texture used for media simulation by many 3D NPR algorithms. Their solution is to apply a 2D similarity transform that matches the image-space motion of certain 3D points in the scene.

The method has two limitations: (1) it works best in the limited case of a static scene in which all objects lie at roughly the same distance from the camera; and (2) that distance is a parameter that must be supplied by the application. In contrast, our method is automatic and handles both camera and object motion (including animated objects). It can be applied independently to each object (or part of an object) in the scene.

Bousseau *et al.* [2006] and Kaplan and Cohen [2005] also describe methods for achieving a temporally coherent dynamic canvas. These methods are automatic and account for camera motion as well as objects that move independently. Both methods track seed points on 3D surfaces and use them to generate (each frame) a canvas texture by joining small pieces of texture that follow the seed points. These methods work well with small-scale, unstructured patterns like canvas and paper textures, but cannot preserve regular or large-scale patterns like hatching and halftone screens. Eissele *et al.* [2004] propose a similar strategy for 2D halftone patterns applied to 3D scenes. Not surprisingly, the method exhibits severe temporal artifacts, with the halftone screen continually breaking apart and re-forming.

Coconu *et al.* [2006] describe a method for applying 2D hatching patterns to 3D scenes, particularly landscape models containing trees. Leaves of the trees are clustered into "high level primitives" and rendered with 2D hatching patterns that are updated each frame using a 2D similarity transform. This method is similar to the one we propose, except they compute the similarity transform by tracking a *single* 3D sample point and orientation vector each frame, whereas we use a collection of 3D sample points and a weighted least-squares optimization to compute the similarity transform that best matches the observed motion of the 3D points. We explain the details in the next section.

3 Transforming 2D patterns

We implemented several styles of "dynamic 2D patterns" in GLSL [Rost 2006]. Our halftone shader uses a previously published halftone method [Ostromoukhov 1999; Durand et al. 2001; Freudenberg et al. 2002] to generate several color layers that follow separate light sources (see for example Figure 1). We also demonstrate hatching and painterly styles in the accompanying video. Each style uses image-space texture maps to encode halftone screens, paper textures ([Kalnins et al. 2002]), or collections of hatching or paint strokes. These "2D patterns" are typically combined in several color layers to convey shading and highlights.

We transform 2D patterns as follows. In a pre-process we distribute 3D sample points over surfaces in the scene. At run time, we use the image-space positions of the samples in the current and previous frame to compute a 2D similarity transform that closely follows their observed motion. We then apply the transform to the pattern. We explain the details in the following sections.

3.1 Samples and weights

We generate sample points using the "stratified point sampling" method of Nehab and Shilane [2004]. This method achieves a relatively uniform distribution of points over each 3D surface, independent of triangulation. 3D samples are stored using barycentric coordinates relative to mesh triangles, so they remain valid when the mesh is transformed or animated.

To reduce the chance of finding too few sample points in any frame, we use a moderately dense set of samples for each object or patch (typically several hundred). Though this is far more than needed to produce a good solution, the performance cost of using additional samples is negligible, as the computation of the similarity transform is linear in the number of samples. The sampling density can be adjusted by the user, though in practice we found that the default settings work well.

Each sample is assigned a weight each frame based on approximately how much of the visible image is taken up by the bit of surface associated with the sample. To achieve this, we compute the image-space area of a disk that lies tangent to the surface at the sample location, with diameter equal to the sample spacing. We set the weight equal to this value times a "fuzzy" visibility measure of the sample, as in the "blurred depth test" of Luft and Deussen [2006], which we implemented using an "ID image" [Kalnins et al. 2002] instead of a depth-buffer. We compare this weighting scheme to a simpler one in the accomanying video and Section 4.

We handle multiple objects (or patches within an object) by considering the sample points of each object (or patch) independently. When multiple patches are used, patterns can be overlapped and blended across patch boundaries to hide the discontinuities, as explained in Section 3.6.

3.2 Least-squares solution

We use Horn's method [Horn 1987] to compute the 2D similarity transform (combining translation, rotation, and uniform scaling) that best maps the sample locations in the previous frame to the current one. Horn's paper provides in-depth derivations, but note that our 2D case is simpler than the 3D case addressed by Horn.

Below, w_i denotes the weight assigned to sample i in the current frame (see Section 3.1), but we use $w_i = 0$ when the weight assigned in the previous frame was 0. This way, we only consider samples that are visible in both the current and previous frames.

Let $\bar{\mathbf{c}}$ and $\bar{\mathbf{p}}$ denote the weighted centroids of the image-space locations of the samples in the current and previous frames, respectively (both using weights w_i). The translation is then simply $\bar{\mathbf{c}} - \bar{\mathbf{p}}$.

Let $\mathbf{c}_i$ denote the image-space location of sample i in the current frame minus $\bar{\mathbf{c}}$. Define $\mathbf{p}_i$ similarly as the location of sample i in the previous frame minus $\bar{\mathbf{p}}$.

We seek the combination of 2D rotation and uniform scaling that best maps $\mathbf{p}_i$ to $\mathbf{c}_i$, taking into account weights w_i. Note that any 2D point or vector can be viewed as a complex number – the notations $x+iy$ and (x,y) are equivalent – and that complex number multiplication achieves 2D rotation and uniform scaling.[1] We can thus formulate the problem as a weighted least squares optimization seeking the complex number $\mathbf{z}$ that minimizes the error:

$$E = \sum_i w_i |\mathbf{z}\mathbf{p}_i - \mathbf{c}_i|^2.$$

The least-squares solution is found by taking partial derivatives with respect to the two unknown components of $\mathbf{z}$, setting equal to 0, and solving. This yields:

$$\mathbf{z} = \left(\sum_i w_i \mathbf{p}_i \cdot \mathbf{c}_i, \quad \sum_i w_i \mathbf{p}_i \times \mathbf{c}_i \right) / \sum_i w_i |\mathbf{p}_i|^2$$

(Here, "$\times$" denotes the "2D cross product.")

As Horn [1987] explains, the above formulation is not symmetric, in the sense that the computed transform from $\mathbf{c}_i$ to $\mathbf{p}_i$ is not the inverse of the transform from $\mathbf{p}_i$ to $\mathbf{c}_i$ — in general, the two rotations are inverses, but the two scale factors are not. We thus adopt the "symmetric" formulation described by Horn: we multiply $\mathbf{z}$ as computed above by the following scale factor s:

$$s = |\mathbf{z}|^{-1} \left(\frac{\sum_i w_i |\mathbf{c}_i|^2}{\sum_i w_i |\mathbf{p}_i|^2} \right)^{\frac{1}{2}}$$

Without this correction, repeated zooming in and out can gradually shrink the pattern, which may be undesirable. In any case, when the cumulative scaling applied to the pattern grows too large (or small), we initiate a transition to a less-scaled version of the pattern, as explained in Section 3.4.

[1] Multiplication by a complex number $\mathbf{z}$ has a geometric interpretation: it scales by $|\mathbf{z}|$ and rotates by the angle between $(1,0)$ and $\mathbf{z}$ (hence $i^2 = -1$).

3.3 Computing texture coordinates

The texture maps used by our shaders are defined in a canonical *uv*-space. By convention, the texture fills the unit square $[0,1]x[0,1]$, and we assume it tiles seamlessly to fill all of *uv*-space. For each separately moving pattern we store an image-space point $\mathbf{o}$ corresponding to the origin in *uv*-space, and image-space vectors $\mathbf{u}$ and $\mathbf{v}$ that correspond to *uv* vectors $(1,0)$ and $(0,1)$, respectively. In other words, $\mathbf{o}$ locates the lower left corner of the pattern in the image, and $\mathbf{u}$ and $\mathbf{v}$ determine its horizontal and vertical edges.

In each frame we compute $\bar{\mathbf{p}}$, $\bar{\mathbf{c}}$ and $\mathbf{z}$ as explained in Section 3.2, then update $\mathbf{o}$, $\mathbf{u}$ and $\mathbf{v}$ as follows:

$$\begin{aligned} \mathbf{o} &\leftarrow \bar{\mathbf{c}} + \mathbf{z}(\mathbf{o} - \bar{\mathbf{p}}) \\ \mathbf{u} &\leftarrow \mathbf{z}\mathbf{u} \\ \mathbf{v} &\leftarrow \mathbf{z}\mathbf{v} \end{aligned}$$

In the fragment shader, *uv*-coordinates are computed from the fragment location $\mathbf{q}$ using: $((\mathbf{q}-\mathbf{o})\cdot\mathbf{u}/|\mathbf{u}|^2, (\mathbf{q}-\mathbf{o})\cdot\mathbf{v}/|\mathbf{v}|^2)$. These coordinates can be used with any 2D pattern, to make it track the apparent motion of the object in image-space.

3.4 LOD transitions

When patterns are scaled very large or small they lose their original appearance. We support a kind of "level of detail" (LOD) whereby patterns are restored to nearly their original size when the cumulative scale factor falls outside of a given range. We set two scaling thresholds $1 < s_0 < s_1 < 2$. (In our examples, we used $s_0 = 1.3$ and $s_1 = 1.7$.) As cumulative scaling increases from 1 to s_0, the pattern grows larger. As scaling increases from s_0 to s_1, the pattern continues growing but fades out, while a half-size copy of the pattern fades in. We thus transition from a somewhat too-large pattern at scale factor s_0 to a somewhat too-small pattern at scale factor s_1 (since the scale factor is applied to a half-sized pattern). As the underlying scale factor continues to increase from s_1 to 2, the pattern grows back to its normal size. We combine hatching and painterly patterns by alpha blending to achieve a cross-fade; for halftone patterns we blend between halftone *screens*, causing the pattern to morph instead of fade.

3.5 Tone correction

The blended halftone screens used for LOD transitions generally don't reproduce tones exactly as the original (scaled or unscaled) screen, so to avoid tone fluctuations during LOD transitions, we use a tone correction method like the one described by Durand *et al.* [2001]. The idea is to take into account how tones will be altered by the halftone process when using a given screen, and compensate by adjusting the input tone in order to produce the desired output tone. In the case addressed by Durand *et al.*, the tone alteration function took a known analytical form that could be inverted. In our case we do not have an analytic expression for the way tones are altered by the given halftone screen, so we tabulate it in a pre-process by applying the halftone screen to each possible input tone value and measuring the result, a strategy like the one used by Salisbury *et al.* [1997] for a similar purpose.

The inverse of the tone alteration function for each halftone screen can be stored as a 1D texture (size $256x1$) that is used in the fragment shader to remap tone values before using them in the halftone process with that screen. For LOD transitions, we compute these 1D textures at regular samples of the LOD transition parameter (we use 16 samples), resulting in a 2D texture (size $256x16$) needed for each halftone screen.

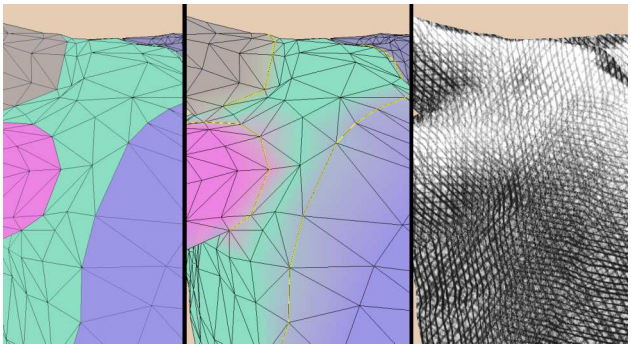


Figure 3: *Left: detail of several patches from the cow model. Middle: visualization of blending weights over patch 1-rings. Right: hatching textures blended across patch boundaries.*

3.6 Multiple patches

Our method greatly reduces the shower door effect compared to patterns that remain fixed in the image, but for some combinations of object shape and camera motions, significant sliding can still occur. To reduce sliding, the user can divide a given mesh into multiple patches, each to be rendered separately, using a similarity transform computed with samples restricted to that patch. This can reduce sliding significantly, as demonstrated in the video.

This introduces a visible discontinuity in the appearance and motion of patterns on either side of the boundary. We address that problem using a simple method for blending patterns across boundaries that effectively hides these discontinuities in many cases – especially for irregular patterns that use alpha blending, such as hatching and painterly styles. (See Figures 3 and 5.)

We blend patterns near patch boundaries using a set of blending weights – at each vertex we store one weight per patch. Weights are initially set to 1 within a patch, and to $1/k$ at boundary vertices, where k is the number of adjacent patches. To render a patch, we render all of its triangles plus those in the 1-ring outside the patch. The "strength" of the pattern is determined by the blending weight at each vertex – a low weight results in a faintly rendered pattern.

With weights initialized as described above, each pattern is drawn at half strength along its boundary, with strength falling to 0 over the outer 1-ring of the patch boundary and rising to 1 over the inner 1-ring. Where two or more patches abut, their patterns cross-fade over the 1-ring of the patch boundaries. Depending on the mesh resolution, we might want to blend patterns across wider regions. We thus let the user specify (via a slider in our GUI) the value n of the n-ring size used in patch blending. We use an iterative process to compute smoothly varying weights, as follows.

