# Image-Based Capture and Rendering with Applications to Urban Planning

Alex Reche

**UNIVERSITE DE NICE-SOPHIA ANTIPOLIS - UFR Sciences**

Ecole doctorale : STIC

**THESE**

pour obtenir le titre de

**Docteur en Sciences**

de l'UNIVERSITE de Nice-Sophia Antipolis

Spécialité : Image de synthèse

présentée et soutenue par

**Alex RECHE MARTINEZ**

# IMAGE-BASED CAPTURE AND RENDERING WITH APPLICATIONS TO URBAN PLANNING

Thèse dirigée par George DRETTAKIS

soutenue le 19 avril 2005

**JURY:**

| | | |
|---|---|---|
| M. | Michel Cosnard | Président |
| M. | Wolfgang Heidrich | Rapporteur |
| M. | Pere Brunet | Rapporteur |
| M. | Pierre Poulin | Examinateur |
| M. | Souheil Soubra | Directeur de Recherche CSTB (CIFRE) |
| M. | George Drettakis | Directeur de These |

The dissertation of Alex RECHE MARTINEZ is approved:

Chair Date

Date

Date

Date

Date

Date

Université de Nice-Sophia Antipolis

4 2005

**Image-Based Capture and Rendering with Applications to Urban Planning**

# Résumé, Introduction et Conclusion en Français

## Résumé

Capture et Rendu à Base d'Images avec Application aux Projets Urbains

par

Alex RECHE MARTINEZ

Doctorat en Graphique par Ordinateur

Université de Nice-Sophia Antipolis

Cette thèse traite le problème de la capture des objets réels dans le but de les intégrer dans un environnement de réalité virtuelle et de les afficher (ou les rendre) d'une manière efficace. La thèse a deux parties : dans la première partie, nous présentons deux nouvelles méthodes pour la capture et le rendu des objets réels avec des techniques à base d'images, et dans la deuxième, nous présentons un système intégré de réalité virtuelle qui permet l'utilisation de plusieurs techniques réalistes pour la conception et l'évaluation de projets d'urbanisme. La première méthode proposée permet la capture d'objets réels avec une géométrie relativement simple, comme les façades de bâtiments par exemple. La capture se fait à partir d'un faible nombre de photographies. Avec cette méthode les objets capturés ont une représentation compacte ; le travail des infographistes est facilité pour permettre un affichage de meilleure qualité et enfin la méthode de rendu ne nécessite pas un découpage géométrique des objets, comme c'était le cas pour les méthodes précédentes. La deuxième méthode proposée traite le cas des objets complexes comme les arbres, à partir d'un ensemble de photographies prises autour de l'objet. Une première estimation de l'opacité est effectuée avec une méthode inspirée de la tomographie, ce qui donne une représentation volumique de l'arbre. Ensuite le détail de haute fréquence du feuillage est ajouté par un algorithme de génération de textures à partir des images d'origine. Le résultat est l'affichage très réaliste des arbres réels capturés, qui peuvent être insérés dans des environnements virtuels. Le système de réalité virtuelle présenté dans la deuxième partie de la thèse intègre des méthodes d'affichage à base d'images, ainsi que d'autres algorithmes d'affichage réaliste (ombres, son 3D, etc.). Le système multi plateforme est opérationnel sur des systèmes de réalité virtuelle allant du plan de travail virtuel à l'Inria Sophia-Antipolis jusqu'à la salle RealityCenter du CSTB. Notre travail dans le cadre d'un projet réel, celui du Tramway de Nice, nous a permis d'évaluer nos choix sur le réalisme et l'interface du système. Plusieurs essais ont été effectués avec les architectes responsables du réaménagement de la Place Garibaldi à Nice, et nous présentons ici les premières conclusions de nos observations.

# Introduction

Être capable de facilement créer et afficher un environnement virtuel (EV) représentant une version modifiée d'un endroit existant serait extrêmement puissant et utile. Cela pourrait par exemple être utilisé pour montrer le résultat d'un nouveau projet urbain ou pour reconstruire un monument actuellement détruit.

Pour ce faire, nous commençons par capturer l'environnement réel où la simulation va être faite. Cet environnement doit être affiché ou rendu, dans un environnement immersif de réalité virtuelle. De nos jours, la capture et le rendu de ce genre de scènes, basées sur des sites réels, relèvent d'un véritable défi, lié à la complexité du monde réel. Le plus simple des objets dans le monde réel est complexe si nous le regardons en détail. Un des aspects source de cette complexité est l'interaction de la lumière entre différents objets. Toute cette complexité doit être prise en compte quand nous essayons de simuler un scénario réel, si nous voulons arriver à donner une véritable impression de réalisme.

De nombreuses recherches sur la création de scènes très réalistes ont été effectuées, en utilisant des algorithmes basés sur des simulations physiques ou en utilisant des images. Seules quelques-unes d'entre elles ont essayé l'intégration de ces deux aspects : La capture d'un environnement réel et le rendu réaliste pour créer un environnement virtuel photo-réaliste. Les algorithmes basés sur des simulations physiques sont très rarement utilisés dans ce genre de projets, car le temps de calcul nécessaire pour obtenir de bons résultats rend impossible son utilisation dans des applications de rendu en temps réel. Un autre problème est la complexité de réalisation liée à la quantité des paramètres nécessaires, qui sont à la fois difficiles à utiliser et à comprendre.

Ces dernières années, beaucoup de recherches sur le rendu à base d'images ont été développées. Ces algorithmes sont d'une utilisation très simple et les résultats sont impressionnants, grâce à l'utilisation des véritables images. En revanche, aucune de ces recherches n'a testé la combinaison de différentes techniques dans une vraie chaîne de travail pour créer un environnement complet.

Le travail réalisé au cours de cette thèse a été financé par une bourse doctorale CIFRE [1]. Le but de ce type de bourse est de promouvoir la recherche au sein de l'entreprise. Cet aspect se reflète dans le travail expliqué ci-après, et particulièrement dans les travaux de recherche et développement utilisés pour un projet réel. Ce projet est très proche d'une application réelle, tout en gardant des aspects de recherche larges et très difficiles dûs à la difficulté des environnements simulés. Ce projet réel a été source d'inspiration de tous les travaux de recherche faits en parallèle à son développement.

Cette thèse comporte deux parties : La première présente le développement de deux nouveaux algorithmes qui facilitent la capture de scènes réelles et leur rendu en temps réel. Chacun de ces algorithmes est adapté à un type d'objets souvent rencontré dans des scènes d'extérieur. La deuxième partie concerne l'intégration d'une série de différents algorithmes de rendu réaliste dans le contexte du projet CREATE EU IST [2], et son évaluation dans un projet d'urbanisme réel.

Dans la première partie de la thèse, nous présentons deux techniques très différentes pour la capture et le rendu à base d'images. La première technique concerne la capture et le rendu des environnements extérieurs en utilisant des images où la géométrie est relativement facile à reconstruire. Ce travail est issu du projet CREATE, qui a deux applications en vue : la planification urbaine et l'héritage culturel. Pour la première application, nous avons étroitement collaboré avec les responsables du projet du nouveau tramway de Nice (France). Pour la deuxième, nous avons travaillé avec la Fondation du Monde Hellénique, qui désire montrer des temples anciens dans leur lieu d'origine tel qu'il est aujourd'hui. Dans ce contexte, nous avons capturé la place Massena, la place Garibaldi de Nice et les ruines du temple de Asclepius à Messene, Grèce. Cet algorithme pour la capture des environnements extérieurs est basé sur la décomposition de chacune des images d'entrée en une série de calques, en utilisant la connaissance du modèle 3D de la scène. Chacun de ces calques représente un niveau de visibilité : Les objets qui apparaissent au premier calque sont les objets complètement visibles, les objets assignés au deuxième calque sont des objets partiellement ou complètement cachés par les objets du premier calque, et ainsi de suite. Cette décomposition des images originales en calques permet aux artistes ou modeleurs de facilement éditer les images pour compléter l'information manquante à cause des occlusions. Les détails de cette technique sont

[1] Cette convention est cofinancée par le Ministère de la Recherche et le CSTB (Centre Scientifique et Technique du Bâtiment, www.cstb.fr) et animée par l'ANRT (Association Nationale de la Recherche Technique) pour le compte du ministère de la Recherche). La bourse CIFRE est un contrat entre le thésard, un laboratoire de recherche, l'INRIA, et une entreprise (pour cette thèse, le CSTB

[2] CREATE est un projet de 3 ans, RTD project funded by the 5th Framework Information Society Technologies (IST) Programme of the European Union (IST-2001-34231), http://www.cs.ucl.ac.uk/create/

présentés dans le chapitre 3. Dans tous ces environnements nous avons décidé de ne pas capturer ou modeler les objets complexes tels que les arbres, car les algorithmes utilisés n'étaient pas adaptés. La deuxième contribution de cette thèse est un algorithme pour la capture et le rendu de ces objets complexes. Nous nous sommes concentrés sur la capture et le rendu d'arbres en utilisant des images. L'utilisation d'images était nécessaire pour les besoins de réalisme de nos rendus. Cet algorithme a deux phases. Dans la première phase, l'algorithme prend comme entrée : une série de photographies prises autour d'un arbre, la position relative de chacune de ses photographies par rapport aux autres et une image avec des informations de transparence pour chaque photographie. Cette information de transparence nous permet séparer l'arbre du reste de l'image (le fond). Grâce à ces informations l'algorithme génère une première version volumétrique de l'arbre en utilisant seulement la transparence. Dans la deuxième phase, l'algorithme met de la couleur à la représentation volumétrique de l'arbre en utilisant les photographies originales, générant ainsi des textures dépendantes du point de vue pour chaque cellule du volume. Ces textures sont très importantes pour ajouter au modèle volumétrique le détail de hautes fréquences produit par le feuillage de l'arbre. Le résultat est un modèle volumétrique d'arbre avec des textures dépendantes du point de vue qui peut être rendu en temps réel. Les détails de cet algorithme sont expliqués dans le Chapitre 4.

Dans la deuxième partie de la thèse nous étudions l'application du photo réalisme dans un projet réel. Mais auparavant, nous avons besoin de répondre à un nombre important de questions. Qu'est-ce que le " Réalisme " dans un environnement virtuel? S'agit-il seulement d'images réalistes? Le réalisme est donné par des images photo-réalistes, mais aussi par des animations réalistes, par le son spatialisé en 3D, par les personnages 3D qui donnent vie à l'environnement virtuel.

Mais quelle est l'importance du réalisme dans les simulations? Est-il vraiment utile d'avoir un environnement extrêmement réaliste quand nous examinons la nouvelle configuration d'une rue ou d'une place, ou quand nous regardons un ancien monument aujourd'hui inexistant dans son emplacement original?

Il est difficile de répondre à ces questions et d'évaluer l'importance du réalisme des simulations, étant donné la quantité de facteurs qui l'influencent.

Il existe quelques recherches à propos de l'évaluation de la perception, mais peu sur un système complet intégrant un environnement photo réaliste augmenté avec son, arbres, population, etc. Nous avons participé à la création de ce système complètement opérationnel, qui a été utilisé dans le contexte d'un réel projet de planification urbaine.

Dans le contexte du projet CREATE, nous avons travaillé en étroite collaboration avec des architectes et archéologues. Ils nous ont donné leurs impressions pendant toute la durée du

processus de développement ainsi que la chance d'évaluer l'importance du réalisme dans les simulations. Nous avons concentré nos efforts, dans le cas de la planification urbaine, sur les travaux faits avec les architectes responsables du projet du Tramway de Nice. Nous avons créé un environnement photo-réaliste avec un des scénarios possibles de configuration de la place Garibaldi avec le tramway. Les architectes pouvaient alors avoir un aperçu des différentes solutions. Dans cette thèse, nous avons participé au développement d'une nouvelle interface et à l'évaluation de l'importance du réalisme dans un environnement virtuel utilisé par des architectes dans leur chaîne de travail.

Cette thèse est structurée comme suit. Dans le prochain chapitre nous allons présenter les travaux précédents en Modélisation et Rendu à Base d'Images (MRBI) qui sont le point commun à toutes les recherches. Chacun des chapitres suivants contient une petite section avec des travaux précédents spécialisés. Dans le Chapitre 3, nous présentons la nouvelle technique de texturation de scènes 3D en utilisant des calques d'images dépendants du point de vue. Dans le Chapitre 4, nous expliquons la nouvelle approche pour la capture et le rendu d'arbres à partir de photographies. Dans le Chapitre 5, nous décrivons comment nous avons fait pour créer l'environnement de réalité virtuelle photo-réaliste augmenté. Enfin, ce système a été utilisé pour évaluer l'importance du réalisme dans les simulations. Le processus d'évaluation est présenté dans le Chapitre 6, juste avant la conclusion.

# Conclusion

La thèse est divisée en deux parties complémentaires. Dans la première partie nous avons présenté deux nouvelles techniques pour la Modélisation et le Rendu à Base d'Images : Une pour les objets relativement simples et une autre pour les arbres. Comme mentionné dans l'introduction, cette thèse est financée par une bourse CIFRE, qui encourage le développement de problèmes réels. Nous avons vu ici une opportunité de développement de nouvelles recherches pour des problèmes réels. En particulier, les deux techniques mentionnées précédemment ont été directement inspirées par les travaux et les problèmes rencontrés pendant le développement du projet de planification urbaine du Tramway de Nice, dans le contexte du projet CREATE EU IST. Dans la deuxième partie nous avons présenté la partie application de la thèse, qui est plus orientée système et développement pour la création d'un environnement virtuel utilisable et son évaluation.

L'avantage de travailler dans un projet réel est que nous sommes confrontés à des problèmes réels, lesquels sont plus complexes et relèvent d'un plus grand défi que les exemples utilisés normalement pour illustrer des nouvelles idées de recherche. Nos recherches ont été dirigées et inspirées par ces problèmes, mais elles ont quand même un point commun, la Modélisation et le Rendu à Base d'Images qui ont été nos taches principales pendant la réalisation du projet CREATE.

Un des problèmes trouvé au début du projet était la texturation des modèles reconstruits en utilisant les images d'origine. L'extraction de textures se pressentait très consommatrice en ressource mémoire, par le dépliage des textures dans l'espace objet, et en ressource temps, nécessaire par l'artiste pour effacer tous les artifices des images. Cela nous a conduit à chercher une solution alternative : Les Textures à Calques Dépendantes du Point de Vue. Cette nouvelle approche résout les principaux problèmes trouvés dans le travail de Debevec ( [17]), travail qui correspondait le plus à nos besoins.

Cette nouvelle approche résout le problème de la subdivision géométrique et le remplissage des régions sans texture. Ce remplissage était fait automatiquement sans donner l'opportunité

à l'utilisateur de modifier le résultat et produisait des résultats flous et de mauvaise qualité. Le principal problème étant l'occlusion, Debevec l'a résolu en subdivisant la géométrie et en remplissant les régions sans information par une interpolation de couleurs automatique. Dans notre nouvelle approche, nous avons résolu le problème au niveau image, et non au niveau géométrie. Nous divisons l'image en un groupe de calques correspondant aux différents niveaux d'occlusion. De cette façon, la géométrie n'est pas modifiée et les manques d'informations dues à l'occlusion peuvent être résolues en éditant les calques des images résultat. Etant donné que le résultat de l'algorithme est une image avec des calques tout à fait standard, l'algorithme peut être facilement intégré dans un logiciel de Modélisation à partir d'images, et le résultat directement édité dans un outil d'édition d'images. De plus, étant donné la connaissance de l'information de profondeur grâce au model 3D, celle-ci peut être intégrée dans le programme d'édition d'images afin d'utiliser des techniques comme celles présentées par Oh [66].

Cependant, des problèmes subsistent. - La géométrie, par exemple, doit être la plus précise possible. A l'heure actuelle, même en utilisant des logiciels commerciaux comme celui de RealViz ou Façade, le temps nécessaire pour avoir un modèle précis est considérablement long comparé au reste du processus. - Le remplissage des parties des images manquantes doit aussi être fait manuellement. Cette nouvelle approche donne le contrôle total à l'artiste pour ajouter ou effacer de l'information, mais elle manque de fonctionnalités pour aider le remplissage. Il existe néanmoins un nombre important de solutions à ce problème, donnant à l'artiste la possibilité de remplir l'information automatiquement et d'accepter ou de refuser le résultat. Cette information peut, par exemple, être récupérée d'une image à une autre. Puisque nous connaissons la géométrie, l'information d'une image peut être copiée/collée dans une autre en déformant la partie copiée pour la placer directement au bon endroit. Une autre solution est l'utilisation d'un algorithme de Inpainting [7, 25] pour remplir les parties manquantes automatiquement, mais toujours en laissant le contrôle à l'artiste pour corriger ces régions si nécessaire. Une implémentation du travail de Drori et al. [25] serait vraiment utile si adaptée pour être guidée par des informations provenant des autres images.

Un autre problème rencontré pendant le développement du projet CREATE a été la capture et le rendu des arbres. Dans le contexte des projets de planification urbaine, la capture d'arbres existants devient nécessaire. Garder la forme et l'aspect des vrais arbres est très important. Nous avons développé une technique inspirée de la tomographie (technique utilisée dans l'imagerie médicale) pour la reconstruction d'un volume d'opacités, qui est ensuite augmenté en utilisant des panneaux d'affichage texturés. Cette reconstruction reproduit exactement l'arbre capturé quand regardé

depuis les mêmes points de vue que les photos d'origine et génère des interpolations plausibles pour les vues intermédiaires. Le volume résultat peut être utilisé dans des applications de rendu temps réel.

Comparé aux techniques existantes pour la modélisation d'arbres, notre méthode est capable de créer une représentation volumétrique d'un arbre existant et nos résultats sont immédiatement adaptés pour un rendu interactif. Le temps nécessaire pour capturer un arbre est relativement faible, comparé au temps nécessaire pour créer un modèle équivalent avec un logiciel de création 3D standard comme Maya ou 3DSMax. Notre technique de capture peut aussi être utilisée dans d'autres contextes, par exemple la recherche en Physiologie des Arbres, où souvent les arbres sont capturés en utilisant un senseur magnétique de position, ce qui implique beaucoup de travail manuel.

Un des problèmes de notre méthode est le manque d'information sémantique. Puisque nous reconstruisons un volume, nous ne savons pas où se trouvent le tronc, les branches et les feuilles. Cela peut être un travail futur très prometteur, puisque cette information sémantique peut être utilisée : - Pour analyser la structure de l'arbre pour en régénérer des nouveaux exemplaires avec une configuration différente de branches et feuilles. - Pour accélérer le rendu en simplifiant le modèle pour les parties solides de l'arbre. La création d'un modèle hybride géométrie/volume serait une solution possible.

Un autre problème est l'énorme consommation de mémoire texture que notre algorithme utilise. Toutefois, nous croyons fortement que cette information de textures peut être facilement comprimée avec une technique de PCA (Analyse en Composantes Principales). Les compagnies de jeux vidéo qui ont vu les résultats ont été très intéressées par la qualité des arbres, mais aussi découragées par la mémoire utilisée, qui dépasse de beaucoup les limites acceptées dans ce genre d'applications.

Le rendu des arbres peut aussi être optimisé en utilisant la structure hiérarchique utilisée dans l'algorithme. Une fois les panneaux d'affichage des plus bas niveaux de la hiérarchie calculés, nous pouvons facilement générer des panneaux d'affichage intermédiaires pour tous les autres niveaux. De cette façon un algorithme de niveau de détail peut être facilement implanté.

Nous avons été contactés par des biologistes qui ont été vraiment intéressés par notre méthode qui permet de rapidement capturer un arbre et d'en prendre toutes ses mensurations. Il existe des méthodes similaires dans le domaine de recherche de la Physiologie des Arbres, comme celui présenté par Phattaralerphong et al. [71]. Nous croyons que notre approche peut être utilisée dans cette voie aussi, avec des résultats d'une qualité bien supérieure dans ce contexte.

Pour finir, nous avons présenté la création et l'évaluation d'un Environnement Virtuel

à Base d'Images utilisé pour un projet réel. Cette partie de la thèse est une partie très orientée application. Nous espérons que les choix faits pour la construction du système, seront peut être utilisés dans d'autres projets du même genre.

Etant données les limitations de temps pour le projet CREATE, les deux techniques développées dans la première partie de la thèse n'ont pas été ajoutées dans le prototype utilisé pour l'évaluation de l'environnement virtuel avec les architectes. En revanche, nous avons intégré deux autres techniques avancées (les textures dépendantes du point de vue et l'éclairage par ciel virtuel) dans ce prototype comme exercice d'application des recherches dans une application réelle d'environnement virtuel. La méthode utilisée dans ce projet devrait nous permettre relativement facilement l'intégration future de nouvelles techniques.

En observant le développement du système de réalité virtuelle, nous constatons que la plus grande contrainte nous a été imposée par l'utilisation de vieilles machines existant dans des installations telles que la CAVE ou le Reality Center. Ces systèmes utilisent Performer sous IRIX pour optimiser le rendu dans de multiples écrans et pipelines de rendu. Etant donnée l'évolution des systèmes et des dispositifs graphiques, l'utilisation d'un système moderne augmenterait la qualité des environnements virtuels que nous avons développé dans ce contexte d'applications. Même si nous pensons que la plupart des choix faits (portabilité, abstraction, utilisation de scripts pour l'extensibilité, etc.) seront encore valides dans un nouveau contexte, certains auraient pu être différents.

L'évaluation a été faite avec un petit nombre d'utilisateurs, néanmoins nous considérons optimums les résultats issus des observations faites dans leur environnement de travail. Le système a été utilisé pour la prise de décisions dans le véritable projet, et les retours ont été très positifs. En utilisant une approche de design participatif et un processus d'évaluation précis, nous pouvons développer un environnement virtuel qui répond aux vrais besoins des utilisateurs finaux. Quelques problèmes ont été décelés par certains utilisateurs, comme la présence de l'interface avec des menus seulement dans la vue aérienne de la scène, ou le manque de précision dans l'emplacement des objets. Toutefois ces problèmes peuvent être résolus grâce à du développement additionnel. Nous avons pu noter que l'utilisation de multiples vues est très appréciée par les utilisateurs, spécialement la vue du balcon, qui a été jugée comme très utile. Nous nous attendions à ces résultats, puisque ces vues ont été directement inspirées par les sketches réalisés par ces mêmes utilisateurs. Cela montre aussi la flexibilité des environnements virtuels comparés aux designs par sketches. Les utilisateurs ont jugé le réalisme très important. La végétation, les ombres et le son ont été marqués comme étant très importants pour évaluer la qualité du design. Les foules sont aussi des éléments importants pour

évaluer les facteurs d'échelle entre les objets et pour donner de la vie à l'environnement.

Nous croyons que cette approche est très prometteuse pour le développement d'environnements et d'outils qui peuvent être d'une véritable utilité pour les utilisateurs.

En conclusion, nous avons présenté dans cette thèse deux nouvelles contributions aux techniques de Modélisation et de Rendu à Base d'Images, qui ont été inspirées par les problèmes trouvés dans le contexte d'un vrai projet de planification urbaine. Nous avons aussi présenté le développement d'un système complet pour l'affichage d'environnements virtuels réalistes, ainsi que son évaluation. Le fait que le travail ait été dirigé dans le contexte d'un projet réel est une contrainte très importante, mais aussi une source d'inspiration très riche.

# English version

## Abstract

Image-Based Capture and Rendering with Applications to Urban Planning

by

Alex RECHE MARTINEZ
PhD in Computer Graphics

Université de Nice-Sophia Antipolis

This thesis addresses the capture real objects to be integrated in a virtual environment and their efficient display. The thesis has two parts; in the first part, we present two new methods for the capture and rendering real objects using image-based techniques, and in the second part, we present an complete virtual reality system that allows the use of different realistic techniques for the design of urban projects, as well as a first evaluation of this system. The first method proposed allows the capture of a real scene with relatively simple geometry, for example building façades. The capture is done using an small number of photographs. With this method the captured objects have a compact representation; the work of the computer graphics artist is simplified to allow higher quality render and the rendering method does not need geometric subdivision as in previous methods. The second method proposed is for the capture of complex objects, in our case trees, using as input a series of photographs taken around the object. A first estimation of opacities is done with a method inspired by tomography, that give us a volumetric representation of the tree. The high frequency detail of the leaves is added by an algorithm of texture generation using the original images. The result is a very realistic rendering of real captured trees, that can be added in a virtual environment. The virtual reality system presented in the second part of the thesis integrates image-based rendering methods, and other algorithms for the realistic rendering (shadows, 3D sound, etc.). The multi platform system works on systems such as the Workbench of the INRIA Sophia-Antipolis or the RealityCenter of the CSTB. Our work on this real world project, the construction of the Tramway of Nice (France), allowed us to evaluate the choices on realism and the interface of the system. The architects in charge of the Tramway project on the Garibaldi square in Nice performed a set of test, and we present here the first conclusions of our observation.

# Contents

# List of Figures

## Acknowledgments

The first author is funded by a CIFRE doctoral fellowship, in partnership with the Centre Scientifique et Technique du Batiment (http://www.cstb.fr). This research presented in this theses were partially funded by the EU IST project CREATE, IST-2001-34231, http://www.cs.ucl.ac.uk/create. ImageModeler$^{TM}$ was gracefully provided by REALVIZ in the context of the CREATE project. We thank Alias|Wavefront for the generous donation of Maya. Thanks to our lab artist Alexandre Olivier-Mangon for his helpful comments and his help for the modelling and capture. Thanks to Frédo Durand for important input, and Pierre Poulin for comments. Thanks also to all my co-authors of the papers published during the duration of this thesis Ignacio Martin [79], Maria Roussou [80] [23]. Thanks are due in particular to Maria Roussou who wrote a major part of [23], which served as the basis of Chapter 6. Thanks to Audrey Bzeznik for supporting the first author and for feeding the troops. Thanks also to the various “tree capture” teams !

# Chapter 1

# Introduction

It would be extremely powerful and useful to be able to easily create and display a virtual environment (VE) based on a modified version of an existing location. This could be used for example in a new urban planning project, or the reconstruction of a destroyed monument.

To be able to do this, we need to start by capturing the real environment where the simulation will be performed, then this environment needs to be displayed, or rendered, in an immersive virtual environment. The capture and rendering of such real scenes is still a challenging task because of the complexity of the real world. Even the simplest object in a real world scene is complex if we consider it in detail; one of the most complex aspects is the light interaction between objects. All this complexity must be taken into account when simulating a real scenario if we want to achieve truly realistic impression.

A lot of research has been done to create very realistic scenes, using physically- or image-based algorithms, but few of them have tried to integrate both real-world capture and realistic rendering to create a photo realistic virtual environment. Physically-based algorithms are not often used in real projects, because these algorithms are time consuming and thus not adapted for use in real-time applications. An additional problem is that often the parameters used to set-up the effects are hard to use and understand.

In recent years, there has been much research on image-based algorithms because these algorithms are easy to use and they give impressive results. However, none of them have truly tried to combine different techniques in a real workflow to create a complete environment.

The work done during this thesis has been funded by a CIFRE doctoral scholarship [1].

[1] This convention is co-funded by the Research Ministry and the CSTB (Centre Scientifique et Technique du Batiment, www.cstb.fr) and managed by the National Association of the Technical Research (ANRT).

The CIFRE scholarship is a contract between the PhD student, a research laboratory, INRIA and a company, the CSTB for this thesis. The goal of these scholarships is to promote both academic excellence and a string application-driven component. This is reflected in the work reported here, and in particular in the research and development of the virtual environment used for a real world project. This project is at the same time very close to a real-world application and very challenging research-wise because of the difficulty of the simulated environments. The real-world project has been source of inspiration of all the research work done in parallel with the more directly application-related development.

This thesis has two parts: The first is the development of two novel algorithms which facilitate capture and rendering of real scenes in real-time. Each one of these algorithms is adapted to a class of objects appearing frequently in outdoors scenes. The second part concerns the integration of several different realistic rendering approaches in the context of the CREATE EU IST[2] project, and its evaluation in a real-world urban-planning project.

In the first part of the thesis, we concentrated our efforts on two very different approaches. The first main contribution of this thesis is on capture and rendering of outdoors environments using photographs, where the geometry is relatively easy to model. This work was in the context of CREATE, which had two target applications: urban planning and cultural heritage. In the first application we collaborated closely with the real world project of the new Tramway construction in the city of Nice (France). In this context, we captured the Massena Square and the Garibaldi Square in Nice and the ruins of the temple of Asclepius in Messene, Greece. The algorithm for the capture of outdoor environments is based on the decomposition of each input image into a series of layers using knowledge of the 3D information of the scene. Each layer represents a level of visibility, this means that objects on the first layer are objects completely visible, objects on the second layer are partially or completely hidden by objects on the first layer and so on. This decomposition of the original images in layers allows an artist or modeller to easily edit the images so that pixels of occluded objects can be edited. The details of the approach are explained in detail in Chapter 3.

In all these environments we decided not to model complex objects such as trees, because the algorithms used were not adapted. The second main contribution of this thesis in on capture and display of such complex objects. We concentrated our efforts on the capture and rendering of trees using photographs. The use of photographs was needed to give the required realism to our renderings. This algorithm has two phases. A first phase takes as input a series of pictures around a

[2]CREATE is a 3-year RTD project funded by the 5th Framework Information Society Technologies (IST) Programme of the European Union (IST-2001-34231), http://www.cs.ucl.ac.uk/create/

tree, the relative position of the cameras when the pictures were taken, and the alpha information that allows the separation of the tree from the background and generates a first version of a volumetric tree using only opacities. The second phase consists in giving a color to the volumetric tree using the original color images, assigning view-dependent textures, thus providing the indispensable high frequency detail of the leaves of the tree. The resulting volumetric tree can be rendered in real time. The details of this algorithm can be found in Chapter 4.

In the second part of the thesis we examine the application of photorealism in a real-world project. Before doing this, we first needed to ask a number of important questions. What is "realism" in a virtual environment? Is it only photo-realistic images? Realism is given by photo realistic images, but also by realistic animations, spatialized sound, simulated people who give life to the virtual environment, shadows, etc. In our approach we wanted to create a realistic environment. Such a fully integrated approach was made possible in the context of the CREATE project, which was a multidisciplinary effort, allowing us to add all these aspects to our simulations.

But how important is the realism in simulations? Is it really useful to have a very realistic environment when examining the new configuration of a street or a square or when looking at an ancient monument in his real location?

These are questions which are hard to answer. It is very difficult to evaluate the importance of realism on simulations because of the number of factors influencing it.

There is some research in the evaluation of perception, but little has been done on a complete system integrating a photo realistic environment enhanced with sound, trees, crowds, etc. We participated in the creation of a complete operational system, which was used in the context of a real-world urban planning project, as part of the CREATE project.

In the context of the CREATE project, we worked closely with architects and archaeologists who gave us their impressions during the entire process and gave us the chance to evaluate the importance of realism on the simulations. We concentrated our efforts on the urban planning case, in our work with the architects working on the real project of the future Tramway in Nice. We created a photo realistic environment with one of the scenarios simulating the future configuration of the Garibaldi square with the tramway. The architects were then able to see the effect of different solutions. In this thesis, we participated in the development of a novel interface and an evaluation study examining the importance of realism and the use of virtual environments for the architect's workflow.

This thesis is structured as follows. In the next chapter we will introduce previous work on Image Based Modeling and Rendering (IBMR) that is the common point of this research. Each one

of the following chapters will contain a small previous work section introducing specialized related work. In Chapter 3 we will explain a new technique to texture a 3D scene by introducing layered view-dependent texture mapping. Then in Chapter 4 we will explain a new approach to capture and render photo realistic trees using photographs; this technique has been inspired by techniques used in medical imaging. In Chapter 5 we will describe how we built a photo-realistic virtual environment enhanced with sound. The resulting system will be used to evaluate the importance of realism on simulations. The process of evaluation will be presented in Chapter 6. In the last chapter we conclude.

# Chapter 2

# Previous Work

In this chapter we discuss the most important related work in image-based modelling and rendering (IBMR), which is both the common focus of all the research in this thesis, and its main source of inspiration.

The complexity of the real world is far greater than the capacities of current hardware for real-time rendering. Any object can be modelled as geometry, but the geometry needed is usually much too complex to be displayed in real-time and it's creation by a modeller or an artist is very long and tedious. The interactions between objects and light are more complex and contribute significantly to give the impression of realism. These interactions are simulated with time and memory consuming algorithms like photon mapping [38] or radiosity [90]. Recently a lot of research has been done to simplify all this complexity by directly using images of the real geometry or environment being used. These are called Image Based Modeling and Rendering (IBMR) techniques.

The idea of IBMR is to use photographs of an object to generate new views of it. In this way we can have very realistic representations of objects in a cost-effective and efficient way. Using this kind of technique we are not limited by the complexity of an object but by the number of original images.

Leonardo da Vinci said: *The body of the air is full of an infinite number of radiant pyramids caused by the objects located in it. These pyramids intersect and interweave without interfering with each other during the independent passage throughout the air in which they are infused.* Kemp [40]

As da Vinci said, the space is filled of pyramids representing all possible rays of light. Imagine that we take a picture with a black and white camera; this picture will record the intensity distribution $P$ coming from any direction $P(\Theta,\Phi)$. If this camera is a color camera, then this func-

tion will take into account the different frequencies of light $P(\Theta,\Phi,\lambda)$. If it is a video camera, it will also record the time $P(\Theta,\Phi,\lambda,t)$. Finally if it is holographic it will record all possible directions $P(\Theta,\Phi,\lambda,t,V_x,V_y,V_z)$. This function is the Plenoptic Function [2] and it represents all these possible rays filling space. Using this function to generate new views consists simply evaluating it for different values of its parameters.

This function has been a source of inspiration of a lot of research in computer graphics and vision and, in a certain way, it is the link between these two research fields.

Of course, the information needed to represent this seven-dimensional function is impossible to capture. However many simplifications of this function have been proposed to make it usable. The most frequent simplifications are the elimination of the frequencies, that assume the world to be monochromatic, and time, assuming that what we capture is an instant of time. The elimination of time has as side effect; we don't take into account changes of light conditions and changes in geometry (animation). Even discarding these two dimensions, the resulting five-dimensional function is very difficult to capture. We will discuss some of the possible simplifications introduced in the previous research that are related to our work.

The most important simplification is the one used to create panoramic images used by Chen et al. [13]. A panoramic image is a 360 degrees image with a unique focal point. By fixing the focal point we eliminate three of the dimensions of the plenoptic function and we keep only the two spherical co-ordinates that give us the direction of each pixel. Creating a new image consists simply in sampling the plenoptic function in all possible directions. The problem here is that not only have we simplified the plenoptic function but we have also eliminated the freedom of translation. We can only rotate the viewpoint or zoom into the image.

If we want to translate the viewpoint we need to find an approximation to the plenoptic function at this new point. This could be done giving a depth to each pixel; knowing the depth of every pixel in an image is equivalent to knowing the world space co-ordinates of the point of geometry represented by the pixel. The easiest assumption is to give the same depth to all pixels: this means that we project the image into a sphere with it's center in the original viewpoint. We can then approximate the plenoptic function for a new viewpoint. Of course, new views using these depth values will be completely distorted and would not look very realistic. More sophisticated depth maps can be used to reproject the images. Such depth maps can be captured using an laser scanner, edited by hand [66] or given as an approximation of the geometry [17], for example. Adding per-pixel depth transforms an image from a 2D into a 2.5D representation and makes possible the de-projection of the image into the 3D space to be able to re-project it into the 2D space of a another

viewpoint. The way all these de-projections and re-projections are performed depend on the IBMR pipeline.

As said before, giving a depth value for each pixel of this image allows the re-projection of this image to another viewpoint; the pixels can then be considered as 3D points. If we look at the image from an oblique angle many holes appear for the parts that were missing in the original image, because they were occluded or simply because they were outside the view. Finding a way to fill these holes in reconstruction is often referred as the "hole-filling problem".

A possible solution for this problem was proposed by Shade et al. [87] where each pixel of the image contains not a unique value of color and depth but many of them. Such Layered Depth Images (LDI) contain enough information to be able to fill some holes, but they are difficult to construct. The same multi layer approach is also used by Oh et al. [66], where they present techniques to make depth edition easier. They take advantage of this depth information when editing the image by doing perspective correction when copying parts of the images from one place to another for example.

There are three different techniques to re-project an image with depth information to a new viewpoint. These are: polygon rendering, forward 3D image warping (or pre-warping), and backward 3D image warping (or post-warping). We will introduce each one of them.

Polygon rendering considers the image with the corresponding depth as 3D geometry that is passed directly to the graphics hardware to be rendered. If pixels are considered as points in 3D then the hole problem is ignored and it is visible. Some systems use a 3D mesh by connecting together neighboring pixels [54]; drawing triangles between pixels partially solves the "hole-filling problem".

The warping technique uses the assumption that knowing the relative position of the two viewpoints, we can build a 4x4 matrix (called the fundamental matrix [44]) that establishes a correspondence between the pixels of the two viewpoints. This matrix can be applied in any sense, from the original viewpoint to the new one or vice versa. Given an original image we can forward warp the pixels of this image to their new location at the new viewpoint, or we can backward warp each pixel of the new viewpoint to find which pixel of the original viewpoint is needed.

Forward 3D image warping is the technique used by McMillan et al. [59] where new cylindrical views are generated using the disparity maps created using a series of cylindrical reference images. Forward warping is also used by Oliveira et al. [67] where the images used as textures are pre-warped using the depth information and the current viewpoint to give the impression of "relief" even if the textured polygons are simply planes. This technique considers the pixels with a

unique color and depth value, so the “hole-filling problem” is visible. An extension was presented in the work of Parilov et al. [69] where different layers are added to the relief textures partially solving the hole-filling problem. This technique also changes the way of reprojecting images by doing post-warping and making it possible to directly warp on the graphics hardware.

Another kind of IBMR technics are what we call “mesh based representations” that use an approximate model of the geometry viewed in photographs. Devebec et al. [18, 17] presented a rendering algorithm called view-dependent texture mapping (VDTM) that uses a single geometry model and a series of photographs around the object. In practice, the three nearest viewpoints are chosen and the corresponding images are projected and blended onto the geometry. Each surface of the model needs to be completely visible or completely invisible to each camera, that means that polygons which are partially visible in one of the input images are subdivided and polygons with missing textures (not visible from any camera) are filled by simple color interpolation. This approach can have numerical problems in some contexts during the polygon subdivision. The fact that the method is completely automatic makes that it lacks of control on polygons with missing textures. These problems are solved in a new approach presented on Chapter 3

In the work presented by Pulli et al. [76], models captured from different view points are used to display the object. In this work different views around an object are chosen. For each view a color image and a scanned model are captured. During rendering the three nearest viewpoints are used to create the new virtual viewpoint; the three scanned models corresponding to these three viewpoints with their corresponding colors are blended. However popping effects appear when changing the viewpoint.

Last, but not least, the Light Field [48] and the Lumigraph [34] are techniques that densely sample the plenoptic function. These samples are then used to generate new views by interpolating between samples. These techniques take advantage of the following ray law: the radiance in the direction of a light ray remains unchanged as it propagates along the ray. The idea consists in building a huge image database with pictures of a given volume from a dense grid of possible viewpoints around this volume. To generate new views of the volume for each pixel of the new view, we compute the direction of the ray going through this pixel and we look for the pixel in the image database that has the same direction. It is difficult and often impossible to find a pixel with exactly the same direction, so the nearest ones are taken and interpolated. Of course, when we say pictures of a volume we mean a volume containing an object to model and render. These techniques are limited to small volumes, because of the volume of data needed. One advantage of these techniques is the fact that they don’t need geometry to represent objects.

In practice, these techniques use two planes to parameterize the space of rays. The near plane, representing the position of the array of cameras, and the far plane representing the focal plane of the cameras. To represent an entire volume six pairs of planes are used building a box around the object.

An alternative parametrization of a Light Field was presented in the work of Wood et al. [104]. As in view-dependent texture mapping, the surface light field assumes that a 3D mesh of an object exists. However, the surface light fields uses a dense set of cameras resulting in a more precise representation of the Plenoptic Function. In this work, at each point on the surface of the geometry there is an associated lumisphere, a 4D representation of the radiance leaving the surface. By taking advantage of the fact that lumispheres vary smoothly on the surface, this representation can be compressed with higher fidelity than the standard Light Field.

Schirmacher et. al. [84] introduced a generalization of the lumigraph using real-time depth maps that can reconstruct arbitrary views from multiple source images on the fly. The basis of their work is the two-plane parameterized Lumigraph with per-pixel depth information. The depth information is computed on the fly using a depth-from-stereo algorithm in software. With a dense depth map, they can model both concave and convex objects. Their current system is primarily limited by the quality and the speed of the stereo algorithm (1-2 frames/second).

Buehler et al. [11] presented a new approach to IBMR that tries to generalize many of the previous methods. In this method a series of cameras are used in a unstructured configuration to generate a Lumigraph style rendering. As in previous methods, a 3D model of the objects is used to choose correct cameras. A more precise metric is also presented to compute which images need to be used for each new viewpoint. This metric takes into account factors such as the epipole consistency, a ray passing through the center of projection of a camera that can be trivially reconstructed from the database, the minimal angular deviation, a more natural and consistent measure of closeness than the used before, the continuity and the resolution sensitivity.

A new family of image-based representations and the corresponding image-based rendering algorithm is presented by Lischinski et al. [51], that introduced the layered depth cube (LDC) to represent objects. A layered depth cube consists of three high-resolution parallel LDIs corresponding to three orthogonal directions. With each sample of the LDC, its depth value, surface normal, the diffuse shading, etc. are stored. At run-time, three LDIs are warped and the z-buffer is used to resolve the visibility. The holes are filled by ray-tracing.

As described in the introduction, this work has been inspired by in a real world application. We worked in a really complex problem and we had to resolve "real-world" problems. The

difficulties found during this development inspired most of our research.

One of these problems was the difficulty to texture a relatively complex geometry using images. One issue was the waste of memory using standard image-based modeling techniques that unproject images into the geometry and the lack of control of missing textures when using projective textures. We encountered this problem during the development of the virtual environment used for the CREATE project. We studied existing solutions to make the modeling process easier, and we developed a new approach inspired on the work of Debevec [17] that partially solves some of the problems of this technique; The lack of user control for missing textures and the need of subdivision of the geometry.

The other problem was the capture and rendering of real trees. In the Garibaldi square we worked on, a set of large oak trees exist, which had to be kept in the captured virtual environment. The project needed these trees, because of their historical meaning. None of the techniques reviewed allowed the accurate capture of a real tree. The observation of trees and our knowledge in medical imaging inspired our research in this way, trying to adapt techniques used in tomography to reconstruct the human body, to reconstruct a tree as a volume. The final tree reconstruction is a volume and is rendered as a cloud of billboards in the same way as other techniques used in image-based rendering where no geometry is needed.

# Part I

# Capture and Render Real World Scenes

# Chapter 3

# Layered View-Dependent Textures

Recent developments in modelling from images and other real-world capture techniques like, laser or time-of-flight scanning, or stereo-based methods, have resulted in an increasing interest in the creation and display of 3D models in realistic computer graphics and virtual environments (VE's). The visual quality of such environments can be very high, thanks to the richness of the textures extracted from high-resolution digital photography and the quality of the reconstructed 3D geometry.

The applications for such VE's are numerous. Urban planning and environmental impact studies, archaeology and education, training, design but also film production or computer games are just a few cases where these highly-realistic virtual environments comprise an important quality and technological advancement.

Modeling-from-images approaches have been largely based on pioneering work in Computer Vision [97, 30]. In Computer Graphics, the work by Debevec [18], and his Facade system have inspired much of the work which followed. Image-based modelling and rendering methods have also been developed [59, 48, 34], which are yet another alternative method to capture and display real world scenes. The Facade approach, and the follow-up interactive rendering version [17], use a view-dependent rendering algorithm. The textures used in display are blended from textures from multiple views, and can be quite satisfactory. In the interactive case, projective textures are used, requiring a geometric visibility pre-process (subdivision of polygons).

In practice, several commercial modelling-from-images products have been since developed and are used for real world projects (e.g., [78, 4, 95]). All of these products however have adopted standard, rather than projective, texture mapping for display: each polygon has an associated texture, which is extracted from the input photographs by applying inverse camera projection.

Geometry is thus displayed as regular textured polygons, and can be directly integrated into a traditional computer graphics or virtual environment rendering system. This choice is also justified by a need for control of visual quality. Artists can intervene at various stages of the process, using standard image-editing tools (such as Adobe Photoshop$^{TM}$ [3], or GIMP [98]), for example to fill in missing texture information or remove undesirable objects etc. They can also edit the textured geometry using standard modelling tools [5, 21], in a traditional graphics/VE production workflow.

In this chapter we present a new algorithm and a new workflow which address a number of shortcomings of previous methods.

- We adopt a view-dependent display model, by blending textures coming from different input photographs corresponding to different viewpoints; the quality of the renderings is thus much higher than the result of standard modelling-from-images products. One goal of our approach is to remain compatible with standard graphics and VE rendering systems, typically in the context of a scene-graph based approach.

- We develop a new projective texture rendering approach, which does not require subdivision of polygons. This approach is based on image-layers, which are automatically generated by our system. This also reduces texture memory requirements compared to texture extraction approaches.

- The image-layers are tightly integrated with standard image-editing programs. Artists can thus edit the image in a standard manner, maintaining tight control over final image quality.

We have implemented our system, and we show results from a reconstructed 3D model of a real-world project, which is the construction of the Nice Tramway. For comparison, we show how existing commercial products can be used to create view-dependent renderings, and we show by example that our method has lower memory consumption, superior visual quality and is easier to use and faster for model creation.

## 3.1 Related Previous Work

As mentioned in Chapter 2, we first review the most closely related work, which is that of Debevec and colleagues, on view-dependent texture mapping. We also briefly discuss layering, visibility ordering and image editing.

### 3.1.1 View Dependent Texture Mapping

In their original paper [18], Debevec et al. presented the first modelling-from-images system Facade. The idea of view-dependent texture mapping (VDTM) was presented, but the rendering approach is clearly off-line and uses model-based stereo to obtain very impressive visual results. This work was transposed to an interactive rendering context [17], in which the polygons which are partially visible in one of the input images are subdivided and missing textures are filled by simple color interpolation. Projective texturing is used and textures are blended by creating a view-map of closest viewing angles. This method has the advantage of being completely automatic; the flip side of this is the lack of quality control in a traditional content-creation workflow. In particular, for surfaces which do not receive a projective texture a very simple hole-filling technique is used. This step, which is one of the main sources of visual error, is done by simple interpolation of colours from neighouring vertices. This makes the use of this algorithm difficult with only one input image, for example. Subdivision of input geometry could also be problematic in some contexts, due to numerical imprecision. The method we will present can be seen as an extension of this approach, by addressing these two issues.

### 3.1.2 Visibility Ordering and Image-Editing

Our algorithm for layer construction is based on a hardware visibility ordering algorithm by Krishnan et al. [41] where, giving a viewpoint, objects in a scene are sorted in groups, each group corresponding to a level of visibility. Objects in the first group are completely visible from the viewpoint, objects on the second group are partially or completely hidden by objects of the first group and so on. This algorithm has a problem with concave objects because they can hide parts of themselves, with cycles between a series of objects and with intersections because a cycle is then created between the two intersected objects. A more complete, software solution has been presented by Snyder and Lengyel [91]. Visibility cycles are resolved in the algorithm by Newell et al. [63]. We choose the approach of Krishnan because is the only hardware solution available, and we modify it to solve some of the aforementioned problems.

Our work is also related to the work of Oh et al. [66], in what concerns the integration of 3D information and image editing. Most of the algorithms developed by Oh et al. could be applied to our image editing operations, making them more efficient and easy to use.

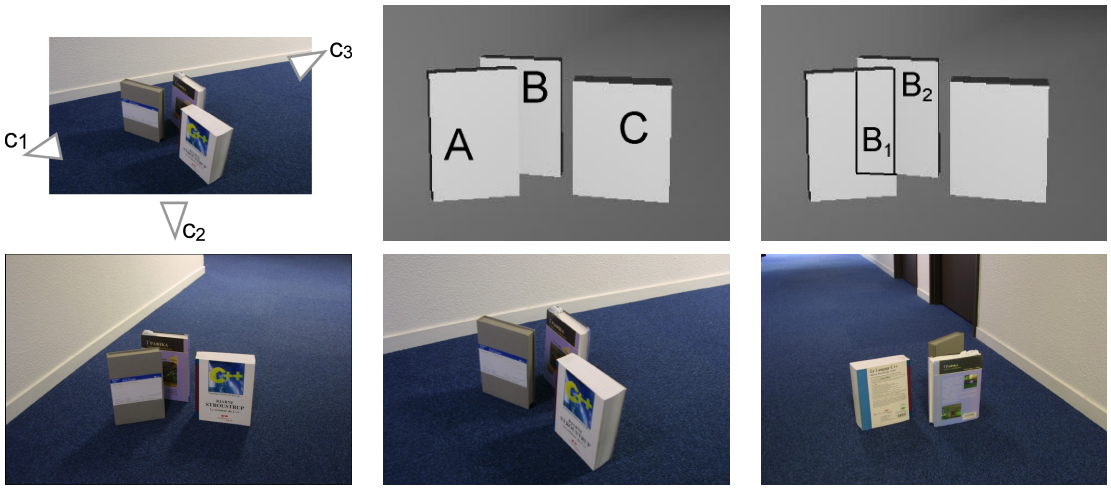



Figure 3.1: Row 1 left: The geometric configuration of the scenes and the cameras. Row 1 right: the corresponding 3D geometry. Row 2: The three images corresponding to the three cameras, $c_1$, $c_2$ and $c_3$.

## 3.2 Overview

We will first discuss 3D model reconstruction using modelling-from-images solutions, and how they can be adapted to view-dependent rendering. We describe the problems which are inherent in these approaches, and how they can be circumvented, albeit at a high cost in preprocessing time by an artist and texture memory.

We will then describe our image layer construction algorithm. Our goal is to achieve projective texture mapping, without the need to subdivide input geometry. Consider the example shown in Fig. 3.1. The scene has three objects, $A$, $B$ and $C$ and we consider 3 views, cameras $c_1, c_2$ and $c_3$. The corresponding images are shown from each camera in the second row. When using the previous projective texture mapping algorithm [17], even for such a simple scene, objects $A$ and $B$ must be subdivided. For view $c_1$, $B$ will be cut into the $B_1$, $B_2$ (see Fig. 3.1 top row). When moving from $c_1$ to $c_2$, object $B_2$ will have the image from $c_1$ as a projective texture, and then will blend the images from $c_1$ and $c_2$. Object $B_1$ however, will always be displayed with the image of $c_2$, since it is invisible from $c_1$.

Instead of doing this, we create layered images, or *image layers*, by first constructing *visibility layers*. For a given camera or view, a *visibility layer* is a set of surfaces for which no pair is mutually occluded. In the example of Fig. 3.1, and for camera $c_1$, there are three layers, the first containing $A$ and $C$, the second containing $B$, which is partially hidden by $A$ and the third the

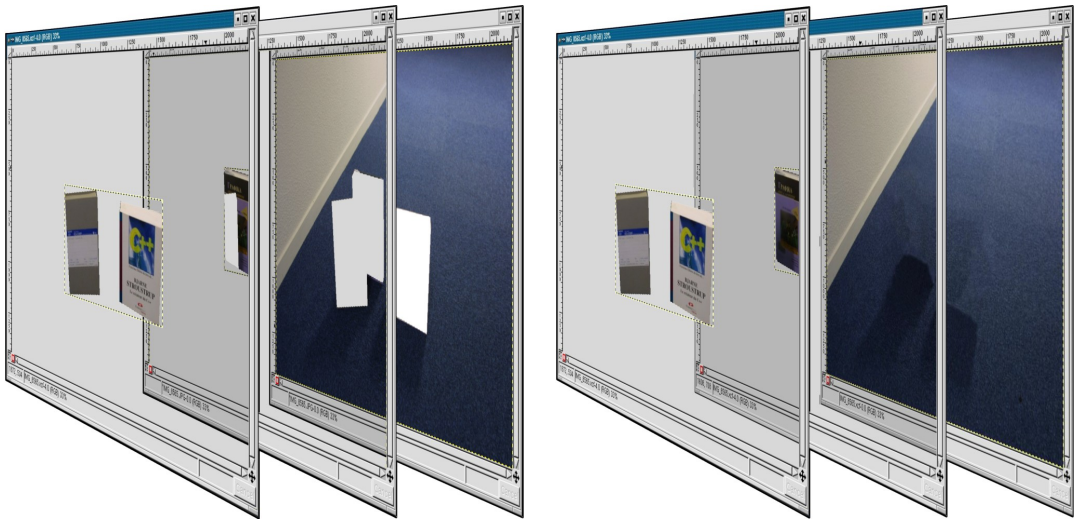


Figure 3.2: The three image-layers viewed in GIMP, corresponding to camera $c_1$, before and after image editing (clone brushing etc.).