For $n-1$ iterations, we visit each vertex and, for each patch, set its weight at that vertex equal to the average of its weights over the neighboring vertices. (We compute all the new weights first, then assign them.) At each iteration, once all patch weights are computed, they are normalized so the sum of the weights stored at each vertex is 1. With this scheme, patch weights remain near 0.5 along patch boundaries, but smoothly fall to zero near the outer n-ring of the patch boundary.

Even with weights that sum to 1 at each vertex, the final tone conveyed by overlapping patterns can be altered in regions where patches are blended. (E.g., in a dark region requiring 100% ink application, using two halftone patterns that each outputs 50% ink will not achieve 100% ink coverage due to overlap in the patterns.)

We thus provide an additional control through which the user can increase or decrease the strength of the pattern around its boundaries. In the pixel shader, before the weight is used, it is raised to a power specified by the user – e.g., using a power of 0.5 increases the strength of the pattern moderately within the n-ring of the boundary. We often used this value in practice.

This method of computing weights and using them to blend between patches where they overlap effectively hides discontinuities between patches in many cases, and lets the user control the width of the overlap region. A drawback is that it depends on the mesh having a reasonably regular triangulation. More sophisticated schemes could be tried when that is not the case.

4 Results and discussion

The accompanying video shows our method in action. The results appear quite natural for camera or object motions that are well-approximated by a series of 2D similarity transforms. Such motions include camera pan or zoom, or camera rotation around the camera's forward line of sight.

The method produces mixed results for a single object rotating in front of the camera. For a compact object like a sphere, the 2D translation produced by the method is reasonable, since visible surfaces largely appear to translate in a consistent direction in 2D. The worst case (for static scenes) is when an elongated object rotates around an axis that is parallel to the film plane (see Figure 4). Looking straight on from the side, opposite ends of the object appear to move toward the center line. No 2D similarity transform well-approximates the apparent motion in this case.

We observed that while sliding can be a considerable problem for close examination of a single elongated shape, the problem is reduced when multiple objects are arranged in a scene (such as the landscape with cows, or the row of skulls shown in the video). Apparently the reason is that when navigating around several objects placed side by side, it is natural to move more slowly and gradually, compared to navigating around a single object. Sliding still occurs, but when sufficiently slowed, it is less noticeable.

Our method of weighting samples is designed to give higher priority to portions of surface that occupy more of the image. For comparison, we also implemented a simpler "unweighted" scheme that assigns weight = 1 if the sample is in the view frustum and front-facing, 0 otherwise. The results are generally similar, but the weighted scheme can be noticeably better, as in the case of the table shown in the video. In that example, the table top and support occupy little space in the image, and so are assigned small weights by the weighted scheme. That results in a better transform,

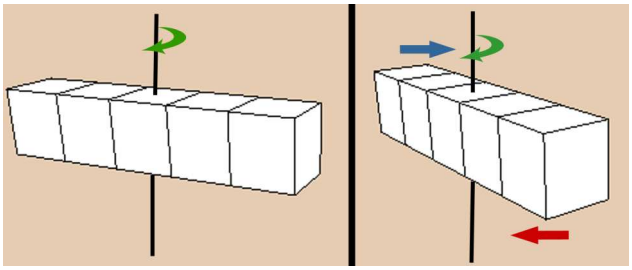


Figure 4: *Worst case scenario: an elongated object rotating around an axis parallel to the film plane. Opposite ends move toward the center line. Our method works poorly in this case since no 2D similarity transform well-approximates the apparent motion.*

Figure 5: *A painterly style applied to an animated model of a hand that has been divided into patches.*

because those surfaces move differently than the side of the table cloth, which fills most of the image and has a consistent motion.

We also observed a case when the weighted scheme seems worse than the unweighted one. The landscape model shown in the video has a large featureless foreground and a few hills on the horizon. The unweighted scheme assigns more importance to the distant hills, while the weighted scheme favors the foreground. Since the foreground lacks clear features, the viewer is less aware of sliding there, while the hills have a clearly perceived location, and any sliding of the pattern across them is quite noticeable. An interesting avenue for future work is thus to investigate weighting schemes that take into account more perceptual aspects of the scene.

A benefit of using 2D patterns is that they require no tedious surface parameterization, which might save time for artists. Artists are more free to design any pattern they want since patterns are ensured not to be deformed. Designs can easily be transferred to new objects. 2D patterns may also be better suitable for integration in hand-drawn animations.

We don't claim that 2D patterns should always be used in place of patterns that are mapped onto surfaces in 3D, especially in the case of foreground objects at the center of attention. Rather, we argue that 2D patterns will be attractive to designers and animators because they are easy to create, they perform well for many motions, and they provide a greater degree of visual abstraction. We envision interactive applications and animations that combine dynamic 2D patterns with other rendering techniques to produce both 2D and 3D effects with a unified artistic style.

References

BOUSSEAU, A., KAPLAN, M., THOLLOT, J., AND SILLION, F. 2006. Interactive watercolor rendering with temporal coherence and abstraction. In *NPAR 2006*, 141–149.

BRASSARD, L. 1998. *The Perception of the Image World.* PhD thesis, Simon Fraser University.

COCONU, L., DEUSSEN, O., AND HEGE, H.-C. 2006. Real-time pen-and-ink illustration of landscapes. In *NPAR 2006*, 27–35.

CUNZI, M., THOLLOT, J., PARIS, S., DEBUNNE, G., GASCUEL, J.-D., AND DURAND, F. 2003. Dynamic canvas for immersive non-photorealistic walkthroughs. In *Proc. Graphics Interface*.

DURAND, F., OSTROMOUKHOV, V., MILLER, M., DURANLEAU, F., AND DORSEY, J. 2001. Decoupling strokes and high-level attributes for interactive traditional drawing. In *12th Eurographics Workshop on Rendering*, 71–82.

EISSELE, M., WEISKOPF, D., AND ERTL, T. 2004. Frame-to-Frame Coherent Halftoning in Image Space. In *Proceedings of Theory and Practice of Computer Graphics 2004*, 188–195.

FREUDENBERG, B., MASUCH, M., AND STROTHOTTE, T. 2002. Real-time halftoning: a primitive for non-photorealistic shading. In *13th Eurographics Workshop on Rendering*, 227–232.

GIRSHICK, A., INTERRANTE, V., HAKER, S., AND LEMOINE, T. 2000. Line direction matters: An argument for the use of principal directions in 3D line drawings. In *NPAR 2000*, 43–52.

HERTZMANN, A., AND ZORIN, D. 2000. Illustrating smooth surfaces. In *SIGGRAPH 2000*, 517–526.

HORN, B. K. P. 1987. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America 4*, 4 (April), 629–642.

KALNINS, R. D., MARKOSIAN, L., MEIER, B. J., KOWALSKI, M. A., LEE, J. C., DAVIDSON, P. L., WEBB, M., HUGHES, J. F., AND FINKELSTEIN, A. 2002. WYSIWYG NPR: Drawing strokes directly on 3D models. *ACM Transactions on Graphics 21*, 3, 755–762.

KAPLAN, M., AND COHEN, E. 2005. A generative model for dynamic canvas motion. In *Proceedings of Computational Aesthetics in Graphics, Visualization and Imaging*, 49–56.

LUFT, T., AND DEUSSEN, O. 2006. Real-time watercolor illustrations of plants using a blurred depth test. In *NPAR 2006*, 11–20.

MEIER, B. J. 1996. Painterly rendering for animation. In *SIGGRAPH 96*, 477–484.

NEHAB, D., AND SHILANE, P. 2004. Stratified point sampling of 3D models. In *Eurographics Symposium on Point-Based Graphics*, 49–56.

OSTROMOUKHOV, V. 1999. Digital facial engraving. In *SIGGRAPH 99*, 417–424.

ROST, R. J. 2006. *OpenGL Shading Language, Second Edition*. Addison Wesley Professional.

SALISBURY, M. P., WONG, M. T., HUGHES, J. F., AND SALESIN, D. H. 1997. Orientable textures for image-based pen-and-ink illustration. In *SIGGRAPH 97*, 401–406.

WEBB, M., PRAUN, E., FINKELSTEIN, A., AND HOPPE, H. 2002. Fine tone control in hardware hatching. In *NPAR 2002*, 53–58.

WINKENBACH, G., AND SALESIN, D. H. 1994. Computer-generated pen-and-ink illustration. In *SIGGRAPH 94*, 91–100.

Acknowledgements

We thank Joe Sacco, Bill Plympton, and Richard Sala for permission to use details from their drawings in Figure 2, Igor Guskov for helpful advice in the early stages of this project, Haixiong Wang, Mike Cook and Rob Martinez for coding support, and the reviewers at INRIA and anonymous reviewers whose advice helped improve the paper. This research was supported in part by the NSF (CCF-0447883).

EUROGRAPHICS 2008 / G. Drettakis and R. Scopigno
(Guest Editors)

Volume 27 (2008), Number 2

Apparent Greyscale: A Simple and Fast Conversion to Perceptually Accurate Images and Video

Kaleigh Smith[1] Pierre-Edouard Landes[2] Joëlle Thollot[2] Karol Myszkowski[1]

[1]MPI Informatik, Saarbrücken, Germany [2]Grenoble Universities, INRIA, France

Abstract

This paper presents a quick and simple method for converting complex images and video to perceptually accurate greyscale versions. We use a two-step approach first to globally assign grey values and determine colour ordering, then second, to locally enhance the greyscale to reproduce the original contrast. Our global mapping is image independent and incorporates the Helmholtz-Kohlrausch colour appearance effect for predicting differences between isoluminant colours. Our multiscale local contrast enhancement reintroduces lost discontinuities only in regions that insufficiently represent original chromatic contrast. All operations are restricted so that they preserve the overall image appearance, lightness range and differences, colour ordering, and spatial details, resulting in perceptually accurate achromatic reproductions of the colour original.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Display Algorithms

1. Introduction

The basic problem of greyscale transformation is to reproduce the intent of the colour original, its contrasts and salient features, while preserving the perceived magnitude and direction of its gradients. The transformation consists of two interdependent tasks: a mapping that assigns a grey value to each pixel or colour, and a discriminability constraint so that the achromatic differences match their corresponding original colour differences. Recent approaches solve discriminability constraints to determine the grey values, producing images in which the original colour contrasts are highly discriminable. However, the greyscale images may exhibit exaggerated dynamic range, an arbitrary achromatic order that differs among colour palettes, and a smoothing or masking of details. These modifications all contribute to make the grey version appear dissimilar from its original and create inconsistency among like images and video frames.

The goal of this work is to create a perceptually accurate version of the colour image that represents its psychophysical effect on a viewer. Such greyscale imagery is important for printed textbooks and catalogues, the stylization of videos and for display on monochromatic medical displays. A perceptually accurate image is one that emulates both global and local impressions: it matches the original values' range and average luminance, its local contrasts are neither exaggerated nor understated, its grey values are ordered according to colour appearance and differences in spatial details are imperceptible. Strong perceptual similarity is particularly important for consistency over varying palettes and temporal coherence for animations.

We present a new two-step greyscale transformation that combines a global mapping based on perceived lightness with a local chromatic contrast enhancement. Our simple algorithm yields comparable images to more complex approaches, and its linear runtime makes it suited to video processing and accelerated graphics hardware implementations. First, the grey values are mapped pixelwise from each colour's apparent lightness, resulting in the reproduction of the original contrast and gradients. Second, the gradient magnitudes are measured in perceptual difference ΔE and adjusted to maintain or improve discriminability with our multiscale chromatic contrast enhancement filter. This twofold approach mimics aspects of the human visual system, which processes global attributes while simultaneously depending on local contrasts such as edges and surrounds.

We choose *lightness* as the quantity for the grey values (and thus their ordering) because it is the achromatic response to a colour stimulus, measuring how bright a colour appears compared to an equally bright white. Colour studies show that lightness depends largely on luminance, but that colour-

© 2007 The Author(s)
Journal compilation © 2007 The Eurographics Association and Blackwell Publishing Ltd.
Published by Blackwell Publishing, 9600 Garsington Road, Oxford OX4 2DQ, UK and 350 Main Street, Malden, MA 02148, USA.

fulness also contributes, as characterized by the Helmholtz-Kohlrausch effect (H-K); a colourful stimulus appears more light than a similar less colourful sample. The H-K effect has been identified as an important factor in greyscale mapping, and although it has been used for clipart greyscale mapping [BB04], no existing greyscale conversion for complex images explicitly takes it into account. Our *global apparent lightness mapping* is independent of the original colour palette, incorporates the H-K effect so is sensitive to small differences even between isoluminant colours, and yields perceptually accurate gradient directions and an appropriate dynamic range.

The mapping from a 3D to 1D colour space reduces the overall difference between colours, jeopardizing discriminability. We are not too sensitive to this loss when it occurs between spatially distant colours, but with adjacent colours it is immediately apparent, especially if an original contrast becomes imperceptible. To solve this problem, we enhance local contrast until its magnitude emulates that in the original. The enhancement restores chromatic differences without overemphasizing luminance differences by adaptively increasing weak contrasts. Furthermore, the process is restricted so that the polarity over edges, overall lightness and colour ordering are preserved, thus maintaining perceptual accuracy.