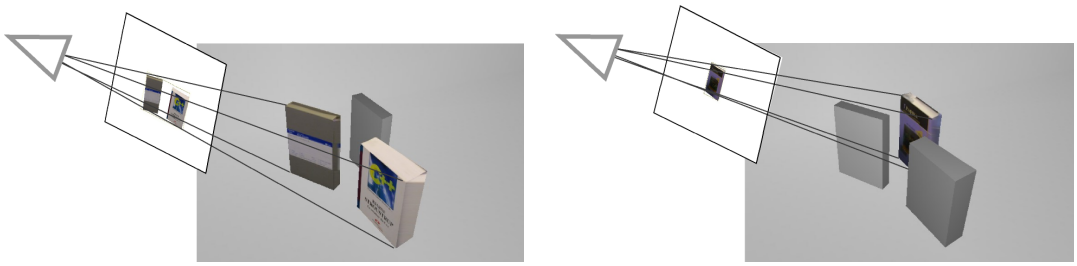


Figure 3.3: Visualisation of the projected layers.

background We then create an *image layer*, corresponding to each *visibility layer* (see Fig. 3.2). The first layer corresponds to the portions of the image from $c_1$ covered by the pixels corresponding to $A$ and $C$, and the second image layer corresponds to object $B$.

We construct geometric visibility layers by adapting the hardware-based algorithm of [41], and then create standard image-editing (e.g., Adobe Photoshop$^{TM}$ [3] or GIMP [98]) layers, which will be used as projective textures. We discuss how these layers can be edited using standard techniques. For example, for the layers of $c_1$, the missing texture of $B$ is filled in GIMP.

The edited layers are then used as projective textures in our new algorithm. The algorithm uses the corresponding compact image layer as projective textures for the appropriate polygons of the reconstructed 3D model, avoiding the need for geometry subdivision. This is shown in Fig. 3.3.

Because the layers and image-editing have resolved visibility we can move around freely.

We have implemented this entire workflow, using REALVIZ ImageModeler$^{TM}$ [78] to reconstruct 3D geometry, and GIMP [98] as an image-editing tool. We present the method and results on a model of Place Massena in Nice, showing how we can rapidly construct appropriate texture layers, and achieve interactive, high-quality, view-dependent display of captured real scenes.

## 3.3 Real Scene Capture

As mentioned previously, to achieve practical solutions for immersive display of captured real scenes, commercial solutions have adopted traditional, non-projective texture mapping. Most commercial products aim at Web-3D or lower image quality applications, in which a single "view-independent" texture per object is acceptable. We briefly describe the workflow used in such solutions, how it can be adapted to multiple viewpoints, as well as the problems encountered in such an approach.

The first step of our approach is to create the coarse 3D model; we have chosen to use an approach based on modelling from images. We first calibrate the cameras of the input photographs, and we construct an approximate, polygonal 3D model of the real scene. In this work we use REALVIZ ImageModeler$^{TM}$ [78]. Other products such as PhotoModeler [95] or Canoma [4], or research systems such as Rekon [73] could be used with the same result. The research system Facade [18, 17], can also be used to the same effect, but has a different, view-dependent approach to rendering, as described in the Section 3.1.

These systems [78, 4, 95, 73] extract textures (see Fig. 3.4), either from a single image or from a combination of images. Texture extraction results in an "unfolding" of the corresponding part of the input photograph into texture space, by applying the inverse perspective transform. The texture can then be applied via standard texture mapping onto the 3D model generated.

As a result, the overall texture requirements using this method are much higher than the resolution of the input images. Consider for example Fig. 3.4, where the entire input photograph has resolution 2160 x 1440 and just the facade texture has resolution 2323 x 439.

Textures extracted by this process are not always complete. Consider for example the last row of 3.5; on the left we see the result of extracted textures before editing, and on the right we see the edited result. The final result is obtained using standard image-editing programs such as Adobe Photoshop$^{TM}$ [3] or GIMP [98]. An artist will typically use clone-brushing or simple editing to repair these problems. Extracted textured are "unfolded" using the inverse perspective transfrom

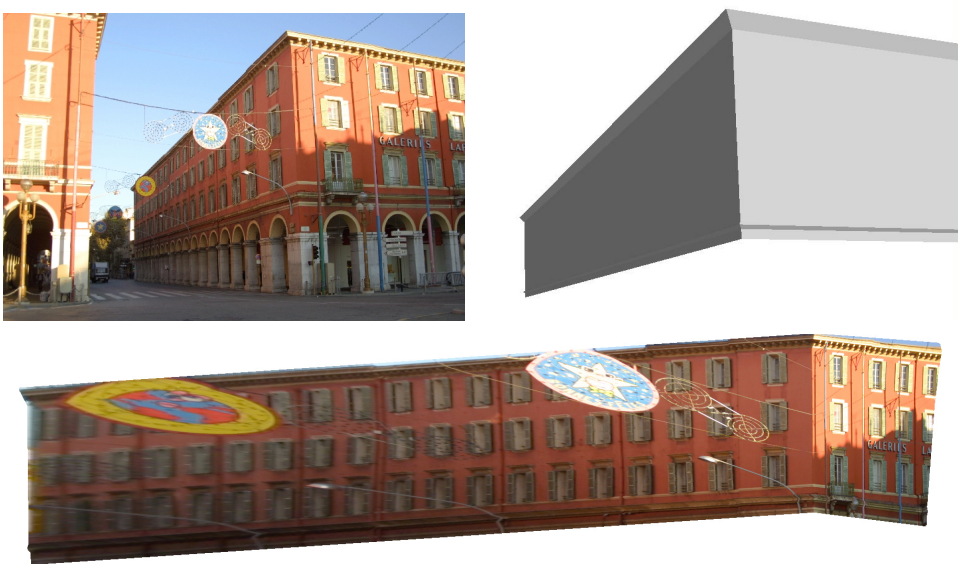


Figure 3.4: Top: the photo of a facade and the corresponding 3D model. Bottom: the extracted texture. Notice how the windows at the extremities have been distorted by perspective unfolding.

and editing is typically performed in this space. Although unfolded texture intuitively correspond to object space, and can be convenient for some editing tasks, artists are used to clone-brushing directly in image space, and can find directly editing the image more convenient. Unfolding textures is fundamentally a inverse projection and resampling problem, and thus is very sensitive to the algorithms used. The unfolded texture can have insufficient resolution in certain regions, and its shape can be unintuitive if a texture is assigned to compound objects (see Fig. 3.4).

View-dependent rendering can also be effected, by displaying the reconstructed model using multiple textures, one per view. We have implemented this and used it in a virtual/augmented reality system [52]; however the construction and editing of the scenes is very cumbersome and the requirements in texture memory extremely high.

### 3.3.1 Handling multiple views

All of the modelling-from-images systems mentioned above, with the notable exception of Facade [18], generate a single textured model of the scene. As a result, the textures extracted are accurate for a single view only. To enable a more visually accurate solution for multiple views, and ideally for a free walkthrough in the virtual environment, a straightforward solution is to use multiple textures and blending. To achieve this, we generate a unique 3D model and extract textures

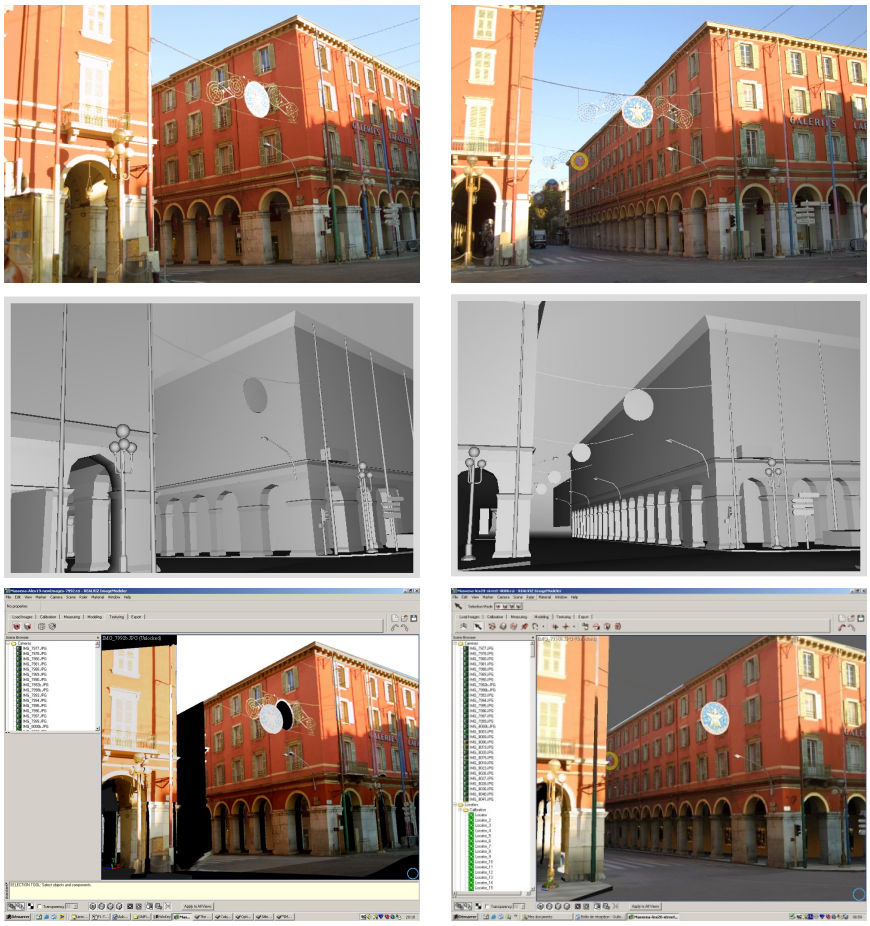


Figure 3.5: First row, 2 of the 3 input photographs used in this example. Second row, the 3D model extracted, viewed from the same viewpoints (snapshots from ImageModeler$^{TM}$. Third row, one intermediate views before texture editing and one after texture editing.

for each viewpoint (input photo camera position) considered. When rendering, for each polygon we blend between appropriate textures, based on a angular distance criteria, which is that of [17].

See Fig. 3.5, in which two input photographs are considered (top row). In the second row we show the two corresponding views of the reconstructed 3D model, and in the third row, the result of the model and extracted textures for the original views. When we move to an intermediate view, we blend between the textures, resulting for example in the images shown in the fourth row.

To be able to move between views, we clearly need to edit the textures and add texture information to the parts of the objects which are hidden in the views corresponding to the camera positions of the original photographs. We describe how this is done in the Section 3.3.2.

Overall, this approach is quite cumbersome, given that extracting and editing textures is a costly and relatively manual process. Another problem with this approach is texture memory usage, since the memory requirement problems mentioned above are multiplied by the number of views being considered.

### 3.3.2 Texture Editing

To effect texture hole filling for objects or parts of objects, multiple techniques are possible. Many systems use automatic techniques, such as extracting textures from different images, selecting the image in which the object is visible (e.g., [73, 78]). However, this poses numerous problems, since the photos have to be colour-calibrated, and perspective distortions are evident due to lack of 3D information and differences in resolution.

In the system we use, ImageModeler$^{TM}$ [78], an additional feature is present, which allows the use of Adobe Photoshop$^{TM}$ [3], to edit the texture. Editing is performed in texture space; as a result the texture is deformed, as shown for example in Fig. 3.4. This approach gives high-quality visual results (see Fig. 3.5, last row). However, it has several inherent problems. First, artists are not used to working in inverse-perspective space, and can thus be confusing for them to clone brush or edit images in this manner. Second, the resolution of the regions which suffer severe perspective distortion (for example the rightmost part of the building facade in Fig. 3.4), is very different from that of the input image. Clone-brushing, which is the most efficient editing technique, is thus very hard.

Nonetheless this solution is workable and gives acceptable results. We have implemented and used it in a virtual/augmented reality system [52]. In the results (Section 6.6) we show examples of its use. However, construction of the 3D data and textures is cumbersome and the resulting model is heavy in texture memory. In addition, if the modeller changes the geometry, any work done on editing the textures is lost, since the original texture needs to be re-extracted, and subsequently edited. Our new layered projective texture solution resolves all of these problems.

## 3.4 Creating Visibility and Image Layers

We now describe our new image layering approach. The first step is to create a set of images with calibrated cameras, and a corresponding coarse 3D model of the scene (Fig. 3.5). Our goal is to create a set of layers for each input image of the set, which will be subsequently used as

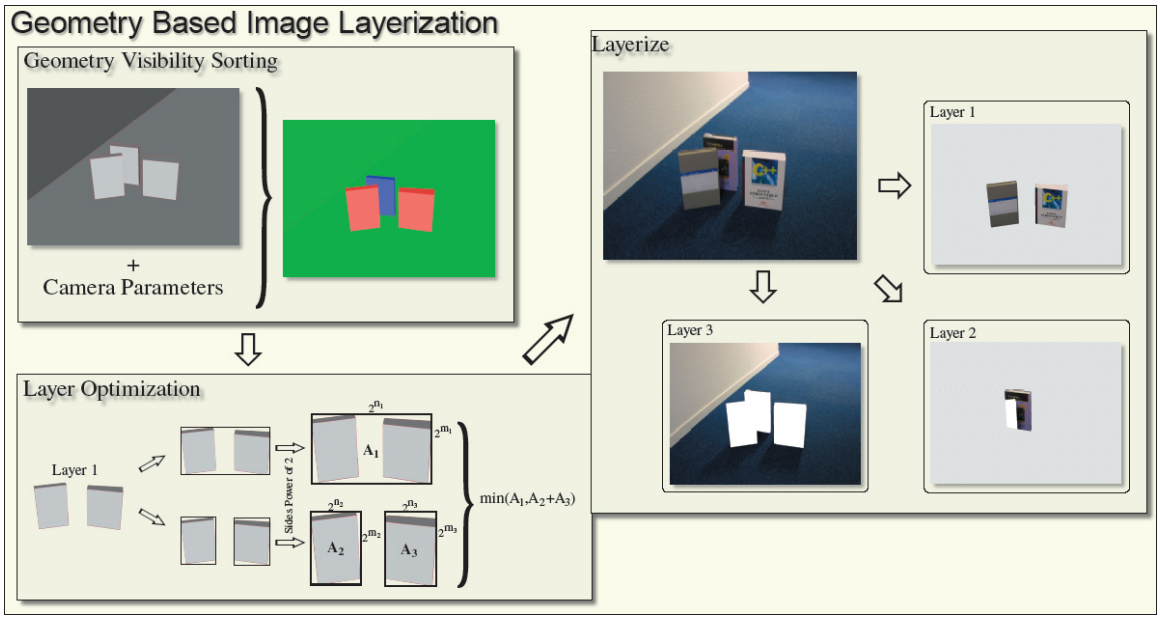



Figure 3.6: Schematic explanation of the layer creation and the optimization process.

projective textures for display. These layers have the following property: for any object in the first layer, no other object in the scene occludes it; for the second layer, once the objects of the first layer have been removed, the same property applies, and so on for each layer. Once these layers have been created, we can use projective texture mapping without the need to subdivide geometry, since we create a separate projective texture map for each layer.

As mentioned previously, we tightly integrate the creation of these layers with a standard image editing program, so that an artist can fill in the missing parts of a given layer using standard techniques such as clone brushing etc.

To create these *image layers*, we first perform a visibility sort in 3D, to create what we call *visibility layers*, which are groups of objects. For concave objects we split them into separate polygons to avoid cycles. Each one of these *visibility layers* is then projected into the image space and subgroups of objects are created in a way that the wasted space is minimum as we can see in Fig. 3.8. Each subgroup is then cut from the original image to create what we call *image layers*, which are true image-editing-programme layers, usable in an image-editing programme such as Adobe Photoshop$^{TM}$ [3] or GIMP [98].

For the first step, we adapt an existing visibility sorting algorithm and then optimize the layers so that we do not waste image resolution. The second step is achieved by sending the output

of the optimization to the editing program which then creates standard image-editing 2D layers, so that an artist can subsequently edit the result. See Fig. 3.6 for an overview of this process.

### 3.4.1 Visibility Layers

To compute the visibility layers we adapt the visibility algorithm of Krishnan et al. [41]. Initially, we render the entire scene into an item buffer, i.e., we use a unique identifier for each object in the scene. We then read this buffer to create an initial working set of potentially visible polygons, which thus contains all objects with at least one visible item buffer pixel containing its id.

The algorithm then iteratively constructs a set of *visibility layers*. For each iteration, the item-buffer of the working set is rendered into the color and the stencil buffer, and the z-test inverted (`GL_LESS`). For each pixel with a stencil value greater than 1, the object corresponding to the item-buffer value of the pixel is partially hidden, and thus the object is removed from the working set. The working set is then rendered iteratively, until the stencil buffer contains values less than or equal to 1.

The remaining objects are those completely visible, so they will be the objects for the current layer. However, an additional test is done to the set of objects to solve a problem in the algorithm of Krishnan [41]. We can see an illustration of this problem in Fig. 3.7. In this case the algorithm chooses objects $A$ and $C$ as objects for the first layer. However, the only completely visible object is object $A$. The additional test consists in a comparison between the first normal rendering pass and an additional normal rendering pass with all the selected objects. All objects that are visible in the final pass and not completely visible in the first pass, cannot be considered as completely visible. Thus, they are removed from the current working set. All the objects selected are then inserted into the current *visibility layer*, and removed from the working set. The current layer is saved, then set to empty, and the iteration continues. If a cycle is encountered the algorithm returns an empty layer. When the first empty layer is found we check if there are objects remaining, if there are we split them into polygons, we assign an identifier to each polygon and we restart the iterations. Nonetheless, cycles can occur, even at the level of individual polygons/triangles, and in this case we would need to identify the cycles and cut the geometry using an algorithm such as that of Newell et al. [63].

The output of this algorithm is a set of *visibility layers*. In each *visibility layer* we have a set objects or polygons, which are used to build layered images.

In our implementation we have optimized this process hierarchically thanks to the struc-

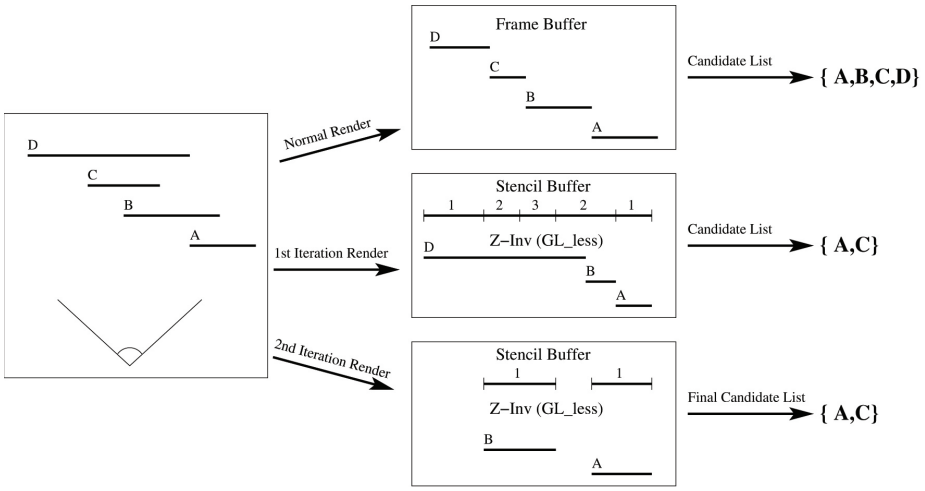



Figure 3.7: Illustration of the problem in the original Krishnan [41] algorithm.

ture of the scene which is provided by the image modelling programme. In particular, the scene provided by ImageModeler$^{TM}$ is organised into objects which contain a set of faces or polygons. Initially, identifiers are given to objects rather than faces, and the algorithm is performed on objects, rapidly eliminating large objects. When we can no longer proceed with objects, we split them into their constituent polygons. In our examples this is enough to resolve the visibility in the scene. Nonetheless, cycles can occur, even at the level of individual polygons/triangles, and in this case we would need to identify the cycles and cut the geometry using an algorithm such as that of Newell et al. [63].

### 3.4.2 Creating Image Layers

To create the *image layers*, we start with the set of *visibility layers*. For each *visibility layer*, we project each constituent polygon into image space, resulting in a 2D contour for each. We then perform a simple clustering algorithm for the set of contours. We take advantage of the fact that due to `OpenGL` restrictions, each texture has to have resolution of a power of 2. For a pair of contours $c_1$ and $c_2$ we merge $c_1$ and $c_2$ into the same layer, if:

$$A_{12} < A_1 + A_2, \tag{3.1}$$

where $A_i$ is the minimal bounding box with resolution in $x$ and $y$ powers of two, with area greater than the area of the bounding box of the projected contour $c_i$. Thus each layer is potentially split

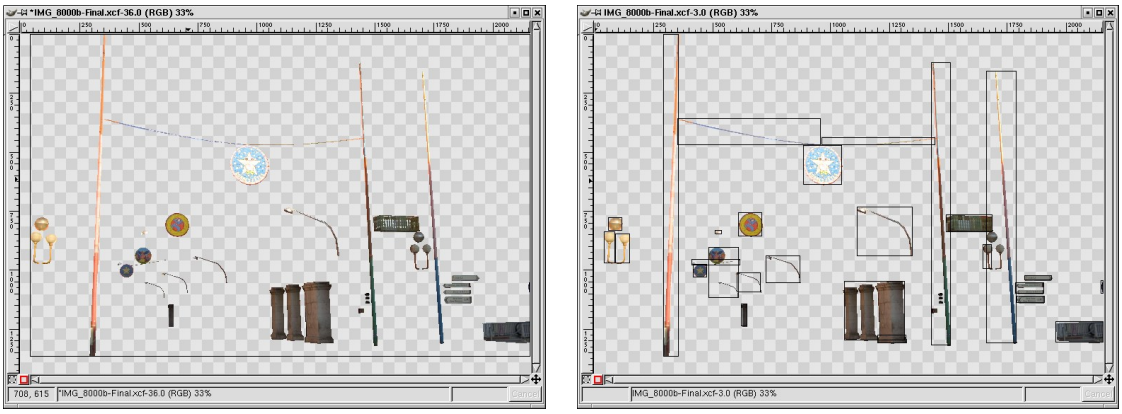


Figure 3.8: Left: A *visibility layer* before optimisation. All objects shown are in the same layer. Right: The resulting layer after optimisation to avoid empty spaces, and reduce overall texture memory.

into several layers, Fig. 3.6. Examples of image layers can be seen in Fig. 3.2 and Fig. 3.8.

This results in an increase of the number of layers, but is required to reduce the texture memory requirements of our projective textures. This is because very distant (in image space) can belong to the same *visibility layer*, which would result in textures with large areas of empty space, wasting texture memory. The final set of layers is then ready for image editing in the appropriate program. These layers consist of a 2D bounding box in image space for the image being treated. This process is shown in Fig. 3.8.

The layers are then sent to the image-editing program (GIMP [98] in our case), which creates a 2D image layer corresponding to the input image. An example of such layers is shown in Fig. 3.8.

## 3.5 Image Editing

Once layers have been created in the editing program, an artist can perform standard operations required to fill in missing details, or to create transparency maps, which are particularly effective for multiple-view rendering.

We return to the example of the facade cited earlier (Fig. 3.4). Using our new approach, the artist now can use all standard image-editing tools, such as clone brushing in the usual manner, directly in image space.

In Fig. 3.9, we see the layers corresponding to the facade and to the posts in front. The

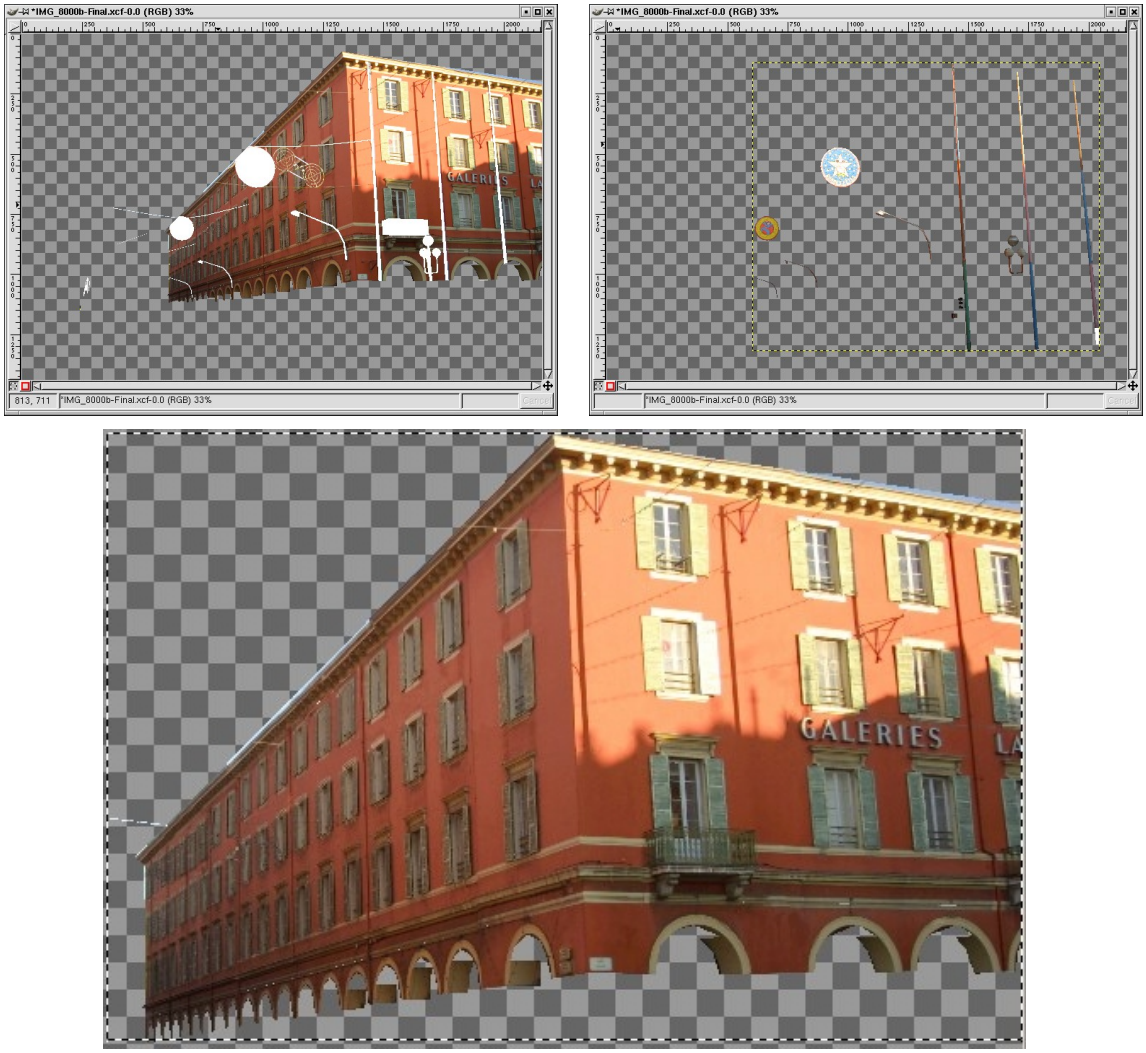



Figure 3.9: Top row: the layers of the facade and the posts in front. Bottom row: the result after clone brushing on the facade.

artist works in the layers of the facade, performs clone-brushing and the work is complete.