We begin by describing other greyscale conversion techniques and highlighting their effects on perceptual accuracy. Then, in Section 3, we give a background on colour appearance models, compare H-K predictors and determine the best suited to greyscale conversion. In Sections 4 and 5, we present our global-local technique to solve greyscale conversion. In Section 6, we compare the results of our technique to other standard results to accentuate the perceptual aspects that are more accurately preserved, and demonstrate our technique's ability on various input types. Finally, we conclude by discussing the implications of our findings, future work and other possible applications.

2. Related Work

There are a variety of printing and display solutions catered to the conversion of images from colour to greyscale. The most straightforward conversion maps a colour to an equiluminant grey value, by desaturation or by picking a single colour channel to mimic the effect of a colour filter.

In their short paper studying chromatic contrast for greyscale conversion, Bala et al. [BE04] take a spatial approach and introduce color contrasts in CIELAB *LCH* (lightness, chroma, hue angle) by adding the high-pass filtered chroma channel to the lightness channel. To prevent overshooting in already bright areas, this correction signal is locally adjusted and its sign is taken from the lightness contrast. The algorithm is susceptible to problems in chroma and lightness misalignment. Taking a local adaptive approach, Bala et al. [BB04] propose a mapping method for business graphics. The distinct colours of the image are sorted according to a simplified lightness predictor that incorporates the H-K effect. To maximize discriminability, adjacent pairs of lightness values are then respaced according to their relative color differences. The approach is uniquely for graphics with up to 10 colours, and is not applicable to complex images.

Gooch et al. [GOTG05] find grey values that best match the original color differences through an objective function minimization process. Original contrast between each pixel and its neighbours is measured by a signed distance, whose magnitude accounts for luminance and chroma difference and whose sign represents the hue shift with respect to a user-defined hue angle. It has $O(N^2)$ to $O(N^4)$ complexity, but a recent extension to a multiresolution framework by Mantiuk et al. improves the algorithm's performance [MMS06]. Rasche et al. [RGW05] propose a similar approach that finds the linear transform matching pairwise grey differences to corresponding color differences. The best transform is found by minimizing an error function that can be evaluated over a smaller set of colors to alleviate computation costs.

In recent work, Grundland et al. [GD07] find a global continuous mapping that adds lost chromatic information to the luminance channel. Their algorithm achieves linear-time performance thanks to Gaussian pairing sampling which limits the amount of processed color differences. In YPQ color space, the color differences are projected onto the two predominant chromatic contrast axes and are then added to the luminance image. A saturation-controlled adjustment of the output dynamic range is adaptively performed to balance between the original range and the desired amount of enhancement. Recently, Neumann et al. [NCN07] present a technique with linear complexity that requires no user intervention. It stresses perceptual loyalty by measuring the image's gradient field by colour differences in their *Coloroid* color space. After discarding all gradient field inconsistencies, fast 2D integration determines the final grayscale image.

In [CF03], Calabria and Fairchild find that image lightness strongly affects perceived contrast, meaning techniques that can arbitrarily modify lightness, like approaches by Rasche and Grundland, may affect image appearance in an adverse way. A greyscale ordering that contradicts the colours' luminance ordering also strongly impacts image appearance, yet in several approaches, ordering is subjective and arbitrary: the choice of hue angle in Gooch's Color2Gray can change all gradient directions, in Rasche's approach a user-defined threshold controls whether a colour is mapped to a darker or lighter value (see Figure 7), and in Grundland's approach ordering depends on the image and parameter choice of the color sampling method. Lastly, image details and salient features may be lost by the choice of neighbourhood size in Gooch's Color2Gray or by unpredictable behavior in inconsistent regions of the gradient field in Neumann's approach

© 2007 The Author(s)
Journal compilation © 2007 The Eurographics Association and Blackwell Publishing Ltd.

(see Figure 9). Since the discussed colour to grey methods depend strongly on local image content, colour palettes and user parameters, they are hindered in their perceptual accuracy and are not directly applicable to animation where the colour palette is frequently modified and pixel correlations change quickly due to occlusion and disocclusion. With this as motivation, we attempt a different approach where perceptual accuracy and consistency is paramount.

3. Apparent Lightness

In order to discuss colour, we must move into the realm where colour exists - that is, in the observer's mind. A physical stimulus produces a perceptual response that we name 'colour', or alternately, an achromatic response we name 'brightness' or 'perceived lightness'. Colour appearance models take on the complex task of predicting a human viewer's perceptual response to colour stimulus - thus defining measures of colour. Throughout this paper, we work in the *CIELAB* and *CIELUV* colour spaces, whose three axes approximate perceived lightness L^*, chroma C^* and hue angle H^*. The first component, L^*, quantifies the perceptual response of a human viewer to luminance and is defined by Hunt as "the brightness of objects relative to that of a similarly illuminated white". Mathematically, it is defined as $L^* = 116(Y/Y_0)^{1/3} - 16$ for luminance Y and reference white luminance Y_0.

While luminance is the dominant contributor to lightness perception, the chromatic component also contributes, and this contribution varies according to both hue and luminance. For example, cornflower blue seems brighter than a dull golden yellow of equal luminance. This phenomenon is characterized by the Helmholtz-Kohlrausch effect, where given two isoluminant colours, the more colourful sample appears brighter.

Helmholtz-Kohlrausch Effect A chromatic stimulus with the same luminance as a white reference stimulus will appear brighter than the reference [Nay97].

There are two experimental approaches for measuring the H-K effect: the Variable-Achromatic-Colour (VAC) approach, in which an achromatic sample's luminance is adjusted to match a colour stimulus; and the Variable-Chromatic-Colour (VCC) approach, in which the chromatic content of a colour stimulus is adjusted until its brightness matches a given grey stimulus [Nay98]. VAC is more common and was used in the seminal 1954 Sanders-Wyszecki study, and again in Wyszecki's later 1964 and 1967 studies [Wys67].

3.1. Helmoltz-Kohlrausch Lightness Predictors

The H-K phenomenon is predicted by a *chromatic lightness term* that corrects L^* based on the colour's chromatic component. We examine three such predictors published by Fairchild and Nayatani for suitability to the greyscale problem. Existing models, like CIECAM02, account for many more complex colour appearance aspects, like surrounding colours, but are less suited to greyscale conversion due to their complexity and because most disregard the Helmholtz-Kohlrausch effect (the reader may refer to Table 17.1 in [Fai05]).

Fairchild's *CIELAB* chromatic lightness metric L^{**} is fit to Wyszecki 1967 data and is defined as [FP91]:

$$L^{**} = L^* + (2.5 - 0.025L^*)\left(0.116\left|sin\left(\frac{H^* - 90}{2}\right)\right| + 0.085\right)C^* \quad (1)$$

Chroma C^* measures colourfulness and a sinusoidal curve predicts the H-K effect's decreased impact at yellow hues and its strongest effect at blues.

Nayatani defines chromatic lightness metrics $L^*_{N_{VAC}}$ and $L^*_{N_{VCC}}$, for each experimental approach, based in CIELUV [Nay97]. † A quantitative difference between them is that $L^*_{N_{VCC}}$ is twice as strong $L^*_{N_{VAC}}$ (in log space). For each method, chromatic object lightness is predicted by the following equations (see Appendix for more details):

$$L^*_{N_{VAC}} = L^* + [-0.1340\ q(\theta) + 0.0872\ K_{Br}]\ s_{uv}\ L^* \quad (2)$$

$$L^*_{N_{VCC}} = L^* + [-0.8660\ q(\theta) + 0.0872\ K_{Br}]\ s_{uv}\ L^* \quad (3)$$

s_{uv} is the chromatic saturation in terms of u',v' which predicts the effect's strength according to colourfulness. The quadrant metric $q(\theta)$ predicts the change of the H-K effect for varying hues, and constant K_{Br} expresses the H-K effect's dependance on the adapting luminance L_a. These chromatic

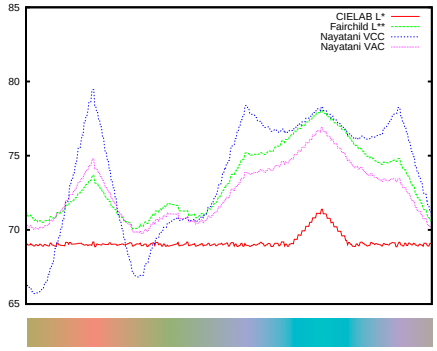


Figure 1: *Lightness values from various H-K effect predictors applied to a spectrum of isoluminant colours, compared to CIE L^*.*

lightness metrics solve a key challenge in greyscale conversion because they predict differences between isoluminant

† For readability, we have used the notation from [Nay98].

© 2007 The Author(s)
Journal compilation © 2007 The Eurographics Association and Blackwell Publishing Ltd.

colours. Figure 1 plots the lightness measured by each metric on a nearly equiluminant colour ramp. It can be seen that more variation occurs when the H-K is being predicted, compared to luminance-based L^* which predicts nearly equal lightness for all colours. Note that other colour pairs will map to the same greyscale value, but that these are predicted to be more similar than the isoluminant colours.

We now decide which predictor is best suited to greyscale conversion. We prefer L^{**} or $L^*_{N_{VAC}}$, because the gathering of VAC data on which they are modeled seems more akin to the goal as it finds a grey that matches a colour. Moreover, in testing $L^*_{N_{VCC}}$, we observe that its stronger effect maps many bright colours to white, making it impossible to distinguish between very bright isoluminant colours [Nay98]. For that reason, and by heeding Nayatani's advice that $L^*_{N_{VAC}}$, instead of $L^*_{N_{VCC}}$, should be used for predicting differences between isoluminant colours, we decide not to use $L^*_{N_{VCC}}$ [Nay98].

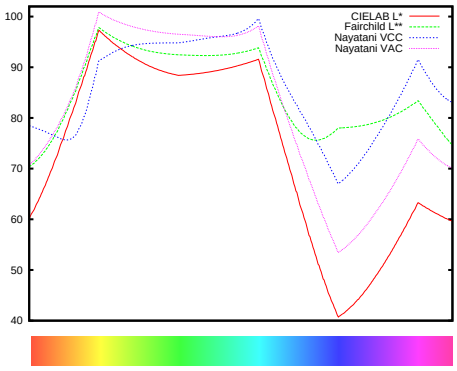


Figure 2: *Lightness values from various H-K effect predictors applied over a full spectrum. L^{**} exhibits a small range and at blue hues differs from L^*.*

Because they are both fit to *VAC* data, the behaviours of L^{**} and $L^*_{N_{VAC}}$ are very similar. Their differences stem from the data on which they are based, and the flexibility of the models. $L^*_{N_{VAC}}$ is based on both Wyszecki 1964 and 1967 data, theoretical arguments about H-K, and the effect of adapting luminance. The L^{**} model is based only on Wyszecki 1967 data and has a simpler treatment of hue which we expect is responsible for the following: we observed L^{**} of blue hues is much higher than L^*, which reduces its range and makes its ordering differ significantly from both $L^*_{N_{VAC}}$ and L^*, see Figure 2. While the model fits the H-K effect perceptual data, this range reduction is problematic for greyscale conversion because colours with different L^* become less discriminable, an observation shared by Bala [BB04].[‡] We therefore conclude that $L^*_{N_{VAC}}$ is the most suitable H-K predictor to use in our global colour to greyscale mapping.

[‡] Bala uses $L^{**}_1 = L^* + 0.143C^*$.

4. Global Apparent Lightness Mapping

We now describe our global mapping according to apparent lightness using the Nayatani model $L^*_N = L^*_{N_{VAC}}$ described in the previous section. The mapping process is as follows:

$$I_{RGB} \rightarrow I_{LUV} \rightarrow I_{L^*_N} \rightarrow G \tag{4}$$

We first convert the colour image to linear RGB by inverse gamma mapping, then transform to *CIELUV* colour space. Its apparent chromatic object lightness channel L^*_N is calculated according to Equation 2. We map L^*_N to greyscale Y values using reference white chromatic values for u^* and v^*. Finally, we apply gamma mapping to move from linear Y space back to a gamma-corrected greyscale image G. As shown in Figure 3 for several colour ramps, the mapping is continuous, there is no colour reordering, no lost discrimination and the dynamic range is preserved.

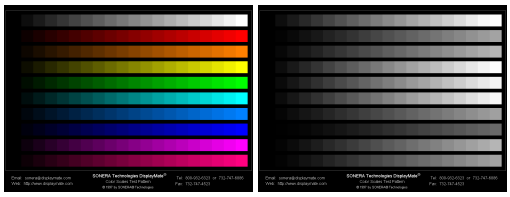


Figure 3: *On a colour test (left), G (right) preserves overall appearance and lightness ordering.*

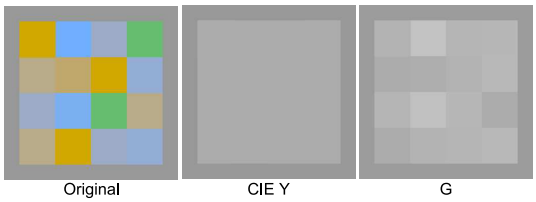


Figure 4: *Our approach maps isoluminant colours to unique, properly ordered greyvalues.*

Due to the compression of a 3D gamut to 1D, L^*_N may map two different colours to a similar lightness, which then are quantized to the same grey value. This occurs only when colours differ uniquely by hue, which is very uncommon in natural images and well-designed graphics. Even for a very challenging image that comprises equiluminant colours sampled from Neumann et. al's paper [NCN07], our global mapping discriminates appropriately, predicting the H-K effect that makes a more colourful blue appear lighter than the duller yellow, as shown in Figure 4 (view original colours on a calibrated screen). Recall that our goal is perceptual accuracy: the resulting low contrast properly represents the low contrast of the colour image, and each unique colour is mapped to a unique greyvalue. By incorporating the H-K effect, our global mapping partially solves the problem of grey value assignment and appropriately orders colours that normal luminance mapping can not discriminate.