This approach is particularly useful for small details. Perspective correction in the style of [66] could be easily added in this case and would render the tool even more useful and efficient.

Another useful kind of edit which is practical in this context is the manual creation of alpha maps. Again, artists are used to working in perspective image space, and thus creating alpha-maps is easier. An example of an alpha map, used to represent the bars of a balcony is shown in Fig. 3.10.

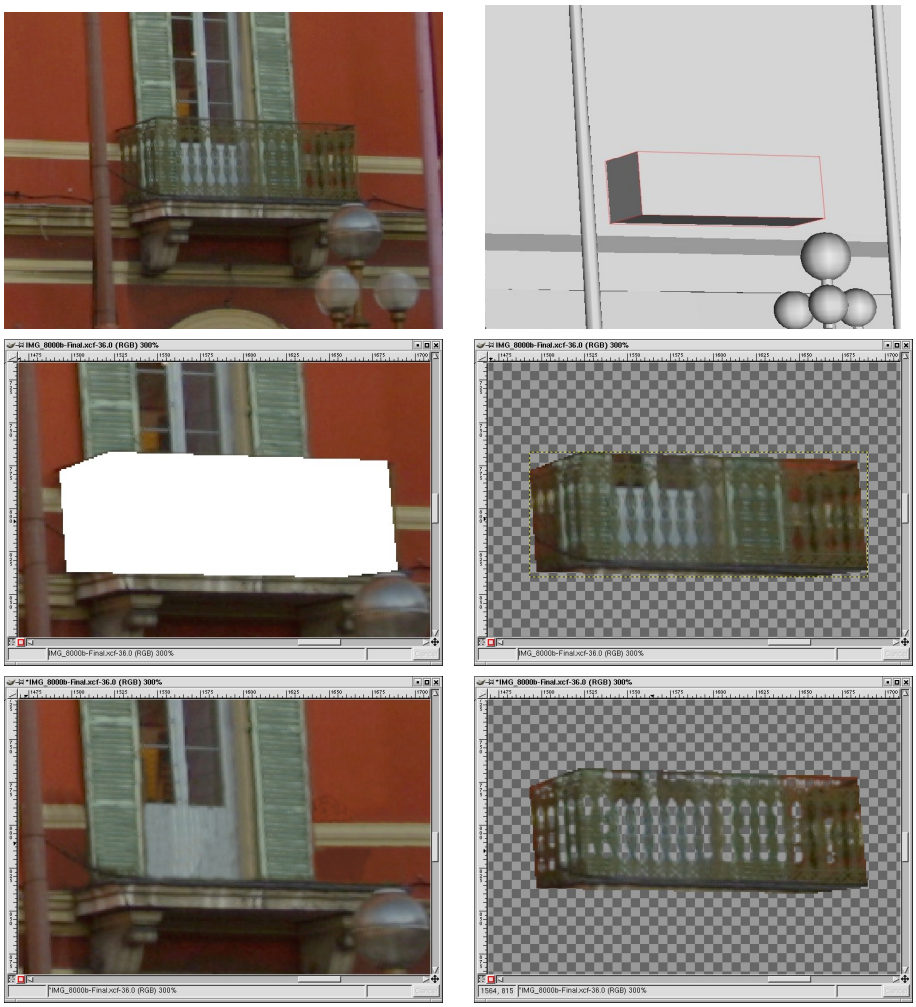


Figure 3.10: First row: zoom of the original image, and reconstructed geometry. Second row: facade and balcony layers. Third row: edited layers which is an alpha map for the balcony.

## 3.6 Layered View Dependent Projective Texture Display

After creating the *visibility layers* and manually editing the *image layers* for the objects in the scene we can now use projective texture mapping to display the captured real scene.

For any viewpoint corresponding to the camera of a photograph, all we need to do is to render the geometry, and for each object assign as a texture the corresponding layer. To allow viewing from other viewpoints, we will blend between textures in the spirit of [17]. We explain this process in detail in what follows.

### 3.6.1 Data Structures for Layered View Dependent Projective Textures

As mentioned previously we have designed our approach to fit into a typical scene-graph based rendering system such as OpenGL Performer$^{TM}$ [86]. We thus describe the data structures in terms of scene graph nodes and traversals.

For standard projective texture mapping, a scene-graph based implementation can have the form shown in Fig. 3.11(a). An image texture node is created, and associated to the geometry nodes (typically polygons). The texture node contains the texture, its width and height and the texture projection matrix (i.e., position of the camera, view direction, up vector, field-of-view near and far).

For layered projective texture mapping, the graph is modified to contain a sub-image texture node, which in addition to the information of a "standard node" described above, contains the image-coordinates of the sub-image $(x_i, y_i, w_i, h_i)$ Fig. 3.11(b).

To render the layers, we need to apply a sheared perspective, transformation, shown in Fig. 3.12. This can be directly implemented using the OpenGL command `glFrustum` with parameters:

$$x'_{min} = x_{min} + \frac{x_i(x_{max} - x_{min})}{w}$$

$$x'_{max} = x_{min} + \frac{(x_i + w_i)(x_{max} - x_{min})}{w}$$

$$y'_{min} = y_{min} + \frac{y_i(y_{may} - y_{min})}{w}$$

$$y'_{may} = y_{min} + \frac{(y_i + w_i)(y_{may} - y_{min})}{w}$$

### 3.6.2 Rendering Multiple Views

To render multiple views, we need to further modify our scene graph structure. We create a view-dependent projective texture node $VDP$ which contains a list of view-dependent texture nodes, associated with each (unique) geometry.

When encountering such a node during graph traversal, the renderer chooses the blending factors for each appropriate texture, sets up the texture matrix for the nodes with non-zero blending factors, and renders the associated geometry in as many passes are required.

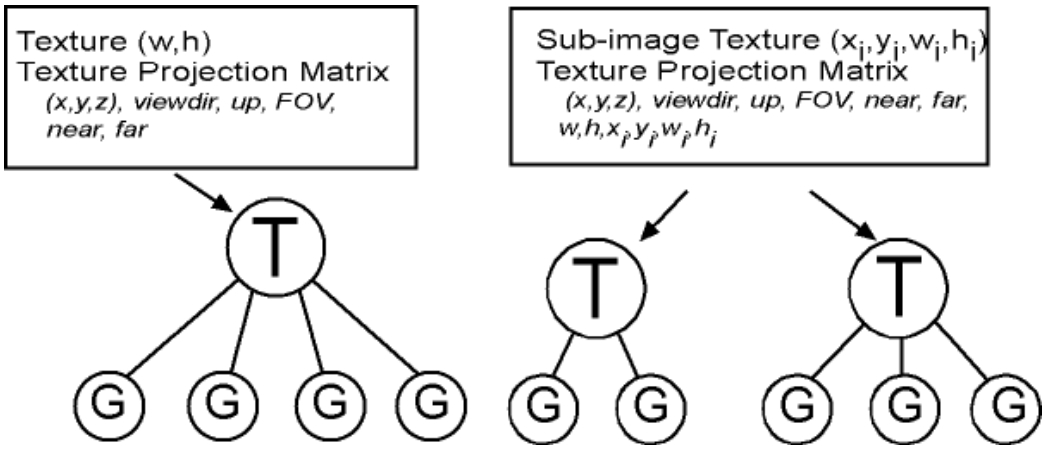



Figure 3.11: Left: scene-graph implementation of standard projective texturing. Right: scene-graph implementation of layered projective texturing.

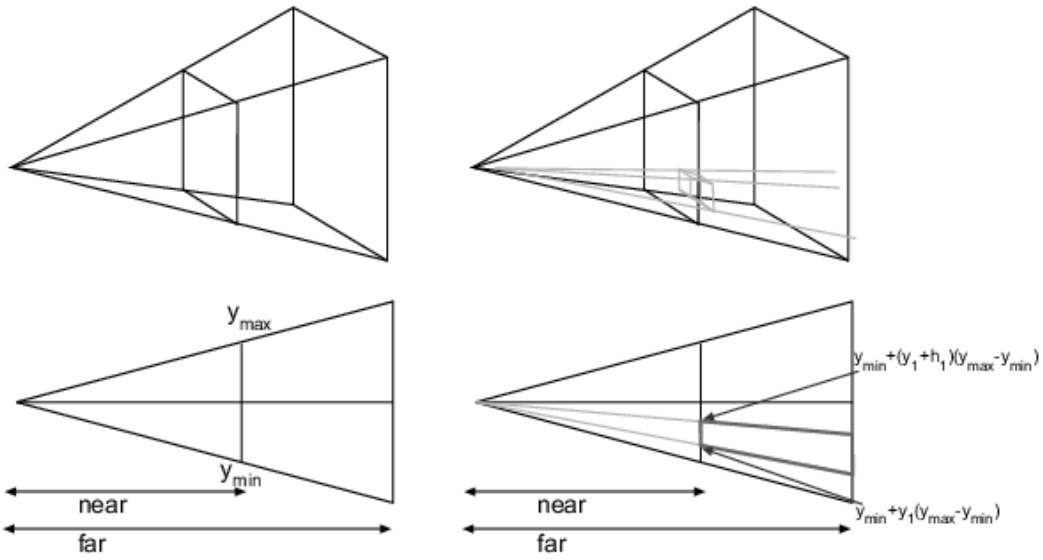



Figure 3.12: Left: Standard perspective frustum, Right: sheared frustum for use with layers.

## 3.7 Results

We show here results of our system; an AVI movie demostrating the system can be found at http://www-sop.inria.fr/reves/publications/data/2003/RD03b/RecheDrettakisPG03.avi. We use the same example as that presented in previous Sections, which is a reconstructed model of Place Massena in Nice. This capture has been performed in the context of a virtual/augmented environ-

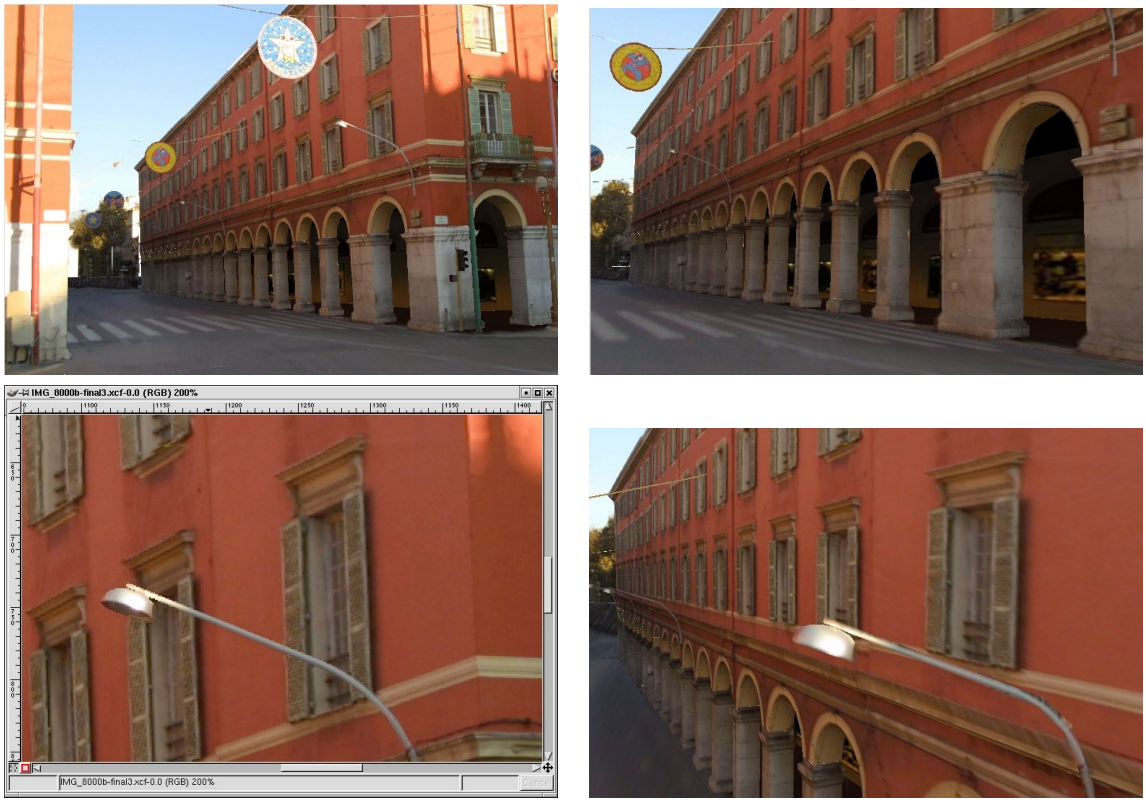



Figure 3.13: Top row: two intermediate (non-input) views, using our new, layered projective texture display. Lower row (right): Closeup of input image; (left) closeup of rendering of our system from a different viewpoint.

ment simulation of the Tramway project in Nice. The Tramway will pass between the building shown here and a station will be placed at this location.

In Fig. 3.13, we show intermediate views (i.e., different from the input photos of resolution 2160x1440) using our view-dependent projective texture method. The algorithm creates a total of between 98 and 160 (optimized) layers for each image, many of which are very small and require no editing. The total texture memory required is between 10-12 Megabytes, depending on the image. The algorithm to create the layers takes less than 3 minutes to complete (most of which is spent in interprocess communication between GIMP and our system). The entire process of editing and cloning the layers took about two days; using a texture extraction-based approach, with the same scene and the same views required three weeks.

As we can see, the quality of the images, and in particular the zoomed-in images is very high using our approach, since textures have not been altered with inverse perspective distortion.

## 3.8 Discussion

We have presented a new workflow for displaying captured real scenes, using projective texture mapping. We create *image layers* of the input photographs, which can be edited in a standard image-editing program to fill in missing texture. The layers are then used as projective textures of the geometry. This is done for a small number of multiple views, and the rendering algorithm blends between the closest views to the current viewpoint.

Compared to the initial interactive projective texture mapping approach [17], we avoid the need to subdivide geometry for visibility, and we introduce more traditional artist intervention to manually fill in missing textures, rather than depending on interpolation. Thus the advantages of our approach are that we avoid increasing the polygon count of the scene, and that the approach fits well with traditional workflows, where artist's control over image quality is paramount; the disadvantage is that hole-filling is no longer automatic.

Another big advantage compared to the initial VDTM approach is the reduction of texture memory used. This is even more important if the scene has a lot of occlusions. To use VDTM approach in this kind of scenes would require a large number of images of the scene, to be able to have information for the occluded geometry. Using VDLTM the information of all these images can be combined into one or two layered images containing all the information, even for occluded objects. See Fig. 3.14 for a graphical explanation of this advantage. A scene like the one shown in this figure can be totally covered using only three images. Of course, if the model we have of the scene is not very precise, or the objects have micro-geometry or materials that depend of the view-point, we will still need a large number of images to capture the view-dependent phenomenon.

Compared to texture-extraction methods, our approach significantly reduces texture memory requirements and results in better texture quality, since inverse perspective results in large textures and quality loss due to resampling.

In addition, for some image-editing tasks, artists are used to working directly in image-space, rather than in inverse perspective space, and thus the image-editing process is faster overall.

High image quality is very important. Informal observation of our system in use has shown that the user is much less sensitive to parallax errors due to the high quality of the projective textures used, and thus a much smaller number of input photographs is required, compared to previous methods.

This is just the first step in building complete and usable solutions to effective, high-quality renderings of captured real scenes for immersive systems.

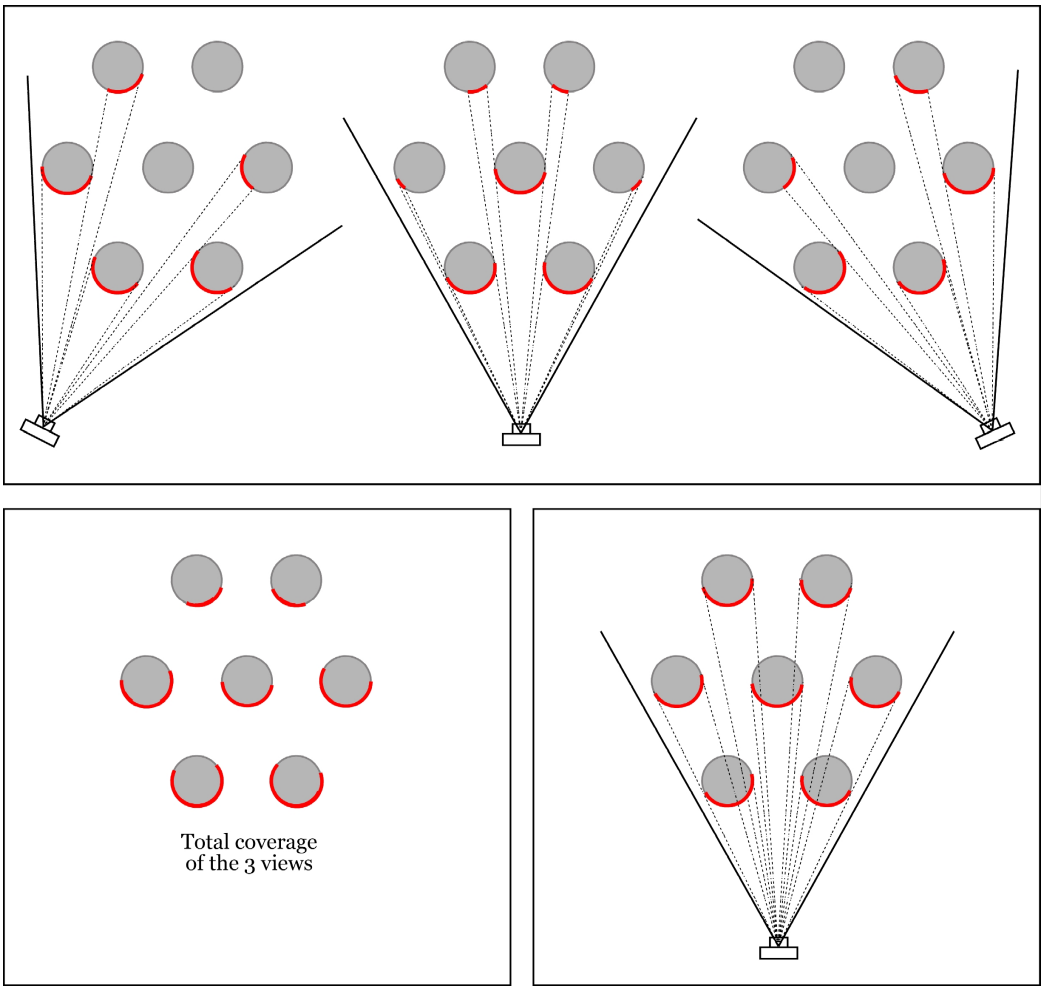



Figure 3.14: Top: 3 views for View-Dependent Texture Mapping. Bottom Left: Coverage of the 3 views shown at the top. Bottom Right: Coverage of a Layered Texture, after warping the three images to a common coordinate system.

On the capture side, there are many preprocessing tasks which require care and manual intervention, such as colour balancing between multiple views, or ensuring that shadows are in the right place due to motion of the sun. Similarly, given that we have multiple views, texture synthesis approaches could be used to avoid manual clone brushing to some extent.

# Chapter 4

# Complex Objects : Trees

Even if the image based modelling and rendering has increased the visual quality of virtual environments (VE's) there are still objects with high level of complexity that are not adapted to be used with the technique explained in the Chapter 3. This complexity can be geometric, trees are a very good example of objects with very complex geometry, or can be gave by the material, glasses or water are good examples.

In the context of the CREATE project we have found difficulties to capture and render such objects that where present in all the created urban scenarios. We have choose to solve the problems for trees that where very important in all scenarios.

Reconstructing and rendering trees is a challenging problem, as said before, due to the geometric complexity involved, and the inherent difficulties of capture. In this chapter we propose a volumetric approach to capture and render trees with relatively sparse foliage. Photographs of such trees typically have single pixels containing the blended projection of numerous leaves/branches and background. We show how we estimate opacity values on a recursive grid, based on alpha-mattes extracted from a small number of calibrated photographs of a tree. This data structure is then used to render billboards attached to the centers of the grid cells. Each billboard is assigned a set of view-dependent textures corresponding to each input view. These textures are generated by approximating coverage masks based on opacity and depth from the camera. Rendering is performed using a view-dependent texturing algorithm. The resulting volumetric tree structure has low polygon count, permitting interactive rendering of realistic 3D trees. We illustrate the implementation of our system on several different real trees, and show that we can insert the resulting model in virtual scenes.

## 4.1 Introduction

Capturing real-world trees for subsequent rendering in virtual environments is an interesting and challenging task. Existing digital capture techniques such as laser range finders, stereo or modelling from images have been extensively developed for solid objects which have a well-defined opaque surface. Once the object is captured, standard computer graphics methods are used for rendering. Trees, and vegetation in general, are a special case. The complex geometric structure of leaves and branches means that traditional scanning methods cannot usually be applied. In particular capturing depth information of such objects is a very difficult task because of the high frequencies implied by the complex, small geometries of the leaves and twigs. Procedural models for trees can be guided by images [89], but result in models with very high polygon counts (hundreds of thousands or even millions of polygons per tree).

Our goals are i) to capture real trees from a small number of photographs, using a relatively simple and fast procedure, without the need for specialized equipment or setup; ii) to create a low polygon count tree model which permits realistic interactive renderings of the captured tree.

To achieve these goals we make a key observation: at a reasonable distance, each pixel in a photograph of a tree is typically the blended projection of a number of leaves/branches and the background. This can be seen as a transparency effect, in which the pixel value corresponds to a mixture of tree/leaf colors and background modulated by the opacity of the "tree volume" along the pyramid defined by the camera and the pixel.

Based on this observation, we adopt a volumetric model for the rendering of trees. We cast the problem in a direct volumetric rendering context [56]; we consider the tree as a volume, with opacity and color values. We estimate opacity based on a small set of calibrated digital photographs of a tree, by optimizing over a recursive grid of opacity values. Our optimization uses alpha-mattes extracted from the photographs as estimates of cumulative opacity at the pixels. Once the recursive grid of opacity values has been estimated, we assign a billboard to each grid cell. For each billboard, a texture is generated for each view, using the original image, the number of billboards projected to the image and the opacity of each cell. These textures capture the high frequency detail and are rendered with a view-dependent texturing algorithm.

The main contributions of our approach are:

- The efficient estimation of the opacities for a tree volume on a recursive grid, using the alpha-mattes computed from a small set of calibrated images around a tree.

- The generation of view-dependent textures for each cell of the recursive grid, which allows high-quality view-dependent rendering of the resulting, low polygon tree model using billboards.

This entire process is efficient and compact, creates satisfactory reconstructions of trees, and is particularly suited to those with relatively sparse foliage, which are not well handled by previous methods.

## 4.2 Related Previous Work

Modelling and rendering of trees has interested computer graphics researchers over the last decades. A large part of previous work concentrated on the generation and rendering of entirely synthetic trees based on procedural methods such as grammars (L-systems) (e.g., [75, 20]), or on rule-based plant growing systems based on codified botanical knowledge such as the AMAP system [16]. Such approaches have been used to generate stunning images of forests and trees in general, with a high degree of realism. However, they do not capture the aspect of existing, real trees.

The multi-layer z-buffer method uses precomputed synthetic images of trees for rendering [58, 57]. Volumetric texture approaches have been used to model and render trees; the complex geometry is represented as an approximation of the reflectance at a distance [64]. An adaptation of this approach to hardware was developed later, using textured slices for interactive rendering [60]. Meyer et al. [61], presented a hierarchical bidirectional texture solution for trees at different levels of detail, i.e., leaves, branches, up to the entire tree. This results in very efficient level-of-detail rendering for trees. Efficient rendering of trees can also be achieved using point-based methods [19]. A more recent approach has been developed by [77], in which a volumetric approach effects an implicit level-of-detail mechanism, for lighting (both sun and sky) and shadowing, using shadow maps.

With the emergence of augmented reality applications, as well as the generation of virtual objects from real sources, the need for capture of existing trees has become more evident. A simple approach using two textures based on photos has been developed [39]; rendering is achieved using horizontal slices through the tree, the emphasis being on shading and shadowing. A different approach to capture and render trees was presented by Shlyakhter et al. [89]. This approach first creates a visual hull-like representation of the shape of the tree and then fits an L-system to

the resulting model. The photographs are reprojected onto the resulting polygons of the L-system, resulting in very realistic representations of real trees, at the price of high polygon counts for the resulting models.

Image-based methods have also been developed which can handle objects with semi-transparent silhouettes. These include opacity hulls [55], surface light fields [104] and microfacet billboards [105]. In all of the above cases, a large part of the object is opaque, defining an overall opaque geometric shape, which is recovered. This shape is required since these approaches use either pure standard "surface textures", or, in the case of microfacet billboards, require the underlying geometry to place and cull the billboards used.

In the approach presented in this chapter, we have been inspired by volume rendering methods. The rendering model we have adopted is based on the original direct volume rendering approaches (e.g., [8, 47, 22, 56]). Although our method is closer to image-based rendering than volume rendering, our approach is related to polygon-rendering solutions for volumes (e.g., [102, 88]).

Our opacity estimation approach is inspired by the approaches used in medical imaging. Our method has similarities to algebraic methods used for computed tomography, such as the SART algorithm (e.g., [6]). The construction of a voxel representation for the tree has similarities with the voxel coloring [85] or space carving [42] approaches. However, these approaches do not currently treat semi-transparent volumes such as those we construct for trees. The Roxel algorithm [9] extends space carving to treat opacity using techniques for opacity estimation [96]. A related approach, using a hierarchy of cells for fast rendering of complex scenes was presented by [12]. Compared to medical imaging or the Roxel approaches, we need to represent high frequency detail with low storage cost, which we achieve using the recursive structure combined with the view-dependent texturing technique.

## 4.3 A Volumetric Rendering Approach for Trees

As mentioned in the introduction, we have chosen a volumetric rendering model to represent and render trees, given the fact that we are dealing with photographs containing many semi-transparent pixels.

Consider the tree shown in Fig. 4.1(a); we create a recursive grid enclosing the tree which is surrounded by the cameras. For a large number of pixels in these photographs, each pixel is semi-transparent, and we estimate an alpha value corresponding to the cumulative opacity $\alpha^p$ for

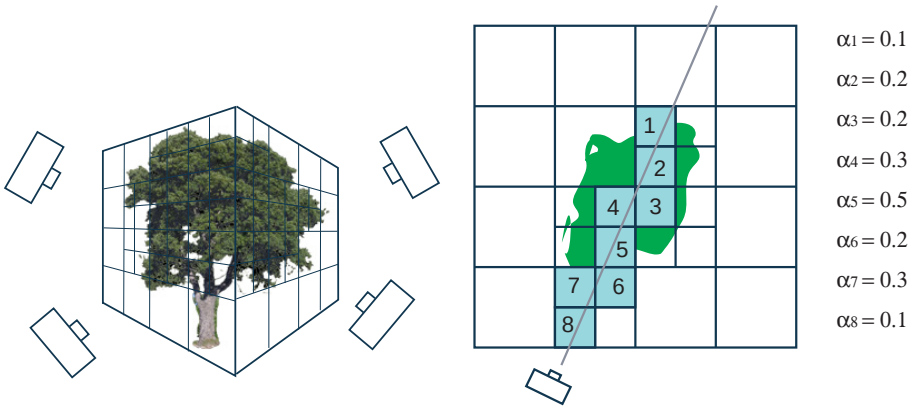



Figure 4.1: (a) A recursive grid is placed around the tree. (b) A ray emanating at pixel $p$ traverses the grid; a set of cells are collected.

each pixel (see Section 4.4). In typical volume rendering applications (e.g., [47, 22]), a sampled opacity field is given at vertices of a grid, typically coming from a scanning device (X-rays etc.). In our case, we will need to estimate the opacity in the grid, based on the alpha values extracted from the photographs. We assign the value $\alpha_i$ to each cell $i$ of the grid. We show how to do this in Section 4.5.2.

Given the opacity and color, we can adopt a standard volume rendering equation to generate images of our trees. Consider a ray emanating from a pixel $p$, which traverses $n$ grid cells which have opacity values $\alpha_i$ (see Fig. 4.1). For now, we assume that for a given pixel we have assigned a color $c_i$ for each cell. The intensity $I$ at a pixel is defined using standard volume rendering equations (e.g., [47]):

$$I = \sum_{i=0}^{n} \alpha_i c_i \Pi_{m=i+1}^{n} (1-\alpha_m) \tag{4.1}$$

which expands to the standard compositing equation for rendering:

$$I = \alpha_n c_n + (1-\alpha_n)(\alpha_{n-1} c_{n-1} + (1-\alpha_{n-1})(..(1-\alpha_1)\alpha_0 c_0)..) \tag{4.2}$$

To render the tree volume, we will develop a rendering algorithm which approximates Eq. 4.2. We first need to estimate the $\alpha_i$ values, and we then need to approximate the color.

In reality, both the opacity and color values are anisotropic and inhomogeneous; estimating and storing them accurately would clearly be intractable, except in the simplest cases. We choose the following simplifications: we will estimate inhomogeneous but *isotropic* discrete opacity values defined on a recursive grid. For color, we will not perform an estimation, but we will partially

Figure 4.2: Three photographs of typical trees we captured using our method. In each photograph the background color is significantly different from the colors of the leaves, trunk and branches.

preserve the anisotropic quality of the color by using a view-dependent texturing approach, with textures generated based on the opacities and the original photographs of the tree (see Section 4.6). We use billboards centered at the cells for the actual rendering.

Our capture method has three main steps. The first step is image capture, calibration and alpha matting. In the second step, the alpha values and the camera calibration are used to estimate the opacity values in a grid. In the third step, we used estimated opacities to generate billboards for each cell and each image. Rendering is achieved with view-dependent texturing on the billboards to efficiently display the trees. These three steps are described in detail in the following sections.

## 4.4 Image Capture, Camera Calibration and Alpha Matting

The first step in our algorithm requires photographing the tree and generating the alpha maps. The tree chosen for reconstruction currently needs to be in a suitable position so that it can be photographed with a relatively distinct background (see for example Fig. 4.2, in which we can photograph the tree in a full circle with a sky background). Trees in urban environments often have this property, at least for the majority of views.

Photographs are then taken while moving around the tree. Our experiments have shown that approximately 20-30 photographs are required. We then need to calibrate the cameras of these photos, see Fig. 4.3. We accomplish this step using REALVIZ ImageModeler$^{TM}$ [78], but any stan-

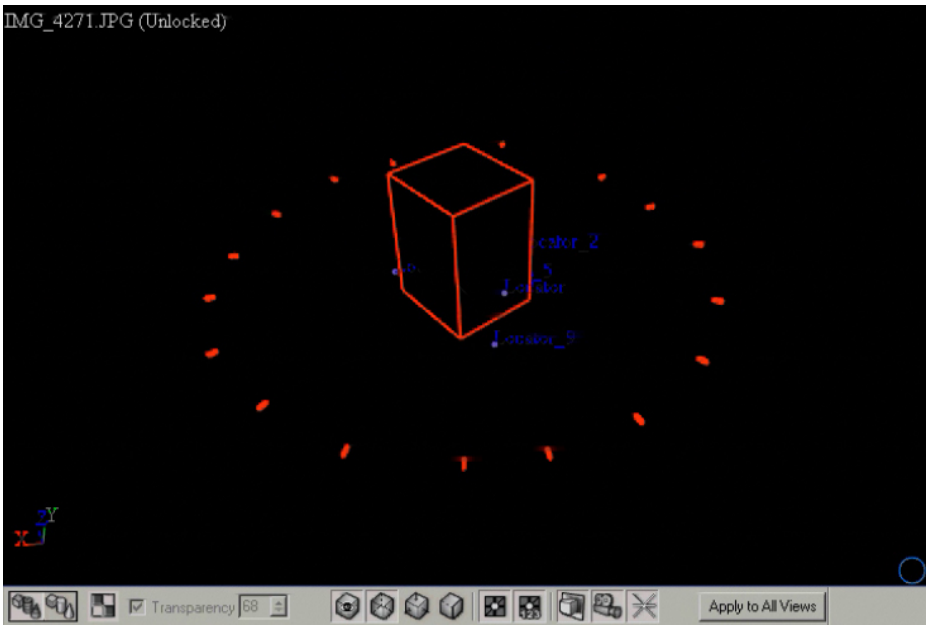



Figure 4.3: Screenshot of the camera calibration, each red dot is a camera and the bounding box represent the minimal bounding box containing the visual hull of the tree.

dard algorithm can be applied for this step [29]. The calibrated cameras will be used in subsequent steps. By adding a few markers in the scene (we use sticks with colored play-doh on the top), calibration takes less than 10 minutes for about 20 images.

Next, we extract alpha masks for each photograph, using the algorithm of Ruzon and Tomasi [83]. This algorithm requires user intervention, in which zones of background and foreground are specified. Consider for example the tree shown in Fig. 4.4. The user specifies foreground and background, and we can see that the alpha-mask algorithm succeeds in finding semi-transparent regions in the image. The area of the pixel contains a mix of leaves/branches and background. The alpha-mask calculates this visibility coverage using colors. Alpha-masks of 0 correspond to background regions and alpha-masks of 1 to completely opaque regions.

The intuition behind the alpha-masking approach is simple. The pixels in the image are transformed into LAB color space and clustered by color distribution. For each pixel, its color can be seen as a best fit interpolation between the distributions of the clusters for the foreground and background colors. The value of this interpolant is the alpha value of the pixel.

We thus consider that for pixels with an alpha mask strictly between 0 and 1, the color of the pixel $C_p$ is a linear interpolation in LAB color space of the background color $C_b$ and the

Figure 4.4: Left: the original image. Middle: the regions specified roughly by the user as foreground and background. Right: the computed alpha-mask (note that we used black for opaque for clarity of the figure).

foreground color $C_f$:

$$C^p = \alpha^p C_f + (1-\alpha^p) C_b \tag{4.3}$$

$C_f$ and $C_b$ are computed as weighted sums of background and foreground, and perturbed accordingly. We create an alpha-image containing the values $\alpha^p$ for each pixel and an image of "pure foreground" $C_f$ values used later for rendering (see Section 4.6).

## 4.5 Reconstruction

We introduce here the process of reconstruction of a tree using as only input the taken pictures, the intrinsic and extrinsic parameters of the cameras and the alpha maps extracted from the original images.

### 4.5.1 Hierarchical Grid Creation

Once we have the intrinsic and extrinsic parameters of the cameras and alpha information, we compute the minimal bounding box that contains the visual hull buildt using the silhouette of the tree at each camera. We consider all pixels with $\alpha^p > 0$ as opaque to compute the silhouette. We ensure in this way that the complete tree is inside this bounding box.

We begin then the hierarchical subdivision of this grid using the following algorithm:

```
while not sufficiently refined
    foreach cell i
        foreach image
            project the cell in the image using the camera information
            check if the projected region contains alpha information
            if no alpha information found then
                empty cell, don't refine and continue with next cell
            endif
        endfor
    endfor
endwhile
```

In other words, if the projection of a cell of the grid in one image doesn't contains alpha information means that this cell is empty, even if the projection of it in another image contains information. The evolution of the refinement of the grid can be seen in Fig. 4.5

### 4.5.2 Opacity Estimation

To achieve volumetric rendering of trees, we need to estimate opacities in each grid cell. We clearly do not have enough information *inside* the volume to be able to estimate both color and opacity, since we only have photographs without any depth information. Even if the depth information were available, it could not be precise enough to be useful. For this reason, we will restrict estimation only to opacity, treating color separately during the texture generation process discussed in Section 4.5.3.

To estimate opacity, we develop a best-fit minimization over all images. This minimization is performed iterating through all pixels of all images and storing the required opacity modification for each cell, that are applied at the end of each iteration.

Before the first iteration, we initialize the opacity values $\alpha_i$ in each cell, by projecting each cell into each alpha-image, and using the minimum of the average opacity values in the respective images as the value of $\alpha_i$. Given that the images are the only data we have available, this minimum is the closest we can get to the final solution at this stage.

A complete iteration of the reconstruction process will loop through each pixel of each image. For each pixel, we cast the pyramid defined by the camera and the pixel square into the grid.

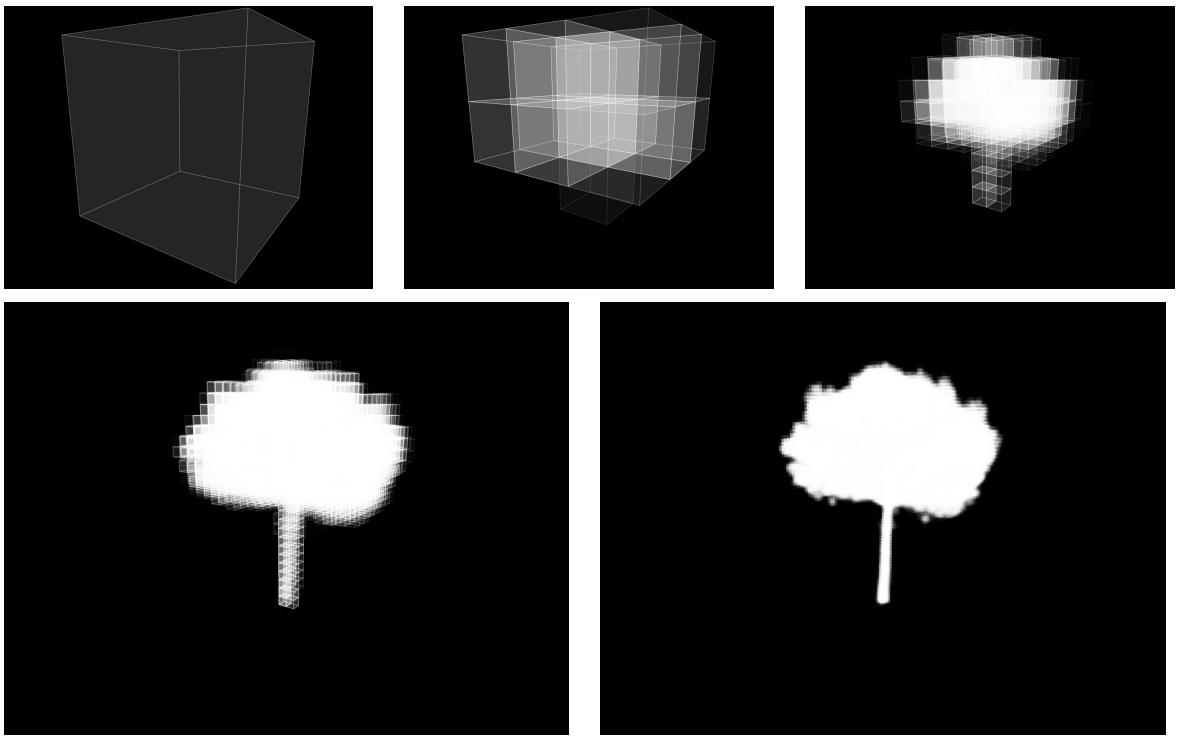


Figure 4.5: Five first levels of refinement of the grid, using the images of the pine tree.

The cells touched by this pyramid are treated. The algorithm performs an optimization step, trying to choose the best values for the opacities of these cells based on the corresponding opacity value $\alpha^p$ of the pixel. Note that empty (transparent) cells are ignored for all steps of our approach.

At each iteration, an initial opacity value $\alpha_i$ (or transparency $t_i = 1 - \alpha_i$) is stored at each cell, together with a $\delta_i^t$, which is initialized to 0 at each cell. We will use the $\delta_i^t$ values to accumulate the movement of transparencies due to the minimization for each pixel in each image. In particular, we accumulate intermediate transparency optimizations for each pixel and cell in $\delta_i^t$, and update all cell transparency values simultaneously at the end of each iteration.

For each pixel of each image we approximate the pyramid by casting a ray through the grid. For each ray we collect the grid cells hit, with initial opacity values $\alpha_i$, $i = 1...n$. We experimented with a larger number of rays, but the resulting increase in accuracy did not justify the additional expense.

Using the same reasoning as for the volume rendering equation (Eq. 4.2), and considering an absorption only model [56], we assume that the pixel opacity value $\alpha^p$ should be:

$$\alpha^p = \alpha_n + (1-\alpha_n)(\alpha_{n-1} + (1-\alpha_{n-1})(...(1-\alpha_1)\alpha_0)..) \tag{4.4}$$

where $\alpha_i$ is the opacity in each cell $i$ traversed by a ray passing trough the pixel $p$. If we recast Eq. 4.4 as a recursive function $f^n$, we get:

$$f^0 = \alpha_0 \tag{4.5}$$

$$f^n = \alpha_n + (1-\alpha_n) f^{n-1} \tag{4.6}$$

$$\alpha^p = f^N \tag{4.7}$$

That we can simplify like this:

$$1 - f^n = (1-\alpha_n)(1-f^{n-1}) \tag{4.8}$$

$$f^n = 1 - \Pi_{i=0}^{n}(1-\alpha_i) \tag{4.9}$$

$$\alpha^p = 1 - \Pi_{i=0}^{N}(1-\alpha_i) \tag{4.10}$$

$$\tag{4.11}$$

If we substitute $t_i = 1-\alpha_i$ (which are the transparency values of the cell), we need to find the values $t_i$ which best fit the equation:

$$t^p = \Pi_{i=0}^{N} t_i \tag{4.12}$$

where $t^p = 1-\alpha^p$. Clearly, we have an under-constrained problem, since we need to satisfy a system of equations (4.12). The number of these equations is equal to the number of pixels in all images. Our goal is to obtain a best-fit solution for the $t_i$ values in the cells. One approach would be to use Eq. 4.12 directly with a standard iterative minimization technique such as a conjugate gradient method.

For reasons of efficiency however, we choose to transform Eq. 4.12 into a sum by taking logarithms:

$$\log t^p = \log \Pi_{i=0}^{N} t_i = \sum_{i=0}^{N} \log t_i \tag{4.13}$$

Eq. 4.13 can be seen as an n-dimensional plane, and desirable values for $\log t_i$ should satisfy this plane equation.

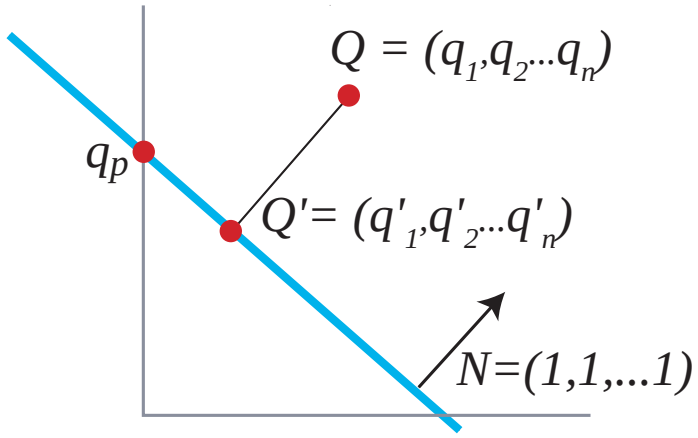



Figure 4.6: Illustration of the projection step for the optimization of the $t_i$ (or respectively $q_i$ values). The illustration is in 2-D for simplicity, but is actually performed in $N$-d, where $N$ is the number of cells intersected by the ray.

Interestingly, the logarithms have a physical interpretation. Transparency $t(s)$ is defined as [56]:

$$t(s) = e^{\int_0^s -\tau(t)dt} \tag{4.14}$$

where $\tau$ is the *extinction coefficient* of the volume. If we approximate the integral using a Riemann sum at intervals $\Delta x$, the discrete transparency $t_i$ can be defined as [56]:

$$t_i = e^{-\tau(i\Delta x)\Delta x} \tag{4.15}$$

Taking the logarithms gives:

$$\log t_i = -\tau(i\Delta x)\Delta x \tag{4.16}$$

Thus the $\log t_i$ values can be seen as a discrete approximation of the extinction coefficients. In what follows we use $q_i = \log t_i$, where $q_i$ is the analog of the discrete extinction coefficient, and $\log t^p = q^p$. We also define the vector $Q = (q_1, q_2, ..., q_n)$.

Recall that the goal is to find a best fit of the $t_i$ values for the cells for the given $t^p$ of the alpha image. The transformation via the logarithm allows us to perform this step by a direct projection onto the $n-$dimensional plane $P$, which has normal $N = (1,1,...1)$ and with intercept $q^p$ on one of the axes (see Fig. 4.6). We can thus simply find the projection $Q' = (q'_1, q'_2...q'_n)$ of $Q$ onto $P$.

We then revert to the space of transparencies by taking the exponent $t'_i = e^{q'_i}$. We will weight this projection by the relative coverage of the ray and the voxels being considered.

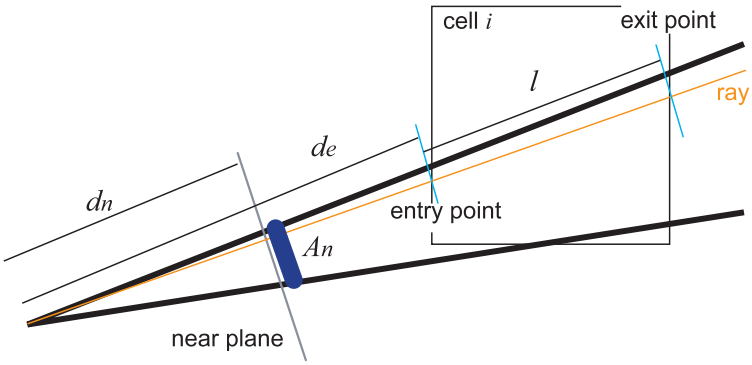



Figure 4.7: Illustration of the weight computation for a cell.

The four corners of the pixel define a pyramid, which we intersect with the near plane (see Fig. 4.7, illustrated in 2D for simplicity). This intersection has area $A_n$; the distance to the near plane is $d_n$. We shoot $N$ rays through a given pixel. A given ray will intersect cell $i$ at a distance $d_e$, and the length from the entry to the exit point of the ray in the cell is $l$. It is assigned an area of $A_s = A_n/N$ at the near plane. We compute the area $A'_s$ of the slice of the pyramid corresponding to the ray using similar triangles: $A'_s/d_n^2 = A_s/d_e^2$ giving $A'_s = A_s(d_e^2/d_n^2)$. We then estimate the volume of the ray/cell intersection as $V = lA'_s$. The weight of the ray will be $w_i = V/V^c$ where $V^c$ is the volume of the cell.

The weighting can be seen as a linear interpolation between the component values of $Q$ and $Q'$. Thus, for a given cell $i$, we have:

$$\delta_i = w_i(t'_i - t_i) \tag{4.17}$$

The stored $\delta_i^t$ value of the cell is then incremented with the $\delta_i$ value, representing the optimization of $t_i$ due to this ray.

To complete an iteration we perform this estimation for each pixel of each image.

When the iteration is complete, for each cell we add the $\delta_i^t$ values, normalized by the sum $w$ of the weights $w_i$, into the transparency values $t_i$ to initialize them for the following iteration. We iterate until convergence, which is defined when the values of $t_i$ change less than a threshold in a single iteration. The result is an estimate of transparencies (and thus opacities) for each grid cell. We try and do a "best-fit" for all input images; this approach will tend towards a local minimum. Note that in practice we recover and store the transparency values $t_i$ from the estimated discrete extinction coefficients $q_i$, passing through the exponential.

Finally, if $t_i$ becomes greater than a *transparency cutoff threshold* when adding in $\delta_i^t$'s, we set $t_i$ to 1; the cell thus becomes transparent and is subsequently ignored. The reconstruction process is summarized as follows:

```
while not converged
    foreach cell i
        init δ_i^t, w_i
    endfor
    foreach image
        foreach pixel
            list-of-cells = cast-ray-into-grid
            create vector Q = (q_1, ..q_n) from t_i's
            project Q onto plane P
            foreach cell i
                δ_i = w'_i(t'_i − t_i)
                update δ_i^t ; w_i += w'_i
            endfor
        endfor
    endfor
    foreach cell i
        t_i += δ_i^t / w_i ; if t_i > thres then t_i = 1
    endfor
endwhile
```

In the Fig.4.8, we can see the evolution of the $abs(\delta_i^t \ / \ w_i)$ mean value at each iteration. As we can see all three examples converge to a value near 0. The iterative process ends when the mean value no longer decreases.

A visualization of slices of opacity values are shown in Fig. 4.9. The process typically takes 3-5 minutes on the models we tested on a Pentium IV 2.7 Ghz PC, and 3-4 iterations are sufficient to achieve convergence.

### 4.5.3 Generation of the billboard textures

At this stage, we have a recursive grid with opacity values which represents the tree. Rendering the tree is a complex task, since we want to preserve high-frequency information existing in the photographs, but at the same time we do not have any additional 3D information other than the

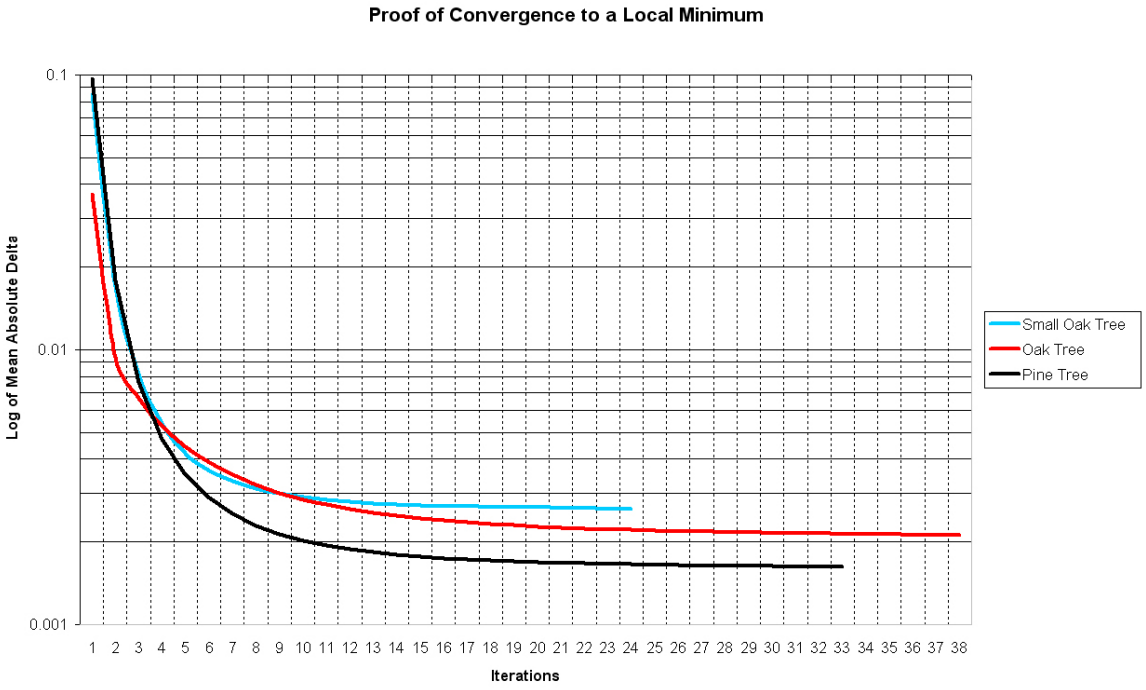



Figure 4.8: Diagram showing the evolution of $abs(\delta_i^t \ / \ w_i)$ mean with iterations.

low-frequency opacity estimate per cell. Given our goal of low-polygon count for the final representation, we cannot subdivide the recursive grid too deeply, since this would lead to an explosion in the number of primitives required.

Our solution is to attach a billboard (a small polygon which is always oriented towards to viewer) to each cell. A naive approach to rendering would be to project a view-dependent texture from the input images onto each billboard, potentially blending between the $n$ closest cameras, in the spirit of Debevec et al. [18] or the Unstructured Lumigraph [11]. Our tests with this approach gave severe blurring artifacts, since texture is repeated across the billboards corresponding to a given pixel (see Fig. 4.10). We present a heuristic solution where we use a billboard for each cell and we generate an appropriate texture for each image.

We generate billboard textures in a preprocessing step by looping through input images and all cells. For a given image, we use the opacity information in the cells, and the texture information in the photograph. The goal is to construct a texture per cell for this image, which is modulated by the opacity information existing in the cell. A separate texture is generated for each cell and for each image.

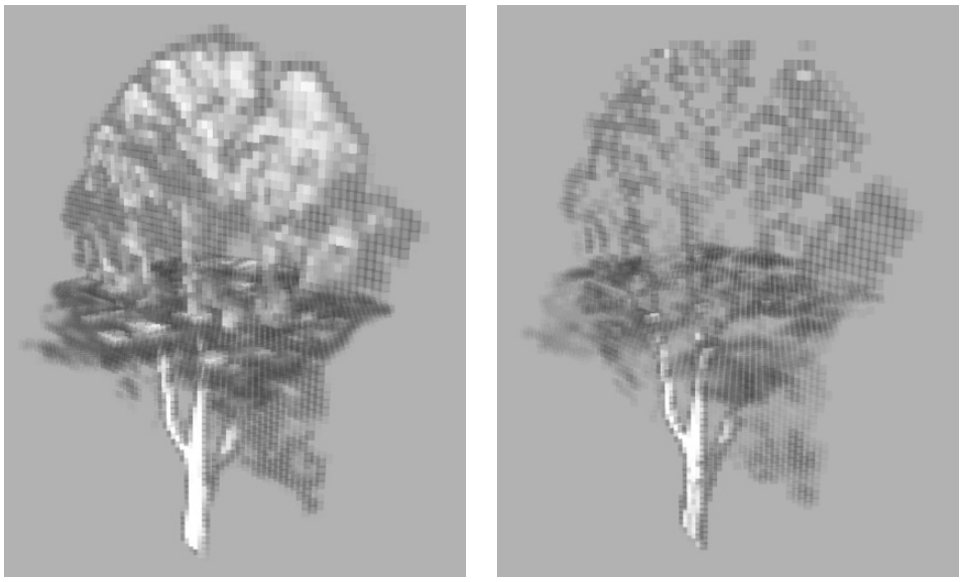


Figure 4.9: Illustration of slices of opacity values at the initialization step (left) and after the end of the reconstruction process (right).