© 2007 The Author(s)
Journal compilation © 2007 The Eurographics Association and Blackwell Publishing Ltd.

5. Local Chromatic Contrast Adjustment

The mapping described in the previous section captures some chromatic content in the greyscale image G according to the H-K effect. However, because of dimension reduction and unaccounted for hue differences, chromatic contrast may be reduced. Humans are most sensitive to these losses at local contrasts, regions where there is a visible discontinuity. To counter the reduction, we increase local contrast in the greyscale image G to better represent the local contrast of original I. The technique is adapted according to the ratio between colour and greyscale contrast, so that increases occur at underrepresented colour edges without unnecessarily enhancing edges that already represent the original. We cater the general adaptively-weigthed multiscale unsharp masking technique [NB98] to our goal of reintroducing chromatic contrast. Recently, this general tool has been used to good effect for adjusting contrast of tone mapped images [KMS07] and to restore contrast from depth perception [LCD06].

We perform contrast adjustments using the Laplacian pyramid that decomposes an image into n bandpass images h_i and a single lowpass image l [BA83]. Laplacian pyramids are built for I and G in *CIELAB* using a binomial coefficient filter of order 4. The h_i of each channel measures its local contrast, but as G contains no chromatic information, its local contrasts are contained entirely in its L^* channel. At each scale in the Laplacian pyramid, we adaptively increase local contrast $h_i(G_{L^*})$ by a perceptually-based amount λ_i, which measures the amount of contrast needed to match colour contrast $h_i(I)$. The enhanced greyscale image G' is computed by modifying G_{L^*} as follows:

$$G'_{L^*} = G_{L^*} + \sum_{i=0}^{n-1} k_i \lambda_i h_i(G_{L^*}) \tag{5}$$

where parameters $k_1, \ldots, k_{n-1}$, $k_i \leq 1$ exist so that the spatial effect can be controlled according to amount of discriminability desired and the intended viewing conditions (image size and viewing distance).

The goal of gain factor λ_i is to measure the remaining chromatic contrast to be restored during the enhancement. We define it as:

$$\lambda_i = \left(\frac{\Delta E(h_i(I))}{|h_i(G_{L^*})|} \right)^p \tag{6}$$

$\Delta E(h_i(I))$ is the colour contrast between a pixel and its neighbourhood which we measure by $\Delta E(h_i(I)) = (h_i(I_{L^*})^2 + h_i(I_{a^*})^2 + h_i(I_{b^*})^2)^{\frac{1}{2}}$. Since the chromatic channels of G contain no contrast information, $|h_i(G_{L^*})| \cong \Delta E(h_i(G))$. The ΔE colour difference is used so that both colour and grey contrasts are expressed in units of perceptual lightness. The parameter $0 \leq p \leq 1$ is used to remap the λ values to a non-linear scale so that weaker contrasts, like those from isoluminant colours, can be enhanced without over emphasizing stronger contrasts.

The parameters introduced exist to provide flexibility, allowing users to tweak according to their preference for desired discriminability. We criticize other approaches for being ad hoc, so have ensured that the parameters provide flexibility without allowing uncontrolled changes to the image. The overall lightness is not altered because we limit the number of subbands that may be enhanced, preventing changes from having too large an impact (in practice $n \leq 4$ levels). Most importantly, by definition of λ, edge polarity can not flip, meaning the lightness order of adjacent regions can not be changed.

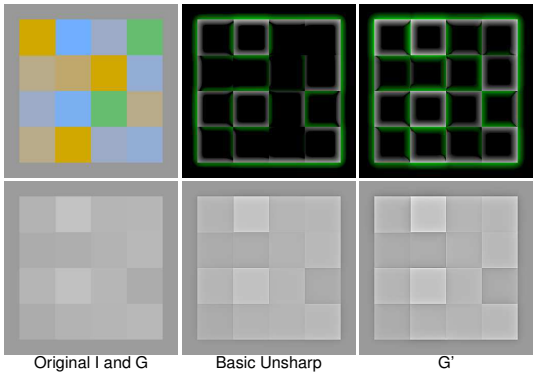


Figure 5: *Compared to the basic unsharp mask (middle), our chromatic enhancement (right) gains contrast where it is low in G and high in I. In the gain images, green values represent negative gain;* $p = 0.25$ $k = \{2, 1, 1, 0.6\}$.

We illustrate the effect of our local chromatic contrast adjustment in Figure 5. Contrast is nearly below threshold between isoluminant regions, especially among the bottom row of colours. A basic sharpening of all contrast (middle) does little to discriminate along that bottom row because the bandpass contains next to no signal. With our chromatic adjustment (right) it is possible to lift these contrasts above threshold without over emphasizing existing contrasts, so the resulting image better represents the original contrast.

6. Results and Discussion

The results presented here strive for perceptual accuracy, and do not attempt to increase or exaggerate discriminability. Therefore, the effects are apparent, but subtle. For comparison, we present either the CIE Y channel or Gimp greyscale with a basic unsharp enhancement, so that the reader is able to compare between images with matching overall sharpness. Additionally, the images presented here are for viewing on a calibrated colour screen (sRGB); for print, our resulting greyscale images should be mapped to the appropriate printer gamut.

We begin by showing that we discriminate between isoluminant colours by applying our approach to two images from Gooch et al. [GOTG05] in Figure 6. In both images, nearly

© 2007 The Author(s)
Journal compilation © 2007 The Eurographics Association and Blackwell Publishing Ltd.

isoluminant regions are distinguishable (parameters p and k, and thus n, are given with each image). We now compare to previous work to support our claims of better perceptual accuracy. We illustrate the consistency problem of local adaptive approaches in Figure 7 with Rasche's result using default parameters (top row) on similar images with varying colours (bottom row). While our results (middle row) also use default parameters $p = 0.5$, $k = \{0.5, 0.5, 0, 0\}$, the correct flower brightness ordering is preserved and the grey-values of leaves and backrounds are identical. Additionally, our images do not exaggerate the overall lightness range. Because we maintain consistency, we are able to apply our approach to video, as shown for a single frame in Figure 8 with constant parameters $p = 0.8$, $k = \{0.2, 0.8, 0, 0\}$. The red flowers become more visible and bright without changing the overall video appearance and maintaining temporal coherence. Please see the accompanying video or our project website for the full animation and other examples of our approach applied over time [WWW08].

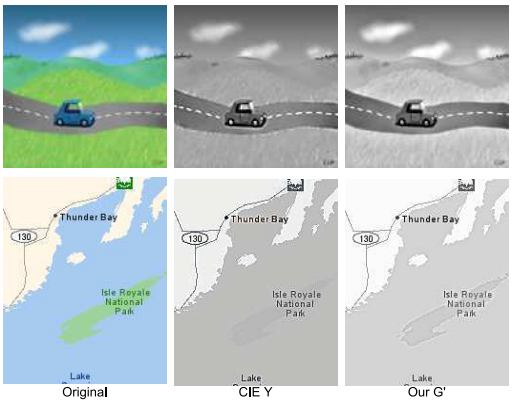


Figure 6: *Our conversions discriminate between colours even with similar luminance. Car $p = 0.5$ $k = \{0.5, 0.5, 0.5\}$; Island $p = 0.8$ $k = \{0.4, 0\}$.*

We consider changes to spatial content, as illustrated by greyscale versions of Monet's Impression Sunrise shown in Figure 9. Gooch's Color2Gray approach (bottom left) dilates the sun and reflection and has a strong blurring effect. Neumann et al. (bottom middle) masks details of the background structures, and alters the water's brightness, giving the impression of another light source. Our approach preserves the lightness of regions, the brightness of the sun, keeps all paint strokes visible, and when visually compared to original contains fewer spatial modifications (bottom right).

Finally, we show results on highly complex images that are more perceptually accurate than similarly sharpened Gimp greyscale. In Figure 10, the hats are more bright and the furthest two are distinguished more easily. In Figure 11, the red fish and stone advance and the two orange fish reappear.

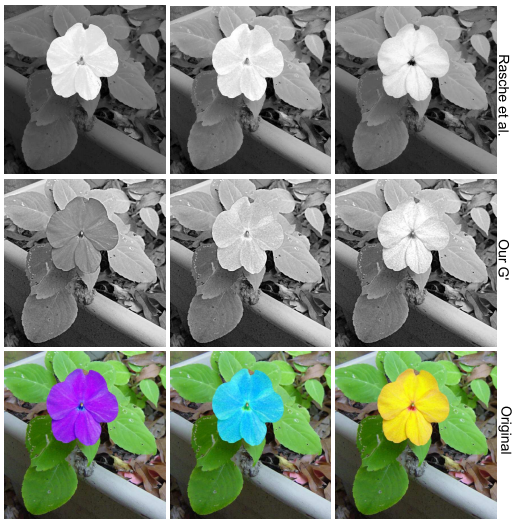


Figure 7: *Consistency and colour ordering: Rasche local conversion top, our conversion (middle), original colour (bottom) are recoloured versions from Rasche.*

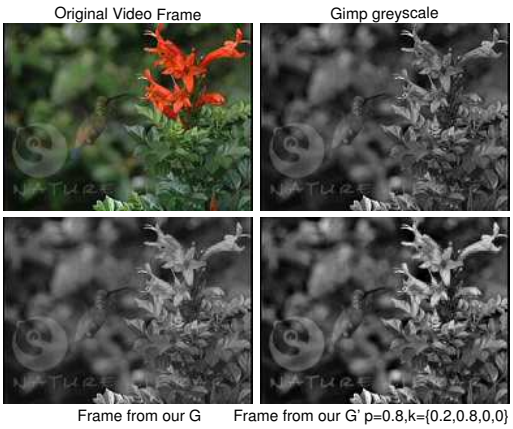


Figure 8: *A frame from our hummingbird video. Source: www.naturelibrary.com.*

We implemented our conversion in Octave using the Colour Engineering Toolbox and a Matlab toolbox for Laplacian pyramids. Our runtime depends on image resolution and the speed of colour mapping and pyramid construction. We specify times for both 1 and 4 storey pyramids, computed on an Intel4 3 GHz CPU. The Impression Sunrise image (311×223 pixels) takes 1.8 or 3.2 seconds; the Impatiens image (570×593) takes 6.7 or 10.8 seconds; and the Hummingbird video (192×144) single scale conversion takes 136.3 seconds with 0.96 seconds for each of the 142 frames. We also provide a single-scale GIMP plugin at [WWW08].

© 2007 The Author(s)
Journal compilation © 2007 The Eurographics Association and Blackwell Publishing Ltd.

7. Conclusions and Future Work

In this paper, we have presented a new approach to color to grey conversion. Our approach offers a more perceptually accurate appearance than standard luminance mapping and generates a closer response to the original than other approaches. In particular, we have incorporated the H-K effect which we consider fundamental to obtaining faithful greyscale reproductions. Our two-step approach is a good compromise between a fully automatic technique (first step) and user control (second step) making this approach well suited for natural images, photographs, artistic reproductions as well as business graphics. Finally a major benefit is the consistency we ensure by avoiding changes in color ordering. This makes this technique well adapted to the treatment of videos and image sets.

The main limitation of our approach is the locality of the second step. It can not restore chromatic contrast between non-adjacent regions. This step also risks introducing temporal inconsistencies, which is prevented by constant local parameters. Our algorithm may enhance artifacts from unreliable chromatic information, which occurs in low-quality agressively compressed media. With respect to colour appearance, we have not taken into account changes of chromatic contrast due to the context of the pixel, specifically its appearance with respect to its surrounding colours.

In future work, we plan to predict image appearance with respect to its color and spatial surrounding by incorporating the masking effect of colour patterns and measuring the *visibility* of the original contrasts using the contrast sensitivity function (CSF) for chromatic channels [Mul85] as a function of spatial frequency. We also plan to investigate methods for modifying enhancement parameters over time while maintaining temporal coherence. Further investigation into the problem of converting video to greyscale or to a reduced colour set is important for video stylization, processing and display on limited devices. We hope our work has fostered new ideas in this direction.

Acknowledgements

We thank Rafal Mantiuk, Hendrik Lensch, Andrei Lintu and the anonymous reviewers for their valuable comments.