As a preprocessing stage, we render all the billboards into an image which stores the number of billboards $N^p$ written to each given pixel $p$. We use the stencil buffer to count the number of billboards rendered by pixel.

For a given image $i$ we traverse the cells in front-to-back order from the viewpoint corresponding to this image. Then for each cell $c$ we construct an $M \times M$ billboard texture $T$. We typically set $M = 8$ or $M = 4$, which is a good compromise; smaller sizes would require higher subdivision. We project the billboard back into the image, using exact resampling (i.e., computing pixel fractions), we create an $M \times M$ texture from the color image, and an $M \times M$ gray-scale texture from the alpha-image (see Fig. 4.11(a)). Note that we use the images containing the "pure foreground" values $C_f$ (Eq. 4.3).

The intuition behind this process is illustrated with an artificial ideal case, shown in Fig. 4.11(a). Three cells in a row project onto a region of the image containing the three leaves. If all information were available, we could generate the three separate textures with each leaf, one for each cell. To do this, we would visit the cells front to back. For each cell, we would project the cell to the image, extract the texture, mask out only the part corresponding to this cell and assign this as a texture for the cell. We would then update the image and proceed to the following cell (Fig. 4.11(b)). The resulting billboards would avoid the blurring problem mentioned above.

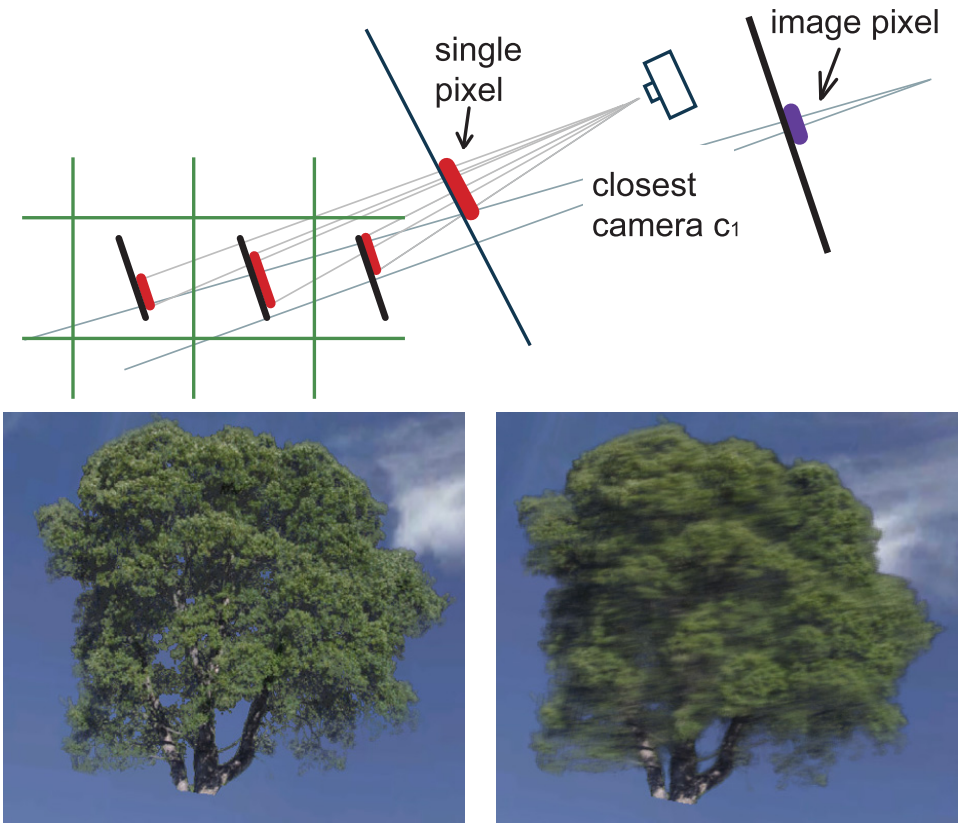



Figure 4.10: An illustration of the problems of naive rendering using view-dependent texturing: texture repetition results in blurring.

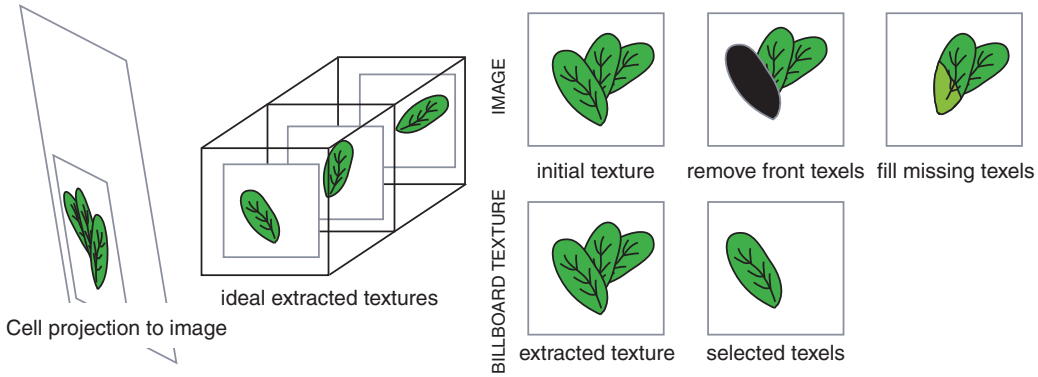



Figure 4.11: (a) Illustration of the ideal case; (b) Texture generation for the first cell, and update of the image.

In what follows, we use the term *pixels* for the values in the RGB and alpha images, and *texels* for pixels of the billboard textures. We project from billboard to image space and vice-versa using appropriate resampling for all values. We use a copy $C_i$ of the (RGBA) image since we modify pixel values.

For a given image and a given cell, we have estimated transparencies $t^c$ for each cell, and we have the $\alpha^p$ (or transparency $t^p = 1 - \alpha^p$) value for each pixel. We also compute the total number of billboards rendered at each pixel using the stencil buffer, and keep track of the remaining number $N_{curr}$ of cells not yet rendered as we process alpha image pixels.

We introduce a heuristic approximation to the ideal case given the minimal information at our disposal, i.e., alpha image pixel values, cell transparencies, and the current number of cells (or billboards) rendered at each pixel.

As in the ideal case, for each image, we render cells front-to-back, and find the region $R$ in the image corresponding to the projection of the cell. For each texel, we decide whether it should be selected using the corresponding image pixels. Our metric for selecting these texels is based on an appropriate comparison of cell and pixel transparencies. Our heuristic assumes that coherent pixel/cell transparencies indicate a higher probability that the pixels correspond to geometry in this cell. We will thus prefer pixels with cell/pixel values which are close according to the metric. To complete the method, we update the pixel color and transparency values after treating each cell.

To implement this approach we first create a quantity from the cell transparency $t^c$ which is comparable to the corresponding alpha-image $t^p$ values. This is done by raising $t^c$ to the power of $N_{curr}$ of billboards not yet rendered at this pixel. This approximates the influence of the current cell on the cumulative transparency of all cells projecting to this pixel: We note this value $t^{cN_{curr}}$. For each pixel we also store a value $t_{curr}$, which is initialized to $t^p$.

For each texel we thus compute the importance, $H_t$, which gives higher values for texels with $t_{curr}$ close to $t^{cN_{curr}}$:

$$H_t = \frac{1}{t_{curr} - t^{cN_{curr}}} \tag{4.18}$$

To complete the approach we update the value of $t_{curr}$ as follows: $t_{curr} = t_{curr}/t^c$, thus "removing" the influence of the current cell as estimated previously using the power computation.

We store an age field with all pixels which is initially 0; recall that texels resample this value. Texels are ordered by age and importance, and the first $\lfloor \alpha^c * M * M \rfloor$ texels are selected to be used on this billboard. For the remaining texels, the age of their corresponding pixels is incremented, thus increasing their chance of being selected later, and spreading out the choice of texels. Some

pixels of the image may remain unselected, but this does not appear to cause significant artifacts.

As seen above, we base the number of texels selected for each billboard on the opacity of the cells. Initially, we tried a random choice of texels given this number, and the results were unsatisfactory. Use of these heuristics clearly improved the result.

The pixels used on the billboard are removed from the copy of the original image, and are filled by interpolating neighboring pixels. A more sophisticated texture generation process could be used for this step, e.g., in the spirit of [27].

The final texture for this cell billboard and this image is thus an $M \times M$ RGBA texture, for which the pixels chosen as valid are assigned the RGB values of the (potentially modified) copy of the original image, and the A value is the value of the $\alpha$ image. This process is summarized in the following pseudo-code:

```
foreach RGBα image i
    make a copy C_i of image i
    render billboards to initialize N_curr at each pixel
    foreach cell c front-to-back
        find projected region R of cell c in image C_i
        create resampled M × M texture T from C_i
        sort texels of T by age and by H_t
        select ⌊α^c * M * M⌋ first texels
        for pixels in R corresponding to selected texels
            t_curr = t_curr / t^c
            replace pixels using texture generation
        for remaining pixels in R, increment age
        for all pixels in R, N_curr = N_curr − 1;
    endfor
endfor
```

The uncompressed textures require a significant amount of memory. To reduce the required texture memory, we dissociate alpha from RGB, and pack two RGB textures if their alphas do not intersect. The resulting memory requirements are about 45% of that originally required. For a typical uncompressed tree of 100Mb, the resulting compressed version requires 45Mb.

## 4.6 View-Dependent Rendering

Once an RGBA texture has been generated for each cell and each original camera, we can render in a straightforward manner. We traverse the cells back-to-front, and render the billboard for each cell with multi-texturing, using the appropriately weighted textures corresponding to the two closest cameras [18]. The RGBA billboard textures are simply blended for each cell visited in back-to-front order, in the sense of the **over** operator [72].

## 4.7 Implementation and Results

Our current implementation uses a trigrid cell structure [26] developed at REVES to model the tree, which subdivides into 3x3x3 subcells (an octree could also be used). This hierarchical structure defines a regular recursive space subdivsion analogous to octrees, that is memory efficient for our purposes. Currently, we subdivide all non-empty cells down to the maximum subdivision level. Empty interior cells are not subdivided.

We present three examples of real captured trees. Our approach is however better appreciated in an animation or walkthrough. An AVI showing a quick explanation of the algorithms and an interactive walkthrough can be downloaded from:

http://www-sop.inria.fr/reves/publications/data/2004/RMD04/

to better appreciate the results. All timings are on a Pentium IV 2.7 Ghz PC.

The first tree we have captured is an oak. We took 22 pictures of the tree, and generated an equivalent set of masks. The estimation took about 15 mins for a 4-level subdivision of our grid, resulting in 51,000 cells with 8x8 billboards, using a cuttoff transparency threshold of 0.94. The texture generation phase took 30 minutes, and resulted in 150Mb of texture, which after packing (which took 31 min), reduces to 56Mb. We show two new (non input) views of the oak in Fig. 4.13(top).

The second tree we have captured is a pine, shown in Fig. 4.13. We took 18 pictures of the tree; estimation took 15 minutes for a level 5 subdivision of our grid, resulting in 110,000 cells, with a cutoff transparency threshold of 0.9. The texture generation for 4x4 billboards took 15 minutes, resulting in 58Mb of texture. After a 17 minute packing process, the final compressed tree requires 22Mb. We show two new (non input) views of the pine in Fig. 4.13(middle).

The third tree we have captured is a small oak, shown in Fig. 4.13. We took 18 pictures of the tree; estimation took 15 minutes for a level 5 subdivision of our grid, resulting in 110,000

Figure 4.12: Two new (non-input) views of the captured oak tree.

cells, with a cutoff transparency threshold of 0.9. The texture generation for 4x4 billboards took 15 minutes, resulting in 58Mb of texture. After a 17 minute packing process, the final compressed tree requires 22Mb. We show two new (non input) views of the pine in Fig. 4.13(bottom).

Use of the transparency cutoff threshold greatly reduces the number of cells (by a factor of 2 and 8 respectively for the oak and pine), making the resulting trees smaller in memory and faster to display. Initially, some cells incorrectly receive non-zero opacity due to missing information such as the small number of cameras, or occlusion in the images. Use of the threshold makes some of these transparent during reconstruction, improving the visual quality.

## 4.8 Discussion

Our method gives very promising results, and we believe that it will be useful as a fast way to capture existing trees and to produce high quality interactive renderings of trees with low polygon count models.

One current difficulty is the need to have a good background to be able to acquire satisfactory alpha mattes. We currently add a uniform color background (a sheet in our case), where possible to help the alpha matte algorithm (see Fig. 4.4(left)). This helps, but is not always a practi-

cal solution. Using the Bayesian approach of [14] may improve the mattes extracted, and we hope that other researchers will develop a more stable alpha extraction approach, perhaps specifically adapted to the case of trees.

Currently, the generated textures partially contain the original lighting information. However, since we manipulate the textures in the generation stage, it is clearly possible to develop some approximations which would permit relighting. In particular, the existence of opacity information should probably allow us to approximate light distribution, and perform approximate lighting or relighting calculations. A stochastic approach to shadow removal and normal estimation [39] could probably be applied. The relatively random nature of the tree leaf distribution should make this easier to achieve than for more structured objects. Exact shadow shapes are harder to identify, and thus the error should be less perceptually evident.

Using a low cutoff threshold makes some cells transparent, but reduces blurring significantly. This is a tradeoff, since depending on the level of subdivision and the number of cameras, existing branches or leaves may be removed. This effect is illustrated in Fig. 4.15 and in the video. Even when features are removed, for example with threshold 0.9, the tree remains very realistic and remains very similar to the input photographs; however blurring is almost completely eliminated. Investigating heuristics to find the best compromise is a promising avenue of research.

Finally, our approach is best suited for viewing the captured tree at a distance no closer than that of the cameras of the input photographs. Closer viewing would require additional processing. Conversely, when viewing from a distance, multi-level rendering could be used, by computing levels-of-detail for the billboards in the hierarchy.

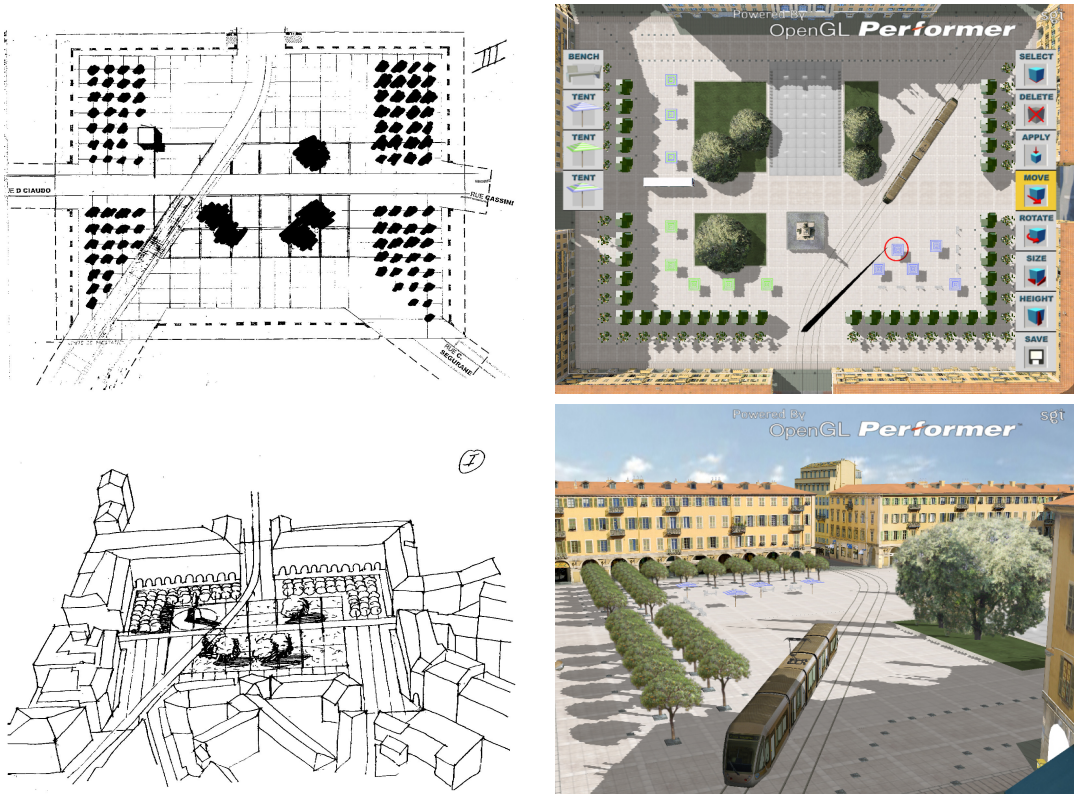
Figure 4.13: From top to bottom: two new (non-input) views of the captured oak tree, two new views of the captured pine tree and two new views of the captured small oak.

Figure 4.14: Two new (non-input) views of the captured pine tree in a synthetic environment.

Figure 4.15: Side by side images of the reconstructed oak tree (synthetic view) using thresholds of 0.9, 0.94, 1.0 (left to right).

# Part II

# Creation and Evaluation of a Real World Example




In this part of the thesis, we explain the creation and evaluation of a photorealistic virtual environment inspired from a real world project. As seen in top figure we have been heavily inspired by standard sketches used by architects to create an appropriate metaphor between standard design and virtual design. The purpose of the project is to provide a very realistic design tool, that does not change the normal workflow of an architect, giving the possibility to see the resulting environment at the same time as the environment is been designed.

In the next chapter we explain in detail the process of creation of this VE. In Chapter 6 we evaluate the final VE to verify the importance of the realism and the use of Virtual Environments in this kind of real world project.

# Chapter 5

# Creation and Display of Photorealistic Interactive Virtual Environments

In this chapter we introduce an image-based 3D capture process for the creation and display of photorealistic virtual environments (VEs). The resulting VEs aim to realistically recreate existing real-world scenes that can be displayed in a range of immersive VR systems using a high-quality, view-dependent algorithm and further enhanced using advanced vegetation, shadow display algorithms and 3D sound. The scene, an urban environment, was chosen according to real-world applications in the area of urban planning/architecture. The users are able to reconstruct or manipulate elements of the VEs according to their needs, as these have been specified through a detailed user requirements survey. Furthermore, a user task analysis and scenario-based approach has been adopted for the design of the virtual prototypes and the evaluation, which is presented on Chapter 6. This work was developed in the context of the EU-funded research project CREATE and the first examples of the prototype system in use are described and demonstrated in this chapter. Since the CREATE project is a multidisciplinary project, it is the result of many peoples work. Our work in the project was related with rendering algorithms, perspective shadow maps [94], point-based rendering [93] and view-dependent texturing [17]. The different parts of the project developed by different partners are; spatialized sound rendering [99] developed by the REVES/INRIA Sophia Antipolis, haptic interface developed by Percro, crowd animation and rendering developed by UCL, panorama capture and image modelling developed and commercialized by RealViz. This chapter is heavily based on the publication [24]

In this thesis, we worked on the view-dependent rendering and its integration into the

common platform, the development and adaptation of the system across platforms, and in particular on the INRIA Linux/Windows workbench as well as the CSTB SGI/Onyx RealityCenter. Part of this thesis involved work together with Alexandre Olivier-Mangon (the modeller of the project) in the improvement and adaptation of the capture algorithms for this challenging application. Finally, a major part of the thesis involved participation in the system design and development, throughout the duration of CREATE, so that the original, research-oriented, code-base could handle the requirements of the more involved applications tested. The system design and development also involved important choices to support the multiple platforms, and enable efficient system integration from the 6 other partners of the project.

## 5.1 Introduction

The creation of realistic 3D models has, typically, been a time-consuming labour-intensive manual process, requiring well-trained modellers. Recent advances in computer graphics and vision [18] allow the creation of realistic models based on photographs, that are of very high-quality. Many methods [34, 11] have been developed to perfect and display such models. These methods, however, have been specifically developed for conventional computer graphics rendering and not for real-time display in immersive virtual reality systems. Even if the integration of realistic models in a real-time VR environment is achieved, the resulting captured environment looks static while the display of additional elements such as shadows or vegetation is particularly difficult. Moreover, apart from the visual effect, the overall perception of realism can be further enhanced by the addition of 3D sound; however, VEs have become more and more complex recently, and real-time audio simulation requires novel algorithmic solutions to allow its integration.

The emphasis on enhancing realism in a virtual environment, due to the extended effort and performance required to achieve it, leaves little room for the development of meaningful interactivity. Highly interactive virtual environments, such as virtual prototyping systems, are usually comprised of simplified 3D models that are optimized as to not consume too many graphics rendering cycles. Hence, the interactive manipulation of photorealistic elements has remained an elusive goal.

The design of realistic and highly interactive virtual environments is a challenging task since the combination of realism and interactivity adds extra difficulty to development and implementation. We believe that both photorealism and interactivity are essential for a number of VR applications. We have therefore chosen to address this challenge through an iterative, user-centered

approach that carefully takes into account the application and user needs.

In this chapter we present our approach and first solutions to this issue, focusing our attention on two areas, realism (both visual and auditory) and interactivity. For realism, we introduce a novel workflow for capture and realistic display of existing real environments based on photographs. By introducing view-dependent display techniques we achieve unprecedented realism for VEs. We further enhance these environments with efficient display of realistic vegetation, dynamic interactive shadows and realistic spatialized 3D sound. A number of design choices had to be made to allow these algorithms to integrate well in the end-user immersive VR systems. A first prototype of the system has been implemented for different immersive displays (an immersive workbench, a curved-screen and a cubic immersive display).

## 5.2 Related Previous Work

We briefly discuss a selection of the most relevant computer graphics/vision approaches for data capture and display. We then present a rapid overview of previous work related to the chosen application domains.

### 5.2.1 Model Creation, Realistic Display and VEs

The premise of our modelling approach is the creation of realistic 3D models from images that can be displayed in VEs. In computer graphics, a number of Image-Based Modelling and Rendering techniques have been developed (e.g., [34, 18]). Despite recent advances (e.g., [11]), these techniques, as explained before, usually require special purpose display methods. Such approaches can be hard to integrate into traditional VR systems, which have numerous software components to handle the complexity of the hardware platform (stereo, tracking, different devices etc.), and are usually created with standard scene-graph APIs such as OpenGL Performer$^{TM}$ [86]. As a result, the application of such techniques into an integrated VE is rare.

We have chosen to use a modelling-from-images approach (i.e., [18], ImageModeler$^{TM}$ from RealViz [78]). Other forms of 3D scanning (laser scanning etc.) could potentially be used for some of the applications we examine; each approach has different tradeoffs. In the context of situated activity in a VE, we believe that the simplicity and cost of capture from photographs, and the quality of the resulting models justifies our choice. This does not hold true for all applications (for example, where millimeter precision is required). In the long run, we believe that combinations of

several different acquisition techniques should be used.

In this work, we have chosen two visual enhancement components which we have integrated into our systems, notably view-dependent textures, shadows and vegetation. For view-dependent texturing we adapted the view-dependent algorithm [17] to the use in a stereo based display. For shadows we have adapted perspective shadow maps [103, 94]. For the display of vegetation, we use a mixed point-based/polygon rendering technique which allows us to handle complexity efficiently. Point-based rendering [46] has recently seen growing interest in the research community (e.g., [82, 101]), but has not yet been used in VE's to our knowledge.

The inclusion of 3D spatialized sound [32] is paramount in achieving a truly convincing virtual reality experience [43]. However, integration of 3D sound in virtual reality applications remains limited due to the lack of standardized tools and heavy signal processing costs that do not scale well with the complexity of the auditory scene.

Figure 5.1: Two faces of a panorama cubemap used for image modelling.

### 5.2.2 VR and Design

As VR technology becomes commonplace, there has been a proliferation of VR in fields such as design, education, and entertainment or, in other words, areas where VR applications are more easily available to and accessible by the general public. In design, VR has been used where conventional media are ill-suited to represent the work processes in ways that make them easy to

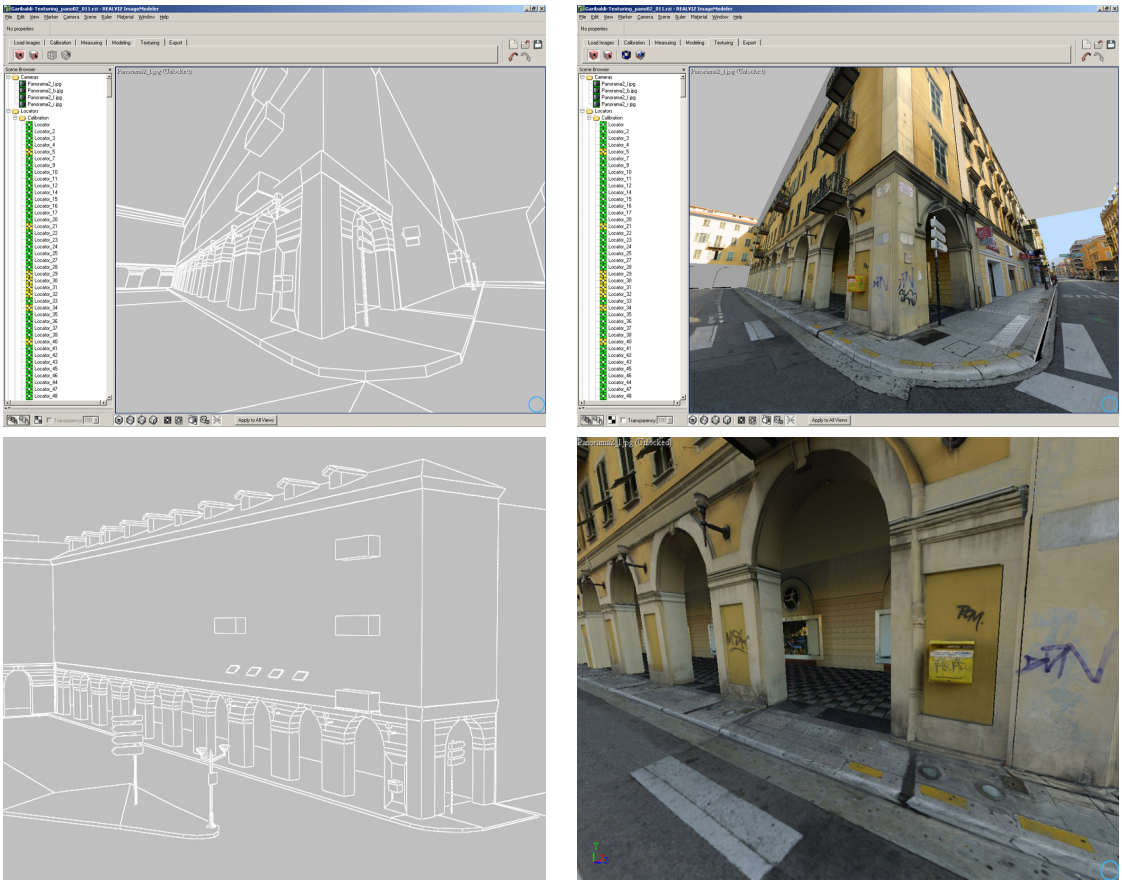


Figure 5.2: Wireframe of the resulting model and a view with extracted textures.

visualize. VR, with its immersive and interactive properties, can offer possibilities and solutions that are otherwise very difficult to obtain.