Appendix A: Nayatani Chromatic Lightness Model

In CIELUV colour space u',v' are CIE 1976 chromaticity of test stimulus and u'_c,v'_c are chromaticities of the reference white and L_a adapting luminance, set by default to 20 as suggested by Nayatani. Following are equations used in chromatic lightness $L^*_{N_{VAC}}$ and $L^*_{N_{VCC}}$ [Nay97, Nay98].

$$K_{Br} = 0.2717\frac{6.469 + 6.362L_a^{0.4495}}{6.469 + L_a^{0.4495}} \tag{7}$$

$$s_{uv} = 13[(u' - u'_c)^2 + (v' - v'_c)^2]^{\frac{1}{2}} \tag{8}$$

$$\theta = tan^{-1}\frac{v' - v'_c}{u' - u'_c} \tag{9}$$

$$\begin{aligned} q(\theta) \;=\; & -0.01585 - 0.03017cos\theta - 0.04556cos2\theta \\ & -0.02667cos3\theta - 0.00295cos4\theta \\ & +0.14592sin\theta + 0.05084sin2\theta \\ & -0.01900sin3\theta - 0.00764sin4\theta \end{aligned} \tag{10}$$

References

[BA83] BURT P. J., ADELSON E. H.: The Laplacian Pyramid as a compact image code. *IEEE Trans. on Comm.* (1983), 532–540.

[BB04] BALA R., BRAUN K.: Color-to-grayscale conversion to maintain discriminability. *Proc. SPIE 5293* (2004), 196–202.

[BE04] BALA R., ESCHBACH R.: Spatial color-to-grayscale transformation preserving chrominance edge information. *Proc. IS&T/SID's 12th Color Imaging Conference* (2004), 82–86.

[CF03] CALABRIA A., FAIRCHILD M.: Perceived image contrast and observer preference i & ii: The effects of lightness, chroma, and sharpness manipulations on contrast perception. *Journal of Imaging Science & Technology*, 47 (2003), 479–493.

[Fai05] FAIRCHILD M.: *Color Appearance Models, 2nd Ed.* Wiley-IS and T, Chichester, UK, 2005.

[FP91] FAIRCHILD M., PIRROTTA E.: Predicting the lightness of chromatic object colors using CIELAB. *Color Research and Application 16*, 6 (1991), 385–393.

[GD07] GRUNDLAND M., DODGSON N. A.: Decolorize: Fast, contrast enhancing, color to grayscale conversion. *Pattern Recogn. 40*, 11 (2007), 2891–2896.

[GOTG05] GOOCH A. A., OLSEN S. C., TUMBLIN J., GOOCH B.: Color2gray: salience-preserving color removal. *ACM Trans. Graph. 24*, 3 (2005), 634–639.

[KMS07] KRAWCZYK G., MYSZKOWSKI K., SEIDEL H.-P.: Contrast restoration by adaptive countershading. In *Computer Graphics Forum (Proc. Eurographics 2007)* (2007), vol. 26.

[LCD06] LUFT T., COLDITZ C., DEUSSEN O.: Image enhancement by unsharp masking the depth buffer. *ACM Trans. Graph. 25*, 3 (2006), 1206–1213.

[MMS06] MANTIUK R., MYSZKOWSKI K., SEIDEL H.-P.: A perceptual framework for contrast processing of high dynamic range images. *ACM Transactions on Applied Perception 3*, 3 (July 2006), 286–308.

[Mul85] MULLEN K. T.: The contrast sensitivity of human color vision to red-green and blue-yellow chromatic gratings. *Journal of Psychology 359* (1985), 381–400.

[Nay97] NAYATANI Y.: Simple estimation methods for the Helmholtz-Kohlrausch effect. *Color Res. Appl. 22* (1997), 385–401.

[Nay98] NAYATANI Y.: Relations between the two kinds of representation methods in the Helmholtz-Kohlrausch effect. *Color Res. Appl. 23* (1998), 288–301.

[NB98] NOWAK R. D., BARANIUK R. G.: Adaptive weighted highpass filters using multiscale analysis. *IEEE Transactions on Image Processing 7*, 7 (1998), 1068 – 1074.

[NCN07] NEUMANN L., CADIK M., NEMCSICS A.: An efficient perception-based adaptive color to gray transformation. In *Proceedings of Computational Aesthetics 2007* (Banff, Canada, 2007), Eurographics Association, pp. 73– 80.

[RGW05] RASCHE K., GEIST R., WESTALL J.: Re-coloring images for gamuts of lower dimension. *Computer Graphics Forum 24* (September 2005), 423–432(10).

[WWW08] WWW: Apparent Greyscale Project http://www.mpi-inf.mpg.de/resources/ApparentGreyscale/, 2008.

[Wys67] WYSZECKI G.: Correlate for lightness in terms of CIE chromaticity coordinates and luminous reflectance. *Journal of the Optical Society of America (1917-1983) 57* (Feb. 1967).

© 2007 The Author(s)
Journal compilation © 2007 The Eurographics Association and Blackwell Publishing Ltd.

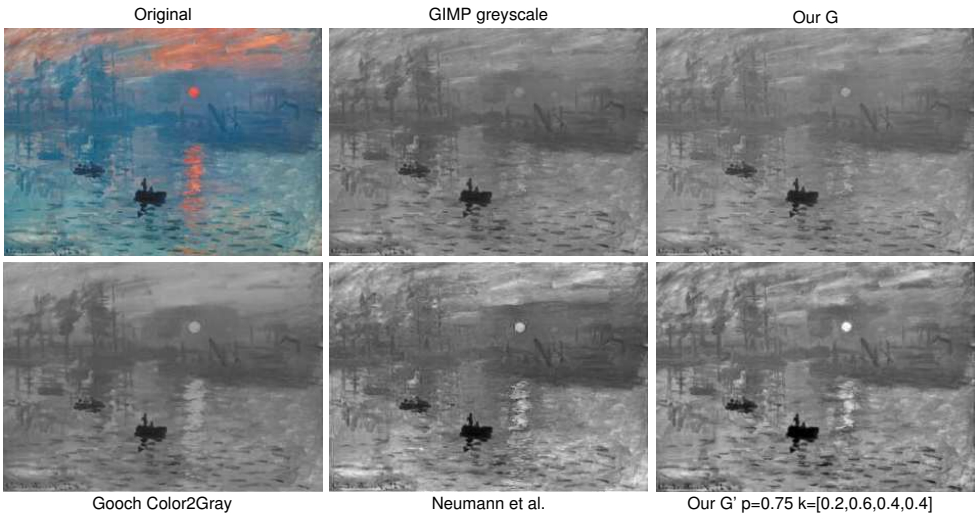


Figure 9: *Our impression is more like the original because it preserves the paint strokes especially in the sky and background. Gooch's image is strongly blurred with a dilated sun and Neumann masks the background and lightens the water.*

Figure 10: *The extreme brightness of the hats is more apparent in our image than Gimp's greyscale which highlights the differences between the furthest two hats. Source: www.vischeck.com.*

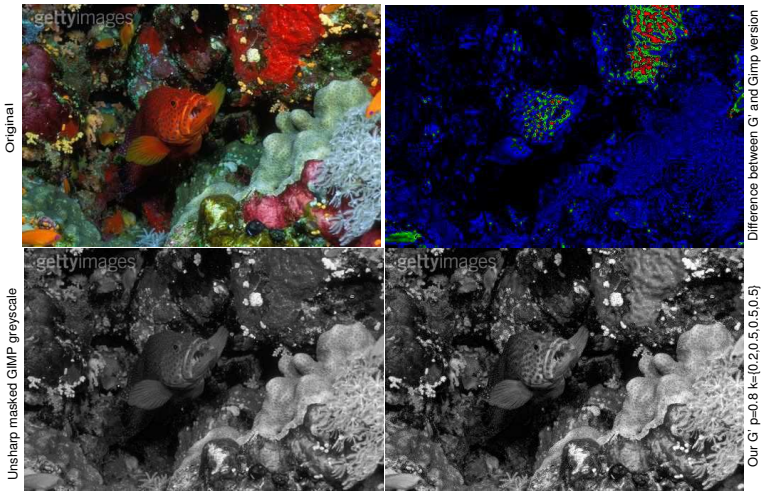


Figure 11: *Our approach accentuates the red fish and stone, and restores salience to the orange fish (bottom left and right side).Source: Getty Images.*

© 2007 The Author(s)
Journal compilation © 2007 The Eurographics Association and Blackwell Publishing Ltd.

Diffusion Curves: A Vector Representation for Smooth-Shaded Images

Alexandrina Orzan[1,2] Adrien Bousseau[1,2,3] Holger Winnemöller[3] Pascal Barla[1] Joëlle Thollot[1,2] David Salesin[3,4]
[1]INRIA [2]Grenoble University [3]Adobe Systems [4]University of Washington

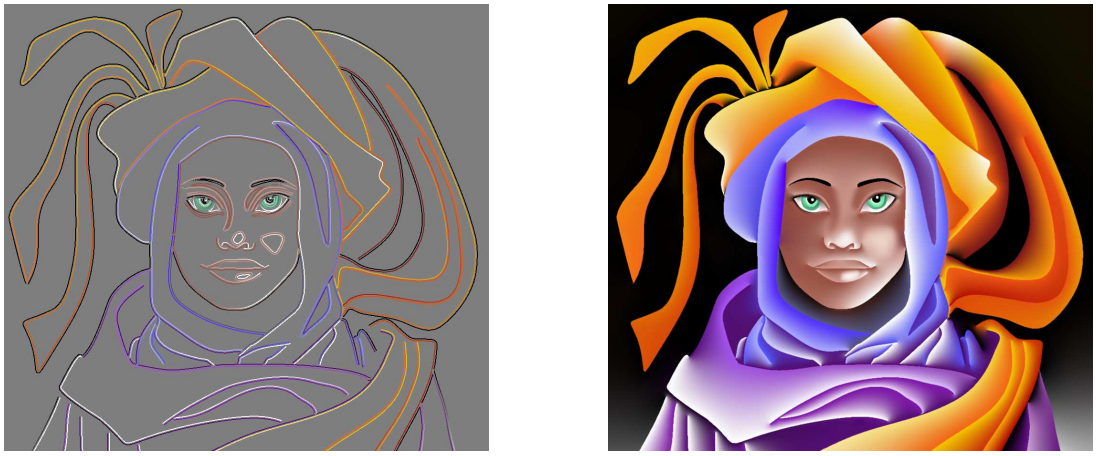


Figure 1: *Diffusion curves (left), and the corresponding color image (right). Note the complex shading on the folds and blur on the face.*

Abstract

We describe a new vector-based primitive for creating smooth-shaded images, called the *diffusion curve*. A diffusion curve partitions the space through which it is drawn, defining different colors on either side. These colors may vary smoothly along the curve. In addition, the sharpness of the color transition from one side of the curve to the other can be controlled. Given a set of diffusion curves, the final image is constructed by solving a Poisson equation whose constraints are specified by the set of gradients across all diffusion curves. Like all vector-based primitives, diffusion curves conveniently support a variety of operations, including geometry-based editing, keyframe animation, and ready stylization. Moreover, their representation is compact and inherently resolution-independent. We describe a GPU-based implementation for rendering images defined by a set of diffusion curves in realtime. We then demonstrate an interactive drawing system for allowing artists to create artworks using diffusion curves, either by drawing the curves in a freehand style, or by tracing existing imagery. The system is simple and intuitive: we show results created by artists after just a few minutes of instruction. Furthermore, we describe a completely automatic conversion process for taking an image and turning it into a set of diffusion curves that closely approximate the original image content.

CR Categories: I.3.3 [COMPUTER GRAPHICS]: Picture/Image Generation, Graphics Utilities;

Keywords: Vector graphics, vectorization, gradient mesh, color diffusion, image creation, image reconstruction

1 Introduction

Vector graphics, in which primitives are defined geometrically, dates back to the earliest days of our field. Early cathode-ray tubes, starting with the Whirlwind-I in 1950, were all vector displays, while the seminal Sketchpad system [Sutherland 1980] allowed users to manipulate geometric primitives like points, lines, and curves. *Raster graphics* provides an alternative representation for describing images via a grid of pixels rather than geometry. Raster graphics arose with the advent of the framebuffer in the 1970s and is now commonly used for storing, displaying, and editing images. However, while raster graphics offers some advantages over vector graphics — primarily, a more direct mapping between the hardware devices used for acquisition and display of images, and their internal representation — vector graphics continues to provide certain benefits as well. Most notably, vector graphics offers a more compact representation, resolution-independence (allowing scaling of images while retaining sharp edges), and geometric editability. Vector-based images are more easily animated (through keyframe animation of their underlying geometric primitives), and more readily stylized (e.g. through controlled perturbation of geometry). For all of these reasons, vector-based drawing tools, such as Adobe Illustrator©, CorelDraw©, and Inkscape©, continue to enjoy great popularity, as do standardized vector representations, such as Flash and SVG.

However, for all of their benefits, vector-based drawing tools offer only limited support for representing complex color gradients, which are integral to many artistic styles. For example, the art movements of realism and hyperrealism rely on smooth gradients to achieve soft shadows, defocus blur, diffuse shading, glossy reflections, and various material effects. The *airbrush* technique, widely used in design and urban art, is fundamentally based on (physical) color diffusion. The *Art Deco* painting movement, various comic styles, as well as numerous painting styles also heavily feature color gradients. While graphic artists, such as Yukio Miyamoto[1], have been able to achieve stunning results using existing vector-graphics tools, creating these artworks requires a high degree of artistic skill, not to mention an enormous degree of patience (the reported creation times for some art pieces run into the hundreds of hours).

[1] www.khulsey.com/masters_yukio_miyamoto.html

What makes creating this artwork so difficult using vector primitives is that most existing vector formats support only linear or radial gradients[2]. Currently, the most sophisticated vector-based tool for handling complex gradients is the *gradient mesh*. A gradient mesh is a lattice with colors at each vertex that are linearly interpolated across the mesh. While more powerful than simple gradients (or "ramps"), gradient meshes still suffer from some limitations. A significant one is that the topological constraints imposed by the mesh lattice give rise to an overcomplete representation that becomes increasingly inefficient and difficult to create and manipulate. Recently, Sun et al. [2007] presented an important advance: a semi-automatic method for optimizing a manually initialized mesh. While their approach lessens the manual demands on the user, it does not address the process of subsequent gradient-mesh manipulation, nor does it facilitate free-hand creation of gradient meshes from scratch.

In this paper we propose a novel vector-graphics primitive, called the *diffusion curve*. A diffusion curve is a curve that diffuses colors on both sides of the space that it divides. The motivations behind such a representation are twofold:

First, this representation supports traditional freehand drawing techniques. Artists often begin by sketching shapes, then adding color later. In a typical drawing session with our tool, the artist first lays down drawing curves corresponding to color boundaries. In contrast with traditional vector graphics, the color boundaries do not need to be closed and may even intersect. A diffusion process then creates smooth gradients on either side of the curves. By specifying blur values along a curve, the artist can also create smooth color transitions across the curve boundaries.

Second, most color variations in an image can be assumed to be caused by edges (material and depth discontinuities, texture edges, shadow borders, etc.) [Koenderink and Doorn 1979; Marr and Hildreth 1980]. Even subtle shading effects can be modeled as though caused by one or more edges, and it has been demonstrated that edges constitute a near-complete and natural primitive for encoding and editing images [Carlsson 1988; Elder 1999; Elder and Goldberg 2001]. In this work, we rely on vision algorithms, such as edge detection, to convert an image into our diffusion curves representation fully automatically.

The main contribution of this paper is therefore the definition of diffusion curves as a fundamental vector primitive, along with two types of tools: (1) A prototype allowing manual creation and editing of diffusion curves. Thanks to an efficient GPU implementation, the artist benefits from instant visual feedback despite the heavy computational demands of a global color diffusion. (2) A fully automatic conversion from a bitmap image based on scale-space analysis. The resulting diffusion curves faithfully represent the original image and can then be edited manually.

2 Previous work

We review here existing techniques to create complex color gradients and blur with modern vector graphic tools.

For a long time, vector graphics have been limited to primitives (paths, polygons) filled with uniform color or linear and radial gradients. Although skillful artists can create rich vector art with these simple tools, the resulting images often present flat or limited shading due to the limitations in terms of complex gradients and blur. Commercial tools such as Adobe Live Trace© assist the user in creating complex vector graphics from input bitmap images. They operate by segmenting an input image into regions of constant or slowly varying color, and fitting polygons onto these primitives. Although this class of methods produces convincing results in uniform areas, the segmentation typically generates a prohibitive number of primitives in smooth regions.

The ArDeco system of Lecot et al. [2006] allows vectorization of more complex gradients using existing linear or radial gradient primitives. It is based on a segmentation of the input image into regions of slowly varying color, and an approximation of color variations within each region with linear or quadratic gradients. The resulting primitives are fully compatible with the SVG standard, but the approximation tends to produce sharp color transitions between segmented regions. A simpler solution to bypass these limitations, adopted by the SVG format and modern tools (Adobe Illustrator©, Corel CorelDraw©, Inkscape©), is to reblur the image once vector primitives have been rasterized. However, they only allow for a uniform blur for each given primitive, which, similar to the limitations of flat colors or simple gradients, necessitates an unpractically large number of primitives to approximate complex image content.

Gradient meshes have been recently introduced (Adobe Illustrator©, Corel CorelDraw©) to address all of these issues by allowing a user to specify color values on the vertices of a quad mesh and smoothly interpolating these values over the mesh faces. However, creating a mesh from scratch requires much skill and patience, because the artist needs to accurately anticipate the mesh resolution and topology necessary to embed the desired smooth features. This is why most users rely on an example bitmap to drive the design of realistic gradient meshes. The users first decompose an input photograph into several sub-objects and then draw meshes over each sub-object following their topology; finally, they sample colors in the original photograph, assigning them to the mesh vertices. Many tutorials describing this approach are available on the Web. Still, drawing effective meshes and performing accurate manual color sampling is very time consuming in practice (several hours or even days for detailed images) and requires a good appreciation of the image complexity to adopt an adequate mesh resolution. Recently, the paper of Sun et al [2007] proposed to assist the user by automatically fitting an input gradient mesh to an input image. The fitting is achieved by minimizing the reconstruction error between the resulting image and an input photograph. Their semi-automatic method greatly reduces the time required to draw a precise mesh and sampling colors, although the user still has to manually specify the sub-objects of the picture and draw the initial meshes with an adequate resolution. Price and Barret [2006] proposed a similar approach for object vectorization, using recursive subdivisions until the reconstruction error falls below a fixed threshold. Their method produces faithful results but also generates many small patches in smooth regions.

Yet, with both approaches, it remains unclear how to efficiently manipulate the resulting meshes for further editing. We believe this is due to the unnecessary constraints imposed by the use of a mesh: using a predefined topology, employing specific patch subdivision schemes, and choosing a global orientation. In practice, this translates into a dense net of patches that are not readily connected to the depicted content. Hence, the manipulation of such a set of primitives quickly becomes prohibitive for the non-expert.

The new representation described in this paper offers the same level of visual complexity as that reached by gradient meshes, but has two main advantages: it is *sparse*, and corresponds to *meaningful* image features. Indeed, the newly introduced diffusion curves are intuitive to create, as each primitive corresponds to an image feature; they are easy to manipulate and animate, as no constraint is imposed on connectivity, and no superfluous subdivision is required; and they are well adapted for stylization, which would be non-trivial with a gradient mesh approach. Moreover, compared to

[2] www.carto.net/papers/svg/comparison_flash_svg/

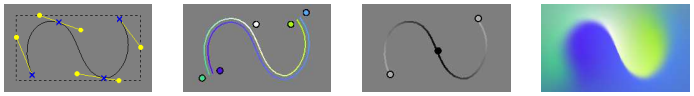
other methods reviewed above, our representation naturally lends itself to automatic extraction from a bitmap image: primitive locations are found completely automatically, and primitive attributes (color and blur) are extracted via vision algorithms.

In other words, compared to regions used in classic vector representations, or patches used in gradient meshes, our approach is motivated by the fact that most of the color variation in an image is caused by, or can be modeled with edges; and that (possibly open) regions or patches are implicitly defined in between. Such a sparse image representation is strongly motivated by the work of Elder [1999], who demonstrated that edges are a near-complete representation for images. Elder [2001] also suggested the possibility of using edges to efficiently manipulate images with basic operations (edge delete, copy and paste). However, we believe the full potential of this approach has as yet not been attained. For this reason, our conversion algorithm starts from the same premises as Elder's system. But by vectorizing edges and their attributes, we extend its manipulation capabilities to include shape, color, contrast, and blur operations. This way, we provide the user with more intuitive editing tools, and also support resolution-independence, stylization and key-frame animation.

3 Diffusion Curves

In this section we introduce the basic primitive of our representation, called a *diffusion curve*, and describe how to efficiently render an image from such primitives. Specification and manipulation of diffusion curves are discussed in subsequent sections.

3.1 Data structure



(a) (b) (c) (d)

Figure 2: *A Diffusion curve is composed of (a) a geometric curve described by a Bézier spline, (b) arbitrary colors on either side, linearly interpolated along the curve, (c) a blur amount linearly interpolated along the curve. The final image (d) is obtained by diffusion and reblurring. Note the complex color distribution and blur variation defined with a handful of controls.*

The basic element of a diffusion curve is a geometric curve defined as a cubic Bézier spline (Figure 2(a)) specified by a set of control points P. The geometry is augmented with additional attributes: two sets of color control points C_l and C_r (Figure 2(b)), corresponding to color constraints on the *right* and *left* half space of the curve; and a set of *blur* control points (Σ) that defines the smoothness of the transition between the two halves (Figure 2(c)). Intuitively, the curves diffuse color on each side with a soft transition across the curve given by its blur (Figure 2(d)).

Color and blur attributes can vary along a curve to create rich color transitions. This variation is guided by an interpolation between the attribute control points in attribute space. In practice, we use linear interpolation and consider colors in RGB space throughout the rendering process, because they map more easily onto an efficient GPU implementation and proved to be sufficient for the artists using our system. Controls points for geometry and attributes are stored independently, since they are generally uncorrelated. This leads to four independent arrays in which the control points (geometry and attribute values) are stored together with their respective parametric position t along the curve:

DiffusionCurve:	P$[n_{pos}]$: array of $(x, y,$ tangent$)$; $C_l[n_l]$: array of (r, g, b, t); $C_r[n_r]$: array of (r, g, b, t); $\Sigma[n_\sigma]$: array of (σ, t);

The diffusion curves structure encodes data similar to Elder's edge-based representation [1999]. However, the vectorial nature of a diffusion curve expands the capabilities of Elder's discrete edges by allowing precise control over both shapes — via manipulation of control points and tangents — and appearance attributes — via color and blur control points (small circles on the Figures). This fine-level control, along with our realtime rendering procedure, facilitates the drawing and editing of smooth-shaded images.

3.2 Rendering smooth gradients from diffusion curves

Three main steps are involved in our rendering model (see Figure 3): (1) rasterize a *color sources* image, where color constraints are represented as colored curves on both sides of each Bézier spline, and the rest of the pixels are uncolored; (2) *diffuse* the color sources similarly to heat diffusion — an iterative process that spreads the colors over the image; we implement the diffusion on the GPU to maintain realtime performance; and (3) *reblur* the resulting image with a spatially varying blur guided by the blur attributes. Technical details about these three steps are explained in the following sections.

3.2.1 Color sources

Using the interpolated color values, the first step renders the left and right color sources $cl(t), cr(t)$ for every pixel along the curves. An alpha mask is computed along with the rendering to indicate the exact location of color sources versus undefined areas.

For perfectly sharp curves, these color sources are theoretically infinitely close. However, rasterizing pixels separated by too small a distance on a discrete pixel grid leads to overlapping pixels. In our case, this means that several color sources are drawn at the same location, creating visual artifacts after the diffusion. Our solution is to distance the color sources from the curve slightly, and to add a color gradient constraint directly on the curve. The gradient maintains the sharp color transition, while the colors, placed at a small distance d in the direction normal to the curve, remain separate.

More precisely, the gradient constraint is expressed as a gradient field $\mathbf{w}$ which is zero everywhere except on the curve, where it is equal to the color derivative across the curve. We decompose the gradient field in a gradient along the x direction $\mathbf{w}_x$ and a gradient along the y direction $\mathbf{w}_y$. For each pixel on the curve, we compute the color derivative across the curve from the curve normal N and the left (cl) and right (cr) colors as follow (we omit the t parameter for clarity): $w_{x,y} = (cl - cr)N_{x,y}$

We rasterize the color and gradient constraints in 3 RGB images: an image C containing colored pixels on each side of the curves, and two images W_x, W_y containing the gradient field components. In practice, the gradient field is rasterized along the curves using lines of one pixel width. Color sources are rasterized using triangle strips of width $2d$ with a special pixel shader that only draws pixels that are at the correct distance d (Figure 3(1)). In our implementation d is set at 3 pixels. Pixel overlap can still occur along a curve in regions of high curvature (where the triangle strip overlaps itself) or when two curves are too close to each other (as with thin structures or intersections). A simple stencil test allows us to discard overlapping color sources before they are drawn, which implies that solely the gradient field $\mathbf{w}$ dictates the color transitions in these areas. An example of such case can be seen in Figure 1, where the eyebrows are accurately rendered despite their thin geometry.