We have chosen to focus our efforts in the design area, and specifically in a urban development project. Our choice has been motivated by the fact that this task is a real-world project that is currently in progress and in need of high-level tools and presentation means that will speed-up and facilitate the work or help in better dissemination of their cause.

For this application, the City of Nice in France and Greater Nice-Cote d'Azur Urban Community (CANCA) have currently commissioned the construction of a tramway that will be integrated in the measures taken to improve urban life by reorganizing and augmenting open space, improving accessibility for pedestrians, stimulating economic life, and helping preserve and display the city's architectural heritage that is endangered by car pollution. The city administration would

like to be able to present several designs of the suggested urban intervention, allowing shop owners, local inhabitants, and travellers to evaluate the result from viewpoints they choose (for example, from street level or from the office they work in) or through "guided tours" of typical usage patterns of new infrastructures (for example, how a typical traveller will arrive at a station or cross a street). The architects who have undertaken the development of the project, wish to thoroughly evaluate the effects of the urban intervention on the environment and the landscape of the city, and in particular use interactive VEs to choose amongst different designs.

## 5.3 Creating a Realistic Virtual Environment

The creation of textured 3D models from photographs results in low-polygon count models which have a very high level of believability. The resulting 3D environment can be edited using standard 3D modelling tools, and virtual elements can be added as needed.

With such an approach, the textures are view-dependent, since the photograph is valid only from the point it was taken. To counter this problem, we introduce a view-dependent capture and display workflow.

### 5.3.1 View-Dependent 3D Models from Images

The data capture workflow starts by choosing a number of viewpoints around which VE activity will take place. Panoramic images are then created by shooting a number of photographs around the viewpoint and "stitching" the images together[1]. These panoramas are rendered as cubemaps and loaded into an image-modelling program, such as ImageModeler$^{TM}$. An example of 2 (out of 4) faces of a panorama cubemap are shown in Fig. 5.1. Other similar systems (e.g., Facade [18], PhotoModeler [95]) could also be used. The cameras of images/faces of the cube are calibrated, and 3D models can then be constructed.

The initial phase of the workflow is the construction of an untextured 3D model of the scene, for example Fig. 5.2. This model is the same for all the panoramas used. Textures are then extracted by projecting the image back into object space. Textures are edited at various stages of the process for each view. The initial images of the panorama can be edited in an image-editing program to remove undesirable elements such as people or cars etc. Since we have multiple views, there is an inevitable time and exposure difference in the photographs. To deal with this, manual

[1] We use a Kaidan head on a tripod and REALVIZ Stitcher$^{TM}$ for this phase, but any standard tool can be used.

editing of the panorama images may be needed to equalize colors and to modify the position of shadows. Additional editing (e.g. “clone brushing”) is required at the texture extraction phase, for example in regions that were hidden in the original views.

Evidently this process is still not as simple or as rapid as we would like. As an example, the calibration and modelling phase for the urban square (Place Garibaldi in Nice) shown in Fig. 5.2 took about 3 weeks. The area covered, however, is quite large and the amount of geometric detail is significant. Texture extraction, including all the editing phases, takes about a week per panorama.

The result of this process is a single 3D model, and one set of textures per panorama (viewpoint). As discussed later, this allows view-dependent display of the models, resulting in significantly higher quality renderings.

We are actively pursuing ways to improve and simplify the capture process. In particular, we have introduced a novel approach presented in Chapter 3 which uses projective textures instead of reprojected/resampled textures. The method begins by creating visibility layers for each image, thus facilitating the work of the artist and significantly reducing the texture memory overhead. The fact that texturing takes place entirely in the image-editing program and in the original image space, significantly accelerates the work of the artist.

Nevertheless, problems that arise from the use of view dependent texturing exist and are handled on a case by case basis. We have discovered that a large number of view dependent textures have to be used when the user travels long distances in the VE because in this case the parallax of the images is too large and image coherence is a problem. Depending on the importance of the object being textured a detailed modelling effort can alleviate the problem.

In the final virtual environment view-dependent textures are only used for nearby objects, for which the geometry is too much complex to be modeled in detail. The reason why we adopt this solution is the heavy use of texture memory by view-dependent textures because the users can travel long distances in the VE, and some of the systems used where limited to 64Mb of memory.

### 5.3.2 Data Acquisition for the CREATE Project

The selected site, located in Southern France, was photographed between November 2002 and May 2003. A section of the site of Place Massena (Fig. 5.4) in Nice, France was initially chosen because of its central location in the development of the tramway project. After involving the end-users (the architects and urban planners) more closely in the design through the user task analysis, an additional location was added, that of Place Garibaldi (Fig. 5.1 and Fig. 5.2).

### 5.3.3 Enhancing Realism of the Virtual Environment

Given the view-dependent textured 3D model, we need a rendering algorithm and an adaptation of display for traditional VE systems. We discuss the two approaches developed for view-dependent texture mapping for immersive systems.

Once the VE has been constructed, we can augment it with virtual elements. For enhanced realism we have added interactive shadow display and vegetation. We have also included spatialized 3D sound which greatly enhances the effect of immersion in the environment. Finally, we also discuss some issues of software engineering and integration that simplified our solution.

#### View-dependent Texture Mapping

A texture extracted with respect to a certain viewpoint is not valid from others. Nonetheless, the realism provided by textures from photographs is such that the user is often "sufficiently fooled" for non-negligeable motion. In essence enough 3D geometry needs to be reconstructed in the near field, and textures from appropriate viewpoints need to be used.

The principle of view-dependent texture mapping was developed by Debevec et al. [18] as an offline process, who later adapted the approach to interactive viewing [17]. The approach used there was based on projective textures, which require that the geometry be subdivided from each viewpoint used such that hidden parts are separate geometrically.


Figure 5.3: Two views of a view-dependent object, the 3D model of the face of the statue with the relief is simply a plane.

Due to this restriction and the tool workflow we used, that is ImageModeler$^{TM}$ [78] for modelling and OpenGL Performer$^{TM}$ [86] for display, we developed a first approach which is based

on texture blending. This entire view-dependent display algorithm is implemented within a scene graph structure, the Performer scene graph. We added a node which handles the fact that a single geometry has multiple textures. The draw callback of the node then computes the closest viewpoint and chooses the appropriate nodes with corresponding textures to apply and computes the percentage of the blend factor for each node, using the current camera position. We give more details about the implementation in Section 5.3.4.


Figure 5.4: Left, the VE with the textured version of the above model. Right, the final VE after the addition of virtual elements, realistic shadows and lighting.

The second approach uses projective textures using layered images as described in Chapter 3[2]. As described above, the capture process provides images layered by visibility, thus avoiding the need to subdivide the geometry according to each viewpoint. Each layer of the image is the projective texture corresponding to the appropriate part of the model. A similar blending approach is used for multiple viewpoints.

### Lighting, Shadows and Vegetation

An important requirement of high-quality rendering is consistent lighting between virtual, inserted elements and the captured geometry corresponding to real objects. For this we have integrated illumination from a sky and sunlight model [74] to achieve consistent lighting of virtual objects. We use efficient acceleration methods to sample the sky hemisphere and achieve the desired effects (see Fig. 5.5).

Some of this accelerations are the creation of a diffuse map using the sky-dome lighting

[2]Please note however, that the layered textured approach was not integrated into the system used for evaluation by the architects.

and the precomputation of visibility maps in different places of the environment, the computation of visibility in other places is done by interpolation.


Figure 5.5: The tree is lit with our implementation of illumination from a sky and sunlight model.

Shadows add a very important dimension to the perception of realism in the environment, particularly when seen on an immersive display. We have developed high-quality shadowing algorithms, which we achieve by using perspective shadow maps [94] integrated into a VR system. This integration requires the adaptation of the basic algorithm to the particularities of scene-graph based solutions (see Fig. 5.6).

To display vegetation we have integrated a point and line-based rendering approach [93, 19] for complex vegetation geometry which is in the distance. Again, a new Performer node is created containing the point array. The array exists next to the original geometry in the graph, and when sufficiently small, an appropriate prefix [19] of the point set is displayed instead of the complex geometry (see Fig. 5.7). As a result, we can treat complex models of trees or bushes efficiently permitting interaction with these elements.

**Sound**

We have designed a 3D sound system that scales well to large numbers of sound sources. Based on spatial and psychoacoustic grouping rules, we dynamically build clusters of sound sources using traditional clustering techniques [49, 36]. Each cluster is then rendered as a single point

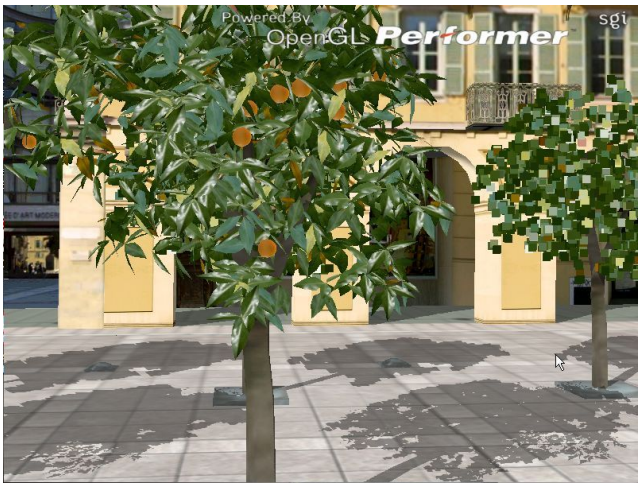



Figure 5.6: Screenshot of a detail of the shadows generated by the perspective shadow map algorithm.

source. The sound signal for each cluster is obtained by individually pre-mixing the signals of each source within the cluster. Contrary to prior related work [35] our approach performs dynamic clustering that can be used for resource allocation.

We also take into account the content of the source sound signals to enhance the accuracy of our grouping strategy. Our approach currently allows for rendering 5 to 10 times more spatialised sources than current state-of-the-art consumer hardware alone which is limited to 32 or 64 simultaneous 3D sources.

Being able to render a large number of spatialised sound sources gives the opportunity to increase the realism of virtual worlds by including virtual image-sources that account for sound reflection and diffraction or model spatially extended sound sources as collections of point-sources (for additional details on the technique, see [99]). These algorithms are implemented within an audio rendering server that interfaces with our VR library and can either run locally or on a distant machine through a network connection. The core of the server is platform-independent while access to audio hardware is implemented through plug-ins, ensuring portability.

### 5.3.4 Software Engineering Issues

The software engineering aspects of this work are considerable. We have based our system on VRCO's CAVElib$^{TM}$ [100] and SGI's OpenGL Performer$^{TM}$ [86] scene graph library. We have used an extensible scripting language XP, originally developed at the Electronic Visualization


Figure 5.7: Trees rendered using point based rendering. The real geometry of the tree is used only when needed (close view), otherwise points are used to render the tree (far view).

Laboratory [68] and extended by the Foundation of the Hellenic World [1]. This language allows the addition of new nodes (such as those for point-based rendering or view-dependent objects), implemented in Performer and directly accessible in the VE scripts.

The benefit of this approach is that these are standard software components, which are provably portable across different VR systems. Initially developed on a PC-based Linux Barco BARON workbench, have been successfully displayed on an SGI IRIX Onyx4-based Reality Center$^{TM}$ and a CAVE-like display (see Fig. 5.8). It has been also ported to a Windows for the same PC based system. The porting processes did not involve major changes to the code. However, the specifics of different hardware, operating system and compiler configurations did require a significant effort on our behalf, to ensure that the system functioned correctly on all platforms.


Figure 5.8: The VE of the urban planning/architecture application as displayed on an immersive curved-screen display and on a workbench.

The use of such libraries does have its constraints however, since structured scenegraphs typically impose a number of choices concerning rendering modes etc. For example, implementation of the shadow mapping algorithm required modification of the basic rendering channels, and restricts the efficiency of multiprocessing. Nevertheless, in our experience, the benefit of having working software components for all the VR device issues largely outweighs the difficulties encountered.

In the final stages of the project, haptics were added in a similar manner as XP objects, allowing a seamless integration in a very short period of time.

Here we will introduce the new nodes added to the XP scripting language. For a more technical details about them see Apendix A.

### Lighting

An additional daylight node is added in the XP-graph. This node provides an implementation of the sky model [74]. The node object takes a time of day and earth position parameters, and will give an intensity in a given direction. Sunlight is calculated simply as a directional light source in the appropriate direction.

At loading time, all virtual objects compute diffuse lighting at their vertices using this object. This includes visibility computation to a relatively large number of samples of the sky dome using ray tracing. For specular lighting, sunlight will be represented by a standard OpenGL directional light, and will be computed using the hardware as the viewpoint changes.

When an object moves, the diffuse component of lighting needs to be recomputed. Depending on performance, this is done with or without recomputation of visibility, or most probably with lower quality than at load time. To effect this update, all virtual objects are stored under a special XP node, which stores the current transformation and matrix information of the node. Thus, during the application traversal of the XP graph, the transformation and matrix of the node will be compared to the stored version, and if a modification is observed, the node of the virtual light will request an update of lighting for the object under it in the graph, by querying the daylight node.

### Shadows

Since we are treating large exterior environments, but manipulating details of the scene in which high-quality shadowing is required, it is necessary to have both high-quality and fast shadow algorithm. To date, the ideal combination appears to be perspective shadow maps [94].

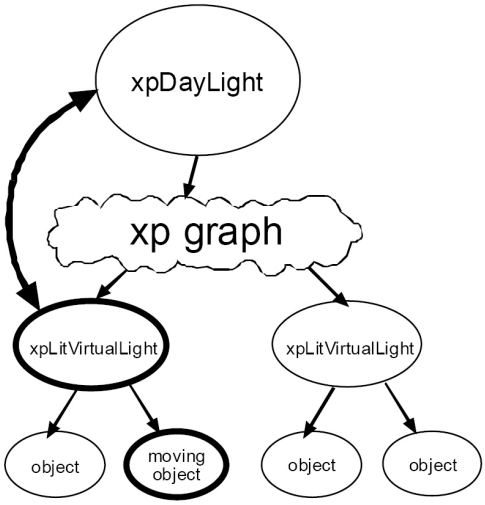



Figure 5.9: Graph showing the node configuration added to use virtual sky-dome lighting.

To adapt the standard virtual perspective shadow maps, we need to make the following observations:

- All objects, and specially captured objects, cast shadows.
- The number of potential shadow casters for most scenes, is low compared to the overall polygon complexity, making the light rendering pass relatively cheap computationally.
- Care must be taken to adjust the shadowing parameters to make the shadows consistent in intensity with those existing in the textures of the real scene.

In implementation terms, a separate channel (pfChannel in Performer) is required for rendering from each light source to generate the perspective shadow map. We call these the "light channels". The light channels are rendered first, and the depth buffer needs to be copied in the postDraw to an appropriate buffer which will be used by the texture mapping/depth comparison process. This required a modification of central XP display loop to handle multiple channels and the rendering in the appropriate order.

In addition, we have two groups of "casters" and "receivers", which cast and receive shadows. During the display loop of the normal rendering channel, potential receivers are rendered differently, using the shadow map extensions for the hardware supporting this (SGI Onyx Infinite Reality and GeForce 3 and higher). This is similar in design to the pfShadow node in

Performer [86]. The caster and receiver groups are defined explicitly in the XP script. Limiting the number of receivers and casters significantly enhances performance, and is necessary for the complex environment we treat.

#### View-Dependent Texture Mapping

For each object a set of (camera, texture) pairs is assigned. Currently this is supported by sets of separate input files, containing the textures with respect to the camera in question.

For any given object with associated textures, we display the polygon(s), and then either a single texture or blend between a set of textures depending on the available input. More sophisticated blending techniques. Using depth information can also be used. For objects with geometric level of detail, we may decide to descend into the hierarchy for display.

We distinguish three typical cases for the display of captured objects:

- The polygonal representation is rendered with a single texture. This is typically the case for the distant objects. In this case we use a standard display algorithm.
- A small number (typically 2) of neighboring views are blended, depending on the current camera angle and its variation from the angle of capture for each view. Depth information, if available, can be taken into account to improve the quality of the result, by warping.
- A larger number (typically 3 or 4) views are chosen, for objects which are manipulated in the near field. This requires a larger number of views for each object.

For a given arbitrary viewpoint, we need to choose the appropriate set of views which will be used, either blended or warped for display. A number of criteria are possible, including distance of the current viewpoint to the views used for capture, or difference in viewing direction between the various views. In addition, appropriate blending factors have to be used in the case of blending. The blending factor is computed using the angular criteria explained in [17].

## 5.4 Adding Interactivity

In addition to the development of visually rich virtual environments, our user-centered approach presupposes the development of usable applications that include a number of interactive capabilities designed to meet real-world user requirements. The design of interactivity has resulted

in a set of representative user task scenarios that have been evaluated and refined, as described in Chapter 6.

The interactions developed for the urban case where the addition of interactive features may allow users to dynamically investigate alternatives by directly manipulating environmental elements (umbrellas and benches for example). The architects of the tramway project in Nice used the VE's interactive features to make judgments, examine alternative development scenarios, and experiment with different possibilities and solutions during their virtual prototyping process (observe, for instance, the aesthetic and functional effect of urban interventions). These processes, if judged successful for part of a site, may be repeated for a larger area. It is believed that by such direct manipulation in an environment, which preserves close-to-real spatial relationships, users can better understand the effect of specific actions, achieving better overall comprehension of the task and, through the active iterative approach, gain a better sense of the relative dimensions or interrelationships between the various elements under examination.

### 5.4.1 Overall Process

As mentioned above, for the Garibaldi scene (which is an entire square of a city), the overall photography, calibration, modelling and texturing required about 2 person-months. The addition of interactivity is performed via the creation of XP scripts, and required 3-4 weeks. A trained artist can perform all of these steps without the need for a programmer. Nonetheless, our system is still a prototype, so in practice many of the steps were quite cumbersome. Many aspects could be automated and the user interface could be much simpler. Given our accumulated experience, we believe that the time required, with the current system, could be reduced at least by half.

## 5.5 Discussion

The introduction of high-fidelity visual and auditory environments is a significant step in allowing VE's to be used in real-world tasks. The addition of view-dependent texturing for realistic and interactive viewing of existing real-world objects in VE's, realistic vegetation, shadows and 3D sound, all contribute to an increased sensation of realism and immersion. Coupled with interactivity, we believe that these elements will permit users to make more informed decisions and improve their capacity to learn and design.

Finally, the user studies done using this developed VE to ensure that the resulting appli-

cation environments meet their users' needs and provide virtual reality tools that address real-world situations are explained in the next chapter.

## 5.6 Conclusion

In this chapter we presented the workflow for capture and display of realistic virtual environments, oriented towards applications in urban planning. We discussed the capture process in detail, using modeling-from-images techniques, and the use of high-quality visual and auditory display algorithms to create a convincing interactive VE, developed in the context of the EU IST project CREATE. In addition we discussed the various aspects of system design and development which were developed in this context which involved close contact with a real-world project, that of the Nice Tramway.

As can be seen in the images in this chapter, the resulting environments provided very high quality images (and audio). The prototype workflow which was developed, from capture to display, was sufficiently mature to be used in the evaluation experiments described next. We believe that the optimization of such a process, in time, effort and cost will result in a standardized workflow. We hope that it will become widely available, and that it will be used as a valuable tool for urban design, as well as in other application domains.

# Chapter 6

# Evaluation of a Real World Virtual Environment

In this chapter we present a participatory design approach to the development of a VE, with an iterative, user-informed process throughout the entire design and development cycle. A preliminary survey was first undertaken with end-users, i.e., architects, chief engineers and decision makers of a real-world project, followed by a study of the traditional workflow employed. We then determined the elements required to make the VE useful in the real-world setting, choosing appropriate elements (image-based textures for facades, realistic vegetation, shadows, 3D sound and crowds) to develop a rich and audiovisually realistic VE. Our participatory design approach guided the development of an appropriate interface and an evaluation methodology to test the overall usability of the system. The VE was evaluated both in the laboratory and in the users' natural work environments. The results suggest that participatory design can help ensure that the VE will meet the practical needs of its end-users; that appropriately chosen multiple views of the scene are useful in urban planning, and that the realism enhancements introduced increase the effectiveness of VEs for urban planning.

As in the previous chapter, the results reported here are part of a collective research effort [23]. Our contribution to this work was mainly in the design of the interface, and the adaptation and development of the entire system for the experiments.

## 6.1 Introduction

The majority of Virtual Reality (VR) applications developed today are products of research that are either prototypes created within very specific contexts or are used for presentation purposes. Despite the promise and the development activity of over two decades, there has been a considerable lack of real-world applications. The issues regarding the deployment of VR in everyday work contexts have been discussed many times and continue to revolve around the familiar practical difficulties: setting up special and costly hardware within facilities that are not easily transportable, requiring special teams of developers and maintenance staff, but also providing the high-level tools that will support users in their complex tasks [62] and can succeed in establishing a collaborative VR work environment amongst individuals of different disciplines [53].

Experienced practitioners in the field of VR have indicated that to work effectively in a virtual environment (VE), the application content must include the ability to access or change environmental/system/meta parameters, create and manipulate particular objects, perform analyses, and export changes to permanent storage [92]. While the current state of VE development has advanced its techniques to support these tasks, rarely does one find complete VEs that achieve both a high-quality photorealistic real-time environment and the level of interactivity required to carry out sufficiently complex real-world tasks.

Our goal is to create a VE infrastructure supporting both audio and visual realism and a high level of interactivity, and to situate and evaluate its utility in an appropriate *real world application* context. We chose the domains of architectural design and urban planning (UP), where realism and interactivity are inherent requirements of the work process. A detailed user requirements analysis with architects and urban planners [81] confirmed the suitability of our choice and led to a detailed study of the existing workflow in these domains. A key element throughout this work has been our close collaboration throughout the entire project with the end-users of a real-world urban planning project, involving the redesign of public spaces as part of the construction of a new Tramway in the city of Nice in France.

Following our initial user needs analysis that guided our choices and design, we proceeded with the development of a complete VE that was continuously informed by the participation of the end-users. Development and evaluation advanced together in order to determine the elements required to make the VE useful in the context of the real-world project we were fortunate to connect to. The resulting VE provides a design and brainstorming tool for architects and decision makers, that also serves as a consensus building tool and as an impressive means for presentation of the

project.

In addition to the participatory design approach, our work includes in-depth formative and preliminary summative evaluation. Evaluation was performed based on observation, questionnaires and interviews with architects that took place both in a controlled lab experiment setting as well as in the "field", i.e. the actual work environment where the VE was used in decision-making meetings of the urban planning project.

In summary, we present a participatory approach to the design of a VE, with an iterative, user-informed process throughout the entire design and development cycle. A preliminary survey was undertaken with end-users (architects, chief engineers and decision makers), in the context of the real-world project, followed by a study of the traditional workflow employed. After this first phase, we determined the elements required to make the VE useful in the real-world setting. As a result, we developed a rich and audiovisually realistic VE, allowing us to test the significance of added realism, in particular image-based textures for facades, vegetation, 3D sound, shadows and crowds. Our participatory design approach guided the development of an appropriate interface, as well as an evaluation methodology to test the overall usability of the system. The VE was evaluated both in the laboratory and in the users' natural work environments. The results, as discussed in Section 6.6, suggest that an iterative process of design and evaluation, if adopted early on in the project's development, can help ensure that the VE will meet the practical needs of its end-users. The evaluation also shows that appropriately chosen multiple views of the scene are useful in urban planning, and that the realism enhancements introduced increase the effectiveness of VEs for urban planning.

## 6.2 Related Work

Virtual reality development for architectural design and urban planning applications can be roughly grouped into two categories: applications that display detailed 3D CAD models of architectural spaces/structures and rapid prototyping systems.

In the first case, the challenge has been to visualize large data sets in as photorealistic a fashion as possible. These environments are mostly used for presentation, recreation, and educational purposes (e.g. review of architecture before it is actually built, cultural heritage reconstructions, 3D entertainment rides, etc.) where complex 3D spaces are constructed so they can be explored in walk-throughs [31, 37]. The majority of these projects allow for little to no interactivity beyond the user's ability to freely navigate about the environment.

On the other hand, the virtual prototyping environments allow immersive VR to be used in earlier phases of a design process and are thus designed to incorporate a higher level of interactivity and object manipulability. In most cases, these capabilities are implemented at the expense of visual realism, as they have been developed by computer scientists in order to further advance research in VR tools. Furthermore, most of these environments support only trivial user tasks and thus cannot be used in real-world situations. Nevertheless, many interesting ideas have been introduced by architectural prototyping projects that we can draw from. The CALVIN project [45], for example, introduced the idea of different perspectives, the mortal (ground-level) viewpoint and the deity (global above-ground) viewpoint, either of which users can assume to interact collaboratively in designing a space in VR.

The core idea from the beginning of our work has been to combine the strengths of the above two categories of virtual environments, namely to achieve the realistic visualization and auralization of a large space coupled with the ability to use it early on in the design process as an interactive work tool. Essentially, our goal is to create a virtual space that can complement and enrich people's day-to-day work practice in a useful and meaningful way. In order to achieve this we adopted the design approach described next.

## 6.3 Design

The basic premise of our VE design approach has been to engage architects, designers and decision makers from the first steps of the design. Putting such a user-centered approach into practice requires collecting and analyzing as much information about our users as possible, through a detailed user requirements process [81] and a deep understanding of how they work.