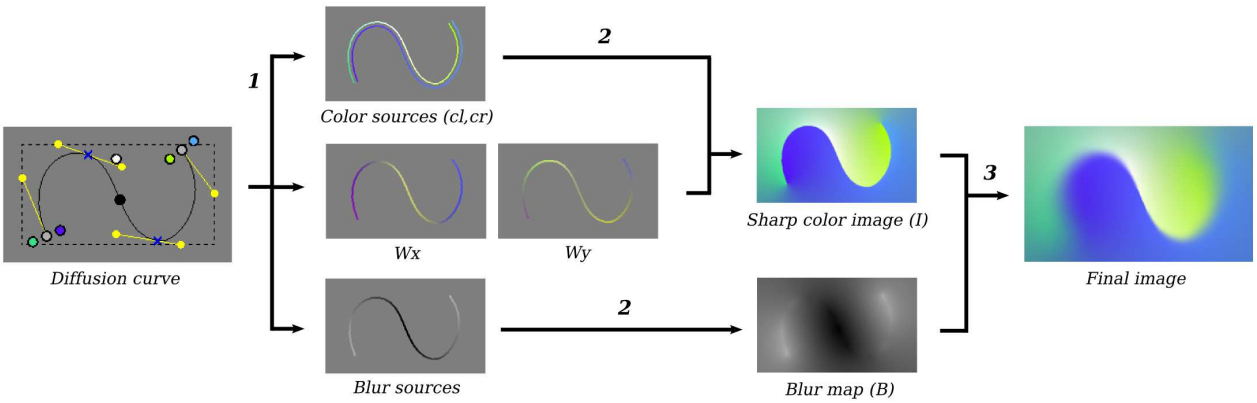


Figure 3: *Rendering diffusion curves requires (1) the rasterization of the color and blur sources, along with the gradient field* $\mathbf{w} = (\mathbf{w}_x, \mathbf{w}_y)$*, (2) the diffusion of colors and blur, and (3) the reblurring of the color image.*

3.2.2 Diffusion

Given the color sources and gradient fields computed in the previous step, we next compute the color image I resulting from the steady state diffusion of the color sources subject to the gradient constraints (Figure 3(2)). Similarly to previous methods [Carlsson 1988; Elder 1999; Pérez et al. 2003], we express this diffusion as the solution to a Poisson equation, where the color sources impose local constraints:

$$\begin{aligned} \Delta I &= \operatorname{div} \mathbf{w} \\ I(x,y) &= C(x,y) \text{ if pixel } (x,y) \text{ stores a color value} \end{aligned}$$

where Δ and div are the Laplace and divergence operators.

Computing the Poisson solution requires solving a large, sparse, linear system, which can be very time consuming if implemented naively. To offer interactive feedback to the artist, we solve the equation with a GPU implementation of the multigrid algorithm [Briggs et al. 2000; Goodnight et al. 2003]. The paper of McCann and Pollard [2008], published in the same proceedings, gives a detailed description of a simple implementation very similar to ours. The idea behind multigrid methods is to use a coarse version of the domain to efficiently solve for the low frequency components of the solution, and a fine version of the domain to refine the high frequency components. We use Jacobi relaxations to solve for each level of the multigrid, and limite the number of relaxation iterations to achieve realtime performances. Typically, for a 512×512 image we use $5i$ Jacobi iterations per multigrid level, with i the level number from fine to coarse. This number of iterations can then be increased when high quality is required. All the images in this paper and in the accompanying video have been rendered using an Nvidia GeForce 8800, providing realtime performance on a 512×512 grid with a reasonable number of curves (several thousands).

3.2.3 Reblurring

The last step of our rendering pipeline takes as input the color image containing sharp edges, produced by the color diffusion, and reblurs it according to blur values stored along each curve. However, because the blur values are only defined along curves, we lack blur values for off-curve pixels. A simple solution, proposed by Elder[1999], diffuses the blur values over the image similarly to the color diffusion described previously. We adopt the same strategy and use our multigrid implementation to create a blur map B from the blur values. The only difference to the color diffusion process is that blur values are located exactly on the curve so we do not require any gradient constraints. This leads to the following equation:

$$\begin{aligned} \Delta B &= 0 \\ B(x,y) &= \sigma(x,y) \text{ if pixel } (x,y) \text{ is on a curve} \end{aligned}$$

Giving the resuting blur map B, we apply a spatially varying blur on the color image (Figure 3(3)), where the size of the blur kernel at each pixel is defined by the required amount of blur for this pixel. Despite a spatially varying blur routine implemented on the GPU [Bertalmio et al. 2004], this step is still computationally expensive for large blur kernels (around one second per frame in our implementation), so we bypass it during curve drawing and manipulations and reactivate it once the drawing interaction is complete.

3.2.4 Panning and zooming

Solving a Poisson equation leads to a global solution, which means that any color value can influence any pixel of the final image. Even though the local constraints introduced by the color sources reduce such global impact, this raises an issue when zooming into a sub-part of an image, because curves outside the current viewport should still influence the viewport's content. To address this problem without requiring a full Poisson solution at a higher resolution, we first compute a low-resolution diffusion on the unzoomed image domain, and use the obtained solution to define Dirichlet boundary conditions around the zooming window. This gives us a sufficiently good approximation to compute a full-resolution diffusion only within the viewport.

4 Creating diffusion curves

The process of creating an image varies among artists. One may start from scratch and give free rein to his imagination while another may prefer to use an existing image as a reference. We provide the user with both options to create diffusion curves. For *manual* creation, the artist can create an image with our tool by sketching the lines of the drawing and then filling in the color. When using an image as a template, we propose two methods. *Assisted:* The artist can trace manually over parts of an image and we recover the colors of the underlying content. *Automatic:* the artist can automatically convert an image into our representation and possibly post-edit it.

4.1 Manual creation

To facilitate content creation for the artist, we offer several standard tools: editing of curve geometry, curve splitting, copy/paste, zooming, color picking, etc. We also developed specific tools: copy/paste of color and blur attributes from one curve to another, editing of attributes control points (add, delete, and modify), etc. The tutorial provided on our project page [3] and conference DVD gives a more complete description of our current prototype interface.

[3] http://artis.imag.fr/Publications/2008/OBWBTS08/

(a) (b) (c) (d)

Figure 4: *Example steps for manual creation: (a) sketching the curves, (b) adjusting the curve's position, (c) setting colors and blur along the diffusion curve and (d) the final result. The image was created by an artist at first contact with the tool and it took 4 hours to create.*

To illustrate how an artist can use our diffusion curves, we show in Figure 4 (and accompanying video) the different stages of an image being drawn with our tool. The artist employs the same intuitive process as in traditional drawing: a sketch followed by color filling.

4.2 Tracing an image

In many situations an artist will not create an artwork entirely from scratch, but instead use existing images for guidance. For this, we offer the possibility of extracting the colors of an underlying bitmap along a drawn curve. This process is illustrated in Figure 5.

Figure 5: *Tracing with diffusion curves: (a) Original bitmaps; (b)* left*: Result of a stylistic tracing using color sampling (artist drawing time: less than a minute);* right*: Result of a tracing using active contours and color sampling (artist drawing time: 90 minutes). (c) The corresponding diffusion curves (color sources have been thickened for illustration purpose).*

The challenge here is to correctly extract and vectorize colors on each side of a curve. We also need to consider that color outliers might occur due to noise in the underlying bitmap or because the curve positioning was suboptimal. We first uniformly sample the colors along the curve at a distance d (same as the one used for rendering) in the direction of the curve's normal. We then identify color outliers by measuring a standard deviation in a neighborhood of the current sample along the curve. To this end, we work in CIE L*a*b* color space (considered perceptually uniform for just-noticeable-differences), and tag a color as an outlier if it deviates too much from the mean in either the L*, a* or b* channel. We then convert back colors to RGB at the end of the vectorization process for compatibility with our rendering system.

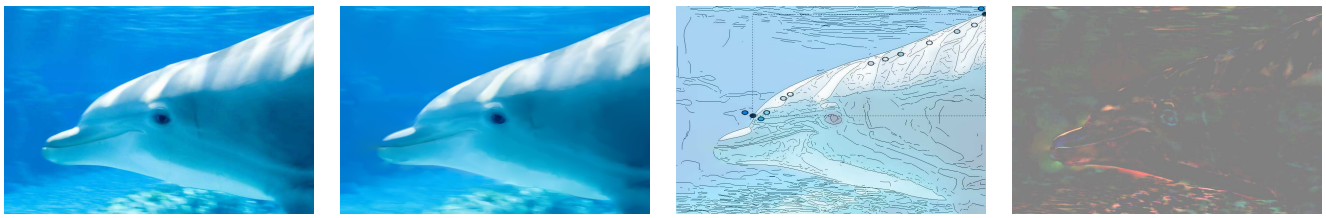
To obtain a linear color interpolation similar to that used for rendering, we fit a polyline to the color points using the Douglas-Peucker algorithm [Douglas and Peucker 1973]. The iterative procedure starts with a line connecting the first and last point and repeatedly subdivides the line into smaller and smaller segments until the maximum distance (still in CIE L*a*b*) between the actual values and the current polyline is smaller than the error tolerance ϵ. The end points of the final polyline yield the color control points that we attach to the curve. A creative example that uses color sampling is illustrated in Figure 5(b)-left image, where an artist has drawn very stylistic shapes, while using the color sampling feature to reproduce the global tone of the original image, similarly to an in-painting process [Bertalmio et al. 2000].

When tracing over a template, one would normally want to position the curves over color discontinuities in the underlying image. Since it is not always easy to draw curves precisely at edge locations in a given image, we provide some help by offering a tool based on *Active Contours* [Kass et al. 1987]. An active contour is attracted to the highest gradient values of the input bitmap and allows the artist to iteratively snap the curve to the closest edge. The contour can also be easily corrected when it falls into local minima, or when a less optimal but more stylistic curve is desired. Figure 5(b)-right shows the image of a lady bug created using geometric snapping and color extraction. While the artist opted for a much more stylized and smoothed look compared to the original, the image still conveys diffuse and glossy effects, defocus blur, and translucency.

4.3 Automatic extraction from a bitmap

Finally, an artist might want to add stylization and expression to an already existing image. We propose a method for automatically extracting and vectorizing diffusion curves data from a bitmap.

Data Extraction: Many approaches exist to find edges and determine their blur and color attributes. We based our data extraction on our previous work [Orzan et al. 2007], as that work was "designed" for edge-based image stylization. Diffusion curves extend our previous approach from a raster-based method to a fully vectorized version. For brevity, we will only recall the basic steps of the previous method, and refer the interested reader to Orzan et al. [2007].



(a) (b) (c) (d)

Figure 6: *Example of our reconstruction: (a) original image; (b) result after conversion into our representation; (c) automatically extracted diffusion curves; (d) RGB difference between original and reconstructed image (amplified by 4); note that the most visible error occurs along edges, most probably because, through vectorization, we change their localization.*

The basic approach relies on the Gaussian scale space, which can be pictured as a stack of increasingly blurred versions of an image, where higher scale images exhibit less and less detail. To obtain edge positions and blur estimates from this scale space, we first extract edges at all available scales using a classical Canny detector [Canny 1986]. Contrary to our previous approach, which works only on the luminance channel, we now apply the Canny detector to the multi-channel color gradient described in Di Zenzo [1986]. This allows us to detect sharp color variations in iso-luminant regions of the image, where a luminance gradient would fail. Then, taking inspiration from previous work in scale-space analysis [Lindeberg 1996; Elder 1999], we find the scale at which an edge is best represented (the more blurred the edge, the higher the scale). We use this ideal scale to identify the degree of blur of an edge, as previously, but also use it to localize edges. It should be noted that very blurry edges are difficult to detect and parameterize accurately. In our system we find that very large gradients are sometimes approximated with a number of smaller ones.

After the scale-space analysis, we are left with anhave chosen to edge map, which contains the edge locations and the blur values for the edge pixels. As explained in our previous paper [Orzan et al. 2007], we also provide a method to extract an edge's *lifetime* (a measure of its spread throughout consecutive scales), which allows for separation of detail and contour edges. While in the past we directly used the extracted data, our new approach requires an additional processing step: colors on either side of the edges must be extracted explicitly. To this end, we connect pixel-chains from the edge map and proceed to sample colors in the original image on each side of the edge in the direction of the edge normal. In practice, the gradient normal to the edge is difficult to estimate for blurry edges, so we use the direction given by the normal of a polyline fitted to each edge. For an estimated blur σ, we pick the colors at a distance $3 \cdot \sigma$ from the edge location, which covers 99% of the edge's contrast, assuming a Gaussian-shaped blur kernel [Elder 1999]. While the $3 \cdot \sigma$ distance ensures a good color extraction for the general case, it poses numerical problems for structures thinner than 3 pixels ($\sigma < 1$); in this particular case, color cannot be measured accurately.

Conversion to diffusion curves: For vectorization of positions, we take inspiration from the approach used in the open source Potrace© software [Selinger 2003]. The method first approximates a pixel chain with a polyline that has a minimal number of segments and the least approximation error, and then transforms the polyline into a smooth poly curve made from end-to-end connected Bézier curves. The conversion from polylines to curves is performed with classical least square Bézier fitting based on a maximum user-specified fitting error and degree of smoothness. For attribute vectorization, we use the same method as in Section 4.2.

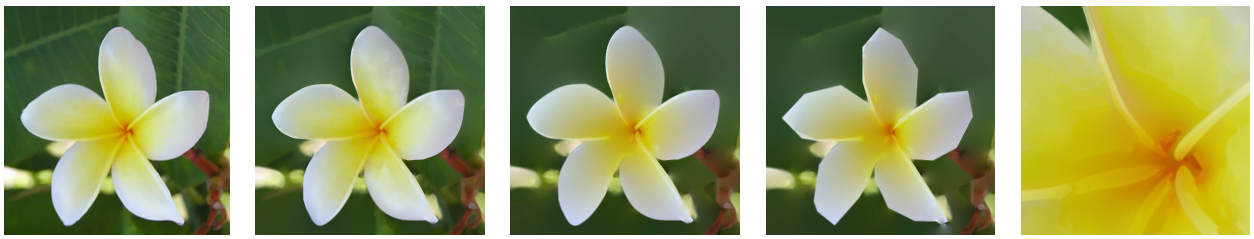
Several parameters determine the complexity and quality of our vectorized image representation. For the edge geometry, the Canny threshold determines how many of the image edges are to be considered for vectorization; a despeckling parameter sets the minimum length of a pixel chain to be considered for vectorization; and finally, two more parameters set the smoothness of the curve fitting and the fitting error. For the blur and color values, two parameters are considered: the size of the neighborhood for eliminating outliers, and the maximum error accepted when fitting the polyline. For most images in this paper, we use a Canny high threshold of 0.82 and low threshold of 0.328, we discard pixel chains with less than 5 pixels, we use a smoothness parameter of 1 (Potrace default) and we set the fitting error to 0, so the curve closely approximates the original edges. For attributes, we consider a neighborhood of 9 samples, and the maximum error accepted is 2 blur scales for the blur and 30 CIE L*a*b* units for colors.

5 Results

Diffusion curves, as vector-based primitives, benefit from the advantages of traditional vector graphics: zooming-in preserves sharp transitions (Figure 8 (e)) and keyframing is easily performed via linear interpolation of geometry and attributes (Figure 7). Our representation is equally well suited for global and local image stylization. Curve shapes and attributes can be easily modified to obtain effects such as that presented in Figure8(d). For diffusion curves extracted from an image, the lifetime measure provided by Orzan et al. [2007] can be used to adjust preservation of detail (Figure8(c)).

Figure 7: *Keyframing with diffusion curves: Three keyframes of an animation.*

To validate our approach and to collect valuable practical feedback, we had various artists use our prototype. Most figures in this paper were generated in these sessions. All artists were well versed in digital content creation tools, but had no technical background. They were given a brief tutorial (see our project page and the conference DVD), amounting to approximately 10 minutes of instructions. The artists were able to create many varied and intricate examples from the very first session and found the manipulation of diffusion curves intuitive after a short accommodation phase. Manual image creation took anywhere from several minutes (Figure 5(b)) to a few hours (Figure 4). However, the artists agreed that a more powerful user interface would greatly speed up the creation process.



(a) (b) (c) (d) (e)

Figure 8: *Stylization effects: (a) original bitmap; (b) Automatic reconstruction; (c) Reconstruction simplified by removing edges with low lifetime; (d) Global shape stylization applied to (c); (e) Enlargement of (b).*

6 Discussion & Future work

In the previous sections, we presented our new vector-based primitive, and explained the various options at an artist's disposal to create smooth-shaded images thanks to this intuitive representation. We now compare our approach with the most commonly used vector tool for creating images with similarly complex gradients: Gradient Meshes. Next, we identify the remaining challenges that we would like to address in future work.

6.1 Comparison with Gradient Meshes

Representational efficiency: In terms of sparsity of encoding, both gradient meshes and diffusion curves are very efficient image representations. A direct comparison between both representations is difficult, as much depends on the chosen image content (for example, gradient meshes require heavy subdivision to depict sharp edges and it can be difficult to conform the mesh topology to complex geometric shapes). Furthermore, Price and Barret [2006] presented a more compact sub-division gradient mesh, yet all available tools employ a regular mesh. While the diffusion curves representation appears more compact at first glance (see Figure 11), it should be noted that each geometric curve can hold an arbitrary amount of color and blur control points (see Table 9). So, while the sparsity of encoding of both representations can be considered comparable, we would argue the flexibility of diffusion curves to be a significant benefit, as it allows us any degree of control on a curve, without a topologically-imposed upper or lower bound on the number of control points.

		Curves	P	Cl	Cr	Σ
Roses	(fig. 5 left)	20	851	581	579	40
Lady bug	(fig. 5 right)	71	521	293	291	144
Curtain	(fig. 4)	131	884	318	304	264
Dolphin	(fig. 6)	1521	6858	3254	3271	3433

Figure 9: *Number of curves, geometric control points (P), left and right color control points (Cl, respectively Cr) and blur control points (Σ) for the images of this paper.*

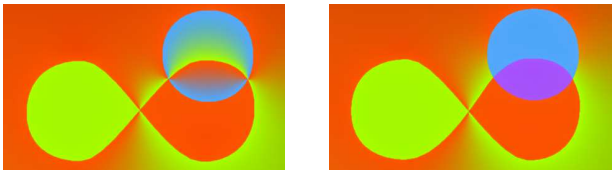
Usability: We believe that diffusion curves are a more natural drawing tool than gradient meshes. As mentioned previously, artists commonly use strokes to delineate boundaries in an image. Diffusion curves also allow an artist to evolve an artwork gradually and naturally. Gradient meshes, on the other hand, require careful planning and a good understanding of the final composition of the intended art piece. Most gradient mesh images are a complex combination of several individual — rectangular or radial — gradient meshes, often overlapping. All these decisions have to be made before the relevant image content can be created and visualized.

Topology: In some situations, the topology constraints of gradient meshes can be rather useful, for example when moving a gradient mesh to a different part of an image, or when warping the entire mesh. Such manipulations are also possible in our representation, but not as straightforward. For moving part of an image, the relevant edges have to be selected and moved as a unit. More importantly, without support for layering and transparency (see Section 6.2) it is difficult to ascertain how the colors of outer edges should interact with their new surroundings. A mesh warp could be implemented as a space warp around a group of edges.

6.2 Future challenges

Currently, our representation is single layered, but we are aware that multiple, independent layers offer more flexibility to the artist. To fully take advantage of a layered system, we need to address the interaction of multiple layers (considering a global Poisson solution), and the additional computational demands. Blending of layers would also require a notion of (gradual) transparency. Our current representation is more related to planar-maps [Asente et al. 2007] that model vector graphics in a single layer.

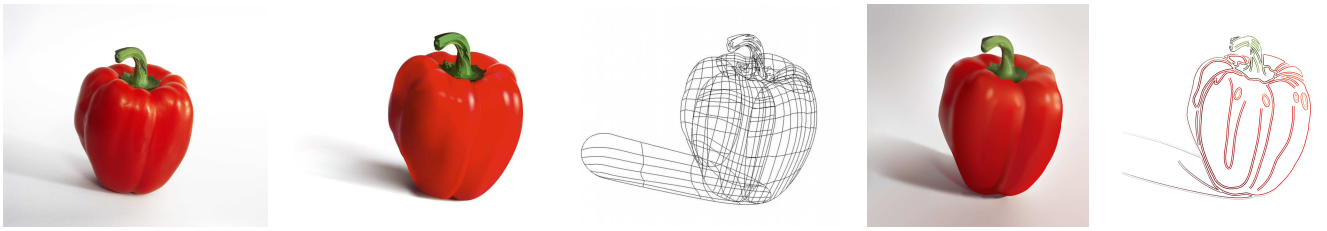
A different point worth improving is the way diffusion curves deal with intersections. Currently, diffusion curves present a specific (although predictable and meaningful) behavior: the colors attached to the two intersecting curves essentially compete with each other, which creates a smooth color gradient after diffusion (Figure 10(a)). If this default behavior is undesirable, the user can correct it by either adding color controls on each side of the intersection, or by splitting the curves in several parts with different colors (Figure 10(b)). Automating such behaviors would represent a powerfull tool for easing user interactions.



(a) (b)

Figure 10: *The default behavior of diffusion curves at intersections (a) can be corrected by curve splitting and color editing (b).*

Another limitation, common to all vector graphics, occurs in images or image regions that contain many small color or luminance variations, such as textures. In practice, most of the visual information of highly textured regions is captured by the automatic conversion, but imprecision occur when the texture is composed of many small structures (small compared to the distance d defined in Section 3.2.1). Moreover, the large amount of curves required to represent textures makes a vector representation inefficient and difficult to manipulate. Incorporating a diffusion curves version of texture synthesis tools is an interesting area of future research.



(a) (b) (c) (d) (e)

Figure 11: *Gradient Mesh comparison: (a) Original photograph; (b,c) Manually created gradient mesh (© Brooke Nuñez Fetissoff http://lifeinvector.com/), with 340 vertices (and as many color control points); (d,e) Our drawing created by manually tracing over the image; there are 38 diffusion curves, with 365 geometric, 176 left-color, and 156 right-color control points.*

Finally, as diffusion curves rely partially on a Poisson solution, nothing prevents their use in a variety of common Poisson-editing applications (e.g. for the transfer of image features such as shadows, or for merging multiple images in a same representation). We plan to extend our prototype in this direction in the near future.

7 Conclusion

We have introduced diffusion curves as a new image representation, offering most of the benefits usually found in vector approaches, such as resolution independence, exact editability, and compactness; while at the same time allowing to depict highly complex image content, generally only realizable with raster graphics. Diffusion curve images are comparable both in quality and coding efficiency with gradient meshes, but are simpler to create (according to several artists who have used both tools), and can be captured from bitmaps fully automatically.

Acknowledgments

Several people have been instrumental in giving this paper its final form. We would like to especially thank Laurence Boissieux and Philippe Chaubaroux for the time they spent testing our prototype and for the artworks they created with it. Particular thanks also to Amy Durocher for the design of the icons in our interface, and to Yann Boulanger for his "sauté de dinde". Finally we would like to thank the anonymous reviewers, and all the people from Adobe and INRIA for their constructive comments and feedbacks. Alexandrina Orzan is supported by a grant from the European Community under the Marie-Curie project MEST-CT-2004-008270.

References

ASENTE, P., SCHUSTER, M., AND PETTIT, T. 2007. Dynamic planar map illustration. *ACM TOG (Proc. of SIGGRAPH) 26*, 3, 30.

BERTALMIO, M., SAPIRO, G., CASELLES, V., AND BALLESTER, C. 2000. Image inpainting. In *Proc. of ACM SIGGRAPH 2000*, 417–424.

BERTALMIO, M., FORT, P., AND SANCHEZ-CRESPO, D. 2004. Real-time, accurate depth of field using anisotropic diffusion and programmable graphics cards. In *Proc. of 3DPVT*, 767–773.

BRIGGS, W. L., HENSON, V. E., AND MCCORMICK, S. F. 2000. *A multigrid tutorial (2nd ed.)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

CANNY, J. 1986. A computational approach to edge detection. *IEEE PAMI 8*, 6, 679–698.

CARLSSON, S. 1988. Sketch based coding of grey level images. *Signal Processing 15*, 1, 57–83.

DOUGLAS, D., AND PEUCKER, T. 1973. Algorithms for the reduction of the number of points required for represent a digitzed line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization 10*, 2, 112–122.

ELDER, J. H., AND GOLDBERG, R. M. 2001. Image editing in the contour domain. *IEEE PAMI 23*, 3, 291–296.

ELDER, J. H. 1999. Are edges incomplete? *International Journal of Computer Vision 34*, 2-3, 97–122.

GOODNIGHT, N., WOOLLEY, C., LEWIN, G., LUEBKE, D., AND HUMPHREYS, G. 2003. A multigrid solver for boundary value problems using programmable graphics hardware. In *Graphics Hardware*, 102–111.

KASS, M., WITKIN, A., AND TERZOPOULOS, D. 1987. Snakes: Active contour models. *International Journal of Computer Vision 1*, 4, 321–331.

KOENDERINK, J. J., AND DOORN, A. J. 1979. The internal representation of solid shape with respect to vision. *Biological Cybernetics 32*, 4, 211–216.

LECOT, G., AND LEVY, B. 2006. Ardeco: Automatic Region DEtection and COnversion. In *Proc. of EGSR*, 349–360.

LINDEBERG, T. 1996. Edge detection and ridge detection with automatic scale selection. In *Proc. of CVPR*, 465–470.

MARR, D., AND HILDRETH, E. C. 1980. Theory of edge detection. *Proc. of the Royal Society of London. Biological Sciences 207*, 187–217.

MCCANN, J., AND POLLARD, N. S. 2008. Real-time gradient-domain painting. *ACM TOG (Proc. of SIGGRAPH) 27*, 3.

ORZAN, A., BOUSSEAU, A., BARLA, P., AND THOLLOT, J. 2007. Structure-preserving manipulation of photographs. In *NPAR*, 103–110.

PÉREZ, P., GANGNET, M., AND BLAKE, A. 2003. Poisson image editing. *ACM TOG (Proc. of SIGGRAPH) 22*, 3, 313–318.

PRICE, B., AND BARRETT, W. 2006. Object-based vectorization for interactive image editing. *Visual Computer (Proc. of Pacific Graphics) 22*, 9, 661–670.

SELINGER, P. 2003. *Potrace: a polygon-based tracing algorithm*.

SUN, J., LIANG, L., WEN, F., AND SHUM, H.-Y. 2007. Image vectorization using optimized gradient meshes. *ACM TOG (Proc. of SIGGRAPH) 26*, 3, 11.

SUTHERLAND, I. E. 1980. *Sketchpad: A man-machine graphical communication system (Outstanding dissertations in the computer sciences)*. Garland Publishing, Inc., New York, NY, USA.

ZENZO, S. D. 1986. A note on the gradient of a multi-image. *Computer Vision, Graphics, and Image Processing 33*, 1, 116–125.