The city of Nice and the Greater Nice-Cote d'Azur Urban Community (CANCA) recently decided to build a Tramway. The project involves 8 km of rail in the most dense parts of the city, requiring the re-design of several open spaces such as the main city squares, "Place Garibaldi" and "Place Massena". We established a working relationship with the officials and the company of architects in charge of the project. Initially, we presented a simple VE prototype of a section of "Place Massena" to the "Mission Tramway", the organization in charge of the overall project (see images in [80]).

The result of this contact led to a closer collaboration with the architects on the re-design of "Place Garibaldi", and enabled us to gain access to all the project data. The architectural design of this square was of major importance, since the "Place Garibaldi" is a historic landmark and,

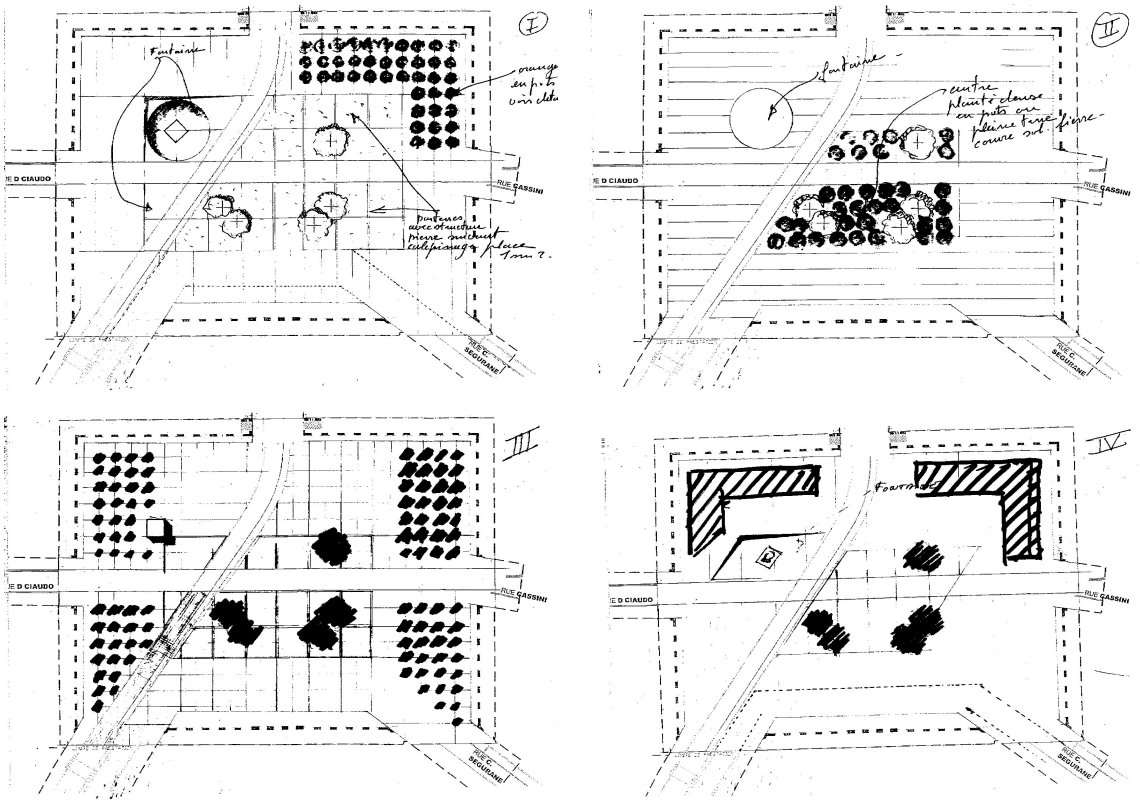



Figure 6.1: Fast architectural sketches of Place Garibaldi.

according to many “the most beautiful square of the city”. As such, many stakeholders participated in the decision making process: local elected officials (principally the mayor), the officials of the city council in charge of open spaces and public works, as well as higher state authorities at a national level who generally have a definitive say in any modification of a historical space. There is also a public consultation which occurred at the beginning and continues throughout the design process.

Our collaboration was founded on the principle of mutual benefit. We were interested in studying and understanding the workflow to allow us to design novel VE tools that combine realism and interactivity, and to apply them in a real-world setting. The architects and decision makers of the Mission Tramway were interested in using the resulting interactive VE as an aid in decision making and brainstorming, as well as a presentation tool.

### 6.3.1 Study of End-User Design Workflow

After multiple discussions with architects, and following them in their work, we made a number of observations on the workflow of the architectural design process in this particular project and on the way that decisions were made. Architects in this project work in a very constrained manner, since the decisions made are ultimately political and several different stakeholders intervene in the process.

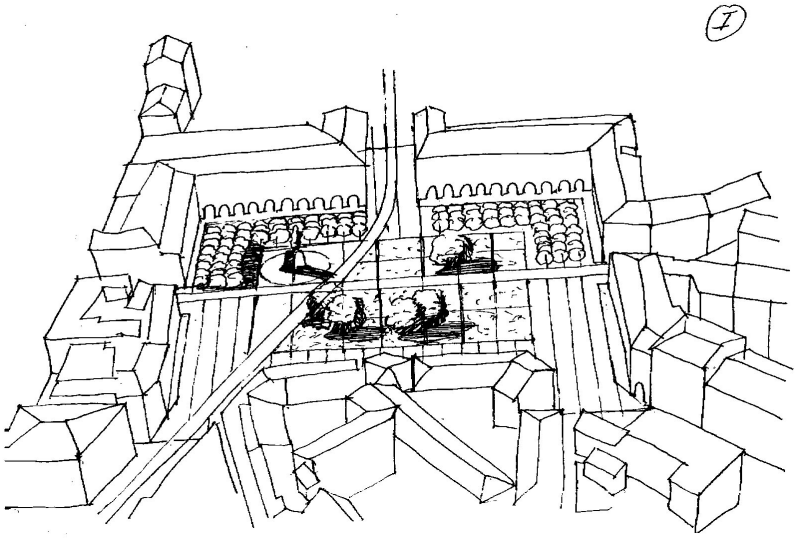


Figure 6.2: Perspectives can be used for an overall view of the design.

For the re-design of the square, several different concepts were proposed, with significant variations: A “Place d’armes” (military, “stone-only” square) in which no additional vegetation was to be allowed; A modern design, with additional vegetation; A more traditional design, with additional vegetation (See Fig. 6.10). The architects had a complex balancing act to follow. According to our interviews, they spent a while debating in numerous meetings the different merits of each choice, without advancing much.

A simulation of the different concepts was shown to the politician charged of the Tramway project during the public event organized by the CREATE project at the INRIA. The 4 screenshots shown in Fig. 6.10 are captures of the virtual environment shown during this meeting. They were very interested in the technology and the applications that it can have in other projects.

The initial design process took place mainly between the municipal administration and the architects. At this stage, architects used simple fast “sketches” to communicate their ideas and

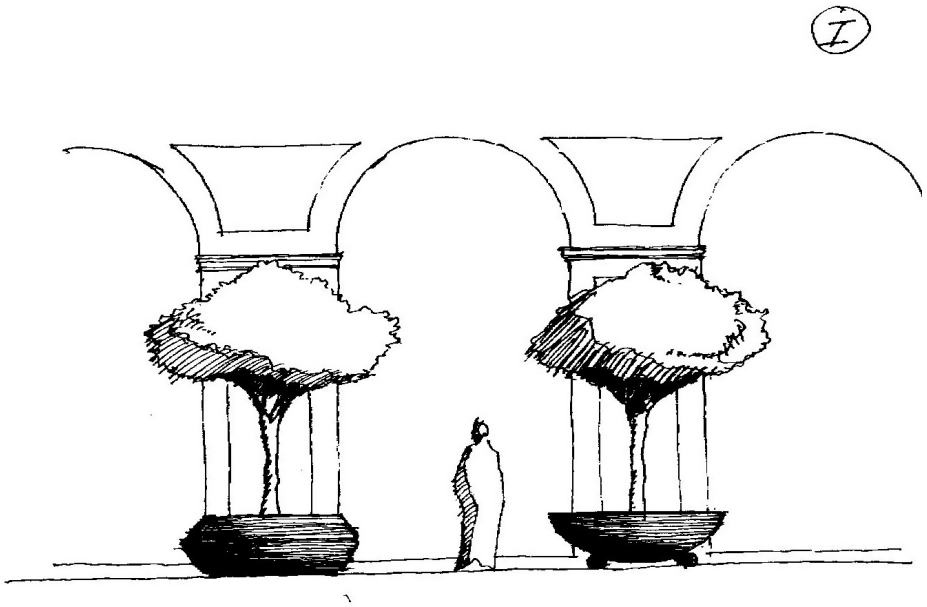


Figure 6.3: More involved perspectives can be used to illustrate specific details in the scene.

the different options. Examples of these “fast” designs are shown in Fig. 6.1.

A 2D plan drawing was used for initial designs at the first stage of conception. As we can see, several different designs were proposed, including a compromise so that the basic “minimal vegetation” requirement of one proposal was being met, while adding some vegetation; however, the proposed vegetation is limited to small trees in pots rather than planted trees.

This is the stage in which all options are truly open, and we can see that architects work with approximate hand-drawn sketches, mostly based on a 2D plan projection, but sometimes using perspective to illustrate an overall concept (Fig. 6.2).

Using this initial stage where they iterate over many experimental sketches on paper, the architects get an overall feel for the design, and sometimes will draw details of part of the site to capture the essence of specific elements (Fig. 6.3).

The next stage is the production of photo-montages (Fig. 6.4) which are shown to the decision makers and the elected officials. These montages are used to achieve agreement by the different parties, and after this process, the design is essentially “fixed”.

Following our observations of the workflow, we decided that we will not address the initial “sketch” phase, since this would be a research topic on its own requiring sketch-based interfaces, an active research area which, however, is not yet mature enough for use with professional users.

Figure 6.4: Detailed montages are used to present the project to decision makers.

### 6.3.2 Design of the VE Interface

The design of our interface was inspired by the workflow described in Section 6.3.1. The initial idea was to preserve the "top view" corresponding to the familiar existing workflow (Fig. 6.5). We also have a ground-level "perspective view" (Fig. 6.6), corresponding to that used in the photo-montage. We have introduced an intermediate "balcony view", shown in Fig. 6.7, where the viewer is presented with a view as if she were standing on the balcony of one of the surrounding buildings in the square. This view corresponds to the higher level sketch perspectives that the architects sometimes use in the initial stages of their design (see Fig. 6.2).

The central functionality of the interface allows the user to manipulate *dynamic* elements in the scene, such as benches, umbrellas or trees in the top view, inspired by the iterative sketches process (Fig. 6.1). In our system the user has the ability to freely switch between top, perspective and balcony view at any time and perform manipulations in all three.

Given our choice of concentrating on the second phase of design, when many of the basic choices have been made, the top view contains the basic layout of trees with their grass-filled container regions, the statue and fountain. Since manipulation occurs mainly in this view, the user is presented with sets of menus on either side of the top view (Fig. 6.5). Menus are only present in the top view.

The left hand side contains the "insert" menus, in this case options for inserting the 3D models of different benches and umbrellas. The right hand side contains the "operations" on the inserted items, such as "resize", "resize width", "move", "rotate", "select" and "apply". Motion and

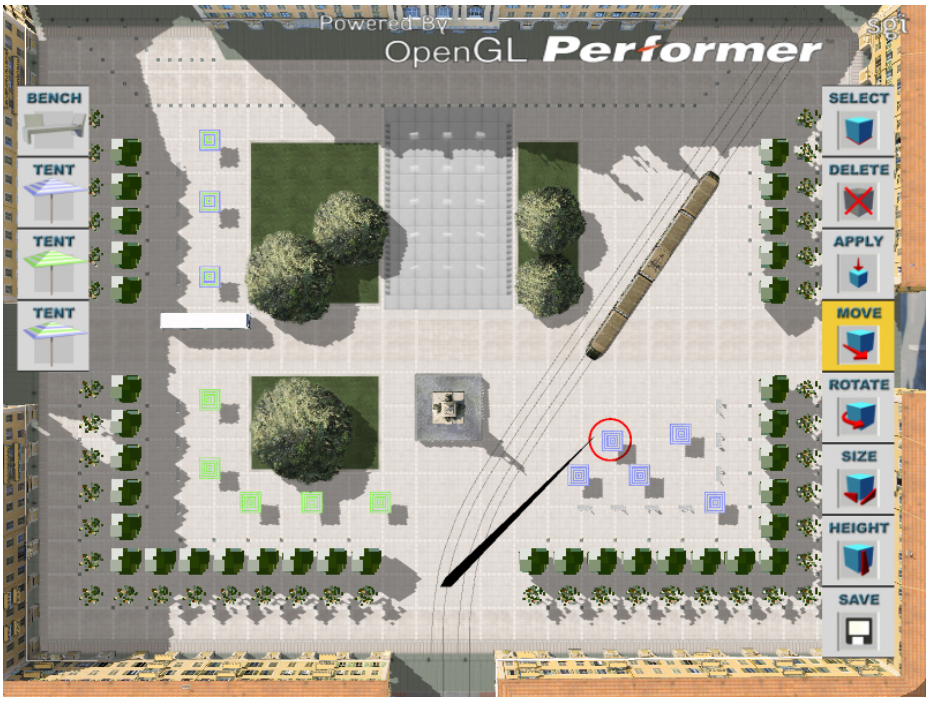



Figure 6.5: The top view of the VE is displayed with two sets of menus for the insertion and manipulation of dynamic objects. A small umbrella (circled in red) is attached to the end of the wand and can be moved and positioned anywhere in the Square.

placement are snapped to a grid in the scene.

Our immersive display interface includes a tracked game-controller, with 12 buttons and a joystick, which operates similarly to a "laser pointer". The 3D model of a rod "extends" the device in the virtual space. Four of the buttons are used. One is the "action" button used for selection of the menu items and for all other selection/deletion/manipulation actions. The remaining three buttons are mapped to the top, perspective and balcony views.

In top view, the user can select an object from the menus using the wand, which places her directly into "insert" mode. When an object icon is selected, a miniature object (umbrella or bench) is attached to the end of the wand and inserted onto the ground of the Square (Fig. 6.5). Each time the user clicks on the action button, an object is placed at the current location. During this time the user can switch to the other views. The user can then select one of the "move" "resize" or "rotate" menus to perform the equivalent action to the chosen object. Manipulation of quantities is performed relative to this fixed point (i.e., distance from the point of intersection of the wand at the time of the first click). Once the user has designed one of the objects to be placed, she can apply the design to a set of objects via the *select* and *apply* menus.


Figure 6.6: Snapshot of the perspective view. Note the realism of the captured facades of the buildings, shadows and point-based rendering of trees.

At any time during the design session the user can move around in the environment and evaluate the result of her work either in perspective view or by moving to the "balcony view", allowing a more "overall" view of the current state of the design.

## 6.4 Evaluation Methodology

Our evaluation methodology draws from the structured framework proposed by [33, 10] for the design and evaluation of user activity in VEs. This includes the combination of user needs analysis, user task scenarios, usability evaluation and formative evaluation, and preliminary summative evaluation.

The user needs analysis was carried out at the very beginning of the project and led to the definition of the user task scenarios that were used in the evaluation sessions. Usability evaluation forms a central tenet of our evaluation methodology, as it involves observing the users of the VE in order to determine if the VE aids or hinders them in reaching their intended goals. The International Organization for Standardization defines Usability (ISO 9241-11:1998) as "the ef-


Figure 6.7: Snapshot of the balcony view of the Place Garibaldi, showing the Tramway passing through the square. The user can move around freely in all views, including the balcony view.

fectiveness, efficiency and satisfaction with which specified users can achieve specified goals in particular environments". Similarly, Nielsen identifies learnability, efficiency, memorability (retention), accuracy (errors), and satisfaction as the most common attributes of usability, as well as issues concerning user interface and interaction [65].

Usability evaluation is usually performed by expert users who follow a set of heuristics for the assessment of a system, such as the ones identified by Nielsen. However, since we adopted a participatory design approach, which means working in real-world application conditions directly with end-users implicated in a real project, the representative users of our environment (the architects of the real-world project) served as both usability experts and typical end-users.

We have chosen to limit our testing to small number of users and follow an in-depth qualitative approach. One of the reasons for this, is the obvious difficulty in evaluations of real-world situations, i.e., getting busy, highly qualified professionals to agree in participating in experimentation, which requires a significant investment in time. Other reasons include the highly experimental nature of the prototypes and the use of innovative and relatively inaccessible equipment (tracked immersive VR displays).

We believe that for this project, case studies, where small groups of users are studied in depth, is more useful for gaining insights into the effectiveness and efficiency of our system, and comes as a natural continuation to the participatory design process that preceded the evaluation. Usability testing was conducted in the controlled setting of our laboratory. The users were asked to carry out a set of predefined tasks. The tasks used for the usability evaluation accurately represented the intended actual use of the application and occurred within a realistic scenario.

The method used in our evaluation included direct observation, a post-experiment questionnaire and post-experiment interviews.

*Direct observation*. Users performed the various tasks whilst being observed by a facilitator. Users were encouraged to use a think-aloud protocol [28] to explain what they are doing, to ask questions and to give information. The facilitator used an interactive style, asking users to expand upon comments and activities. Sessions were also videotaped for further analysis.

*Questionnaire*. A usability questionnaire was developed to identify the user's perception of the effectiveness and efficiency of the system and their level of satisfaction with the interaction. Our questionnaire was constructed by merging a number of standard user satisfaction questionnaires, such as the approaches provided by Perlman [70] and others [15]. The questionnaires include questions that require answers on a 1-7 Likert scale [50].

*Interviews*. An informal interview following the experience was used to help identify the various issues that occured during the experience that could not be captured by the questionnaire. The interview was particularly important for understanding the issues involved in the in situ usage of the system, where the use of a questionnaire does not make sense.

## 6.5 Evaluation

As mentioned previously, we performed evaluation of the VE both in a controlled laboratory context and in the natural work environment of the various individuals involved in the real urban planning project. In the next sections we describe both these forms of evaluation, controlled and situated.

### 6.5.1 Experiment

The goal of the controlled experiments was to evaluate the combined effect of realism and interactivity for a real-world urban planning task, as well as the system's usability. As mentioned

in Chapter 5, the realism enhancements we tested were: high-quality shadows, realistic point-based vegetation, 3D sound and crowds. The way this was done was by including three different levels of realistic features into the same scene, one where visual enhancements and sound where included, one with no enhancements and one with audio/visual enhancements and simulated population. The experiment was run with three of the collaborating professional architects, all directly implicated in the real Nice Tramway project, and specifically in the design of the new Garibaldi square. Prior to the sessions with the architects, we ran pilot studies with engineers who had no previous VR or computer graphics experience.

**Procedure**

The experiment took place on a Barco Baron workbench with the tracked game controller operating as described previously (Fig. 6.8). Each subject was head-tracked and wore active stereo glasses.


Figure 6.8: View of a user of the system on the workbench.

The overall goal, as defined by the users themselves from the beginning of the project, was to “create, define the appropriate size and position of design elements (different umbrellas and benches), and to evaluate the occupation of space and overall aesthetic effect on the new design of

the square".

The main experiment was preceded by a simple training environment in order to allow the user to learn the interface.

**Training Environment**

The training environment contains the same interface as the main experiment, but the VE is a simplified version of the scene. The menus described in Section 6.3.2 (Fig. 6.5) are all present in the training scene, but the scene consists simply of a plane representing the ground. This results in an uncluttered, simple VE, which allows the user to learn the interface more easily. In addition, this VE has no latency problems, since it contains none of the realism VE enhancements we have added.

The training sessions starts with the facilitator explaining the overall goal of the experiment, and then introducing the interface. The facilitator performs all the tasks (insert an umbrella, move, resize and rotate an umbrella, select and apply). The meaning and use of the top, perspective and balcony views are all explained and presented.

The user is then guided through these tasks, and asked to repeat them until she feels comfortable with the interface, typically for around 10-15 minutes.

**Main Experiment**

The main experiment VE included the entire environment of the new design of Place Garibaldi, with the same interface as the training session (Fig. 6.5). The task was to place, size and arrange the umbrellas and stone benches in regions close to the orange trees. The user was asked to first position the elements, then determine the size (height and size) of the umbrellas and the placement of the benches, and finally to evaluate the "presence" and "occupation" of the square.

To evaluate the effect of realism, three levels of realistic VE enhancements were encoded into the same experiment. To achieve this we assigned a different colored umbrella to be placed in each of the three corners of the scene as seen in the top view (see Fig. 6.5, lower right, lower left and top left corners). The users were asked to design and place umbrellas and benches for each one of the corners/color codes in sequence, completing the design for each corner/color color before proceeding to the next. Thus the design for each corner was implicitly a separate sub-task, encoded by umbrella color. Each sub-task was performed with a different level of realism. In particular:

- The task in the lower right corner (blue umbrellas), was displayed with shadows, point-based trees and sound.
- The task in the lower left corner (green umbrellas), used "standard" VR quality (e.g., no shadows, no sound, billboard trees, and no crowds).
- The task in the upper left corner (mixed-color umbrellas), was displayed with all enhancements (shadows, point-based trees, sound and crowds).

Each user was presented with the top-view at the outset, and was told to create and place umbrellas and benches, starting with the lower right corner. Thus the user started with blue umbrellas. The user would insert a primitive (umbrella or bench), and typically position it in top-view. The users were reminded that they could switch freely between views. In a typical session, the user would place a set of umbrellas, either aligned or not, to populate the space. They would then switch to perspective view to correct or adjust placement, and to resize the width and height of the umbrellas. Balcony view was often used to judge the design.

The order in which the levels of realism were tested is important; we chose display enhancements first, no-enhancements and all enhancements with crowds, in that order, to avoid an implicit ranking of quality. It is important to note that the user was not informed of the changes to visual realism elements, and we attempted to identify their relative importance in the questionnaire. Sometimes the users did observe the differences (for example shadows or not etc.), but the facilitator tried to insist that the user concentrate on the task.

### 6.5.2 Field Deployment of the System

In addition to the controlled lab experiments, the system has been used at several different occasions in the context of the real project. This was part of the agreement, in which the authorities provided us access to the architects and the data in exchange for their use of our system.

We visited the authorities and the architectural offices at several occasions and report here only the most significant meetings with the official project working group in charge of the design of outdoor spaces. One of these meetings concerned the discussion of a planned proposal for the choice of the type of trees to be used in the square. The choices included either the 3 meter-high orange trees or the 8-meter high oak (see Fig. 6.9). As the meeting took place at the Nice City Hall, our VE tool was presented on a simulator running on a laptop and using a portable projector with a stereo loud-speaker based sound system and the standard gamepad-based navigation input.


Figure 6.9: Snapshots of the VEs used in the discussion session with the architects. Left, the "Orange tree" solution; right, the "oak tree" solution.

The working group was comprised of 3 high-ranking city officials, in charge of public spaces and urban planning, 3-4 city middle-level managers who were mainly architects concerned with the overall view of the project, 2 officials from the Mission Tramway, and the architect in charge of the overall project.

The functionality of the tool, including all the realism features (image-based captured buildings, shadows, billboard-based vegetation, 3D sound and crowds) and the top, balcony and perspective views were presented, along with the other system capabilities such as the ability to freely move around in the virtual square with the Tramway, bus and crowds. The two scenarios (orange trees and oak trees) were mapped onto two different buttons on the handheld device. Once the demonstration of the tool was completed, the members of the working group took over, exploring the different views and locations provided for the virtual square and making different choices. They chose one or another type of tree in different places in the square in order to test spatial relationships, the placement of objects, but also the different effect of shadows etc.

Another example of field deployment involved the use of the full VE system (tracked stereo-vision workbench) for a brainstorming session. Two of the main architects and the designer of the project participated in this session and used the system as an opportunity to discuss issues concerning the design of the square, most notably the choice of trees and ground elements.

Following sketches of the architects for the different possible scenarios of the square we

Figure 6.10: Screenshot of the 4 scenarios used during the presentation to the politicians.

create a virtual environment showing them. The four scenarios where, sorted like in Fig. 6.10 a "Place d'armes" (military, "stone-only" square) where no trees and no cars are present, a "stone-only" square with cars around the square, a pedestrian square with trees and no cars and a pedestrian square with trees and cars around it.

During the public event organized at INRIA by the CREATE project, the politicians in charge of the project were present, and notably, the politician in charge of all transportation issues in the municipality as was the chief engineer of the project. During this public event we presented the four different scenarios, the politician immediately declared that things were much clearer and that he now understood the implications of placing trees or a road around the square.

## 6.6 Evaluation Results and Discussion

To evaluate the laboratory experiment, we used direct observation, a questionnaire and interviews. In the in situ evaluation, the use of a questionnaire is inappropriate, thus limiting evalu-

ation to observation and interviews.

### 6.6.1 Observations from the laboratory experiments

After completing the experiment, each of the three architects was asked to complete the questionnaire and participated in the post-experiment interview that followed. We studied the video-taped sessions in order to better reflect on the important issues and difficulties that were expressed by the users during their interaction with the system.

We classify our observations by learnability and ease of use, effectiveness and efficiency, user satisfaction, VE/interfaces features and realism.

In terms of learnability and ease of use, all participants ranked the system as easy to learn (6 or 7 on the Likert scale) and stated that they were able to use the tool without difficulty. We were particularly pleased with this result, since 2 out of the 3 subjects have no experience with interactive 3D systems or video games.

In terms of effectiveness and effciency, there was a uniform approval (6 or 7 on the Likert scale) of the utility of the tool and the fact that the system would improve productivity in the workplace. The top view, although familiar, was judged moderately useful (3-5 on the Likert scale). From observing the videos and the interviews it became clear that the precision was insufficient. In retrospect this was to be expected, since the distance from the object being manipulated is too large. Clearly, the best solution to this is a mixed 2D-3D interface, where a "pen-like" interface could be used to directly place objects onto the top-view, as is currently done in existing CAD tools. The need to have the same interface as CAD tools for these tasks was explicitly mentioned by one user. Subject 2 clearly stated that the perception of ambience and scale were extremely useful and important for an architect in the evaluation of an urban planning project, and found that the tool had great potential for brainstorming and interactively trying out different alternatives.

In terms of satisfaction, all users stated both in the questionnaire and in the interviews that they liked the tool. The only other observation (only made by Subject 1) was that given a choice between latency and realism it is clearly preferable to have low-latency and less-realistic display. In the case of the first subject, the parameters for point-based rendering were incorrectly set, and this made manipulation in the trees difficult. This was the reason for this comment. For Subjects 2 and 3 this problem was corrected. Subject 1 also stated that one of the main values of the system was that it removes the "break" which exists between traditional 3D CAD systems and the resulting design.

In terms of VE interface features, the balcony view was used extensively and was greatly

appreciated by all three participants.

All users agreed that this was a particularly useful view of the environment, and that it helped in their judgement of the resulting design, but also during the design. In the questionnaire and the interview Subject 1 pointed out that it would be nice to have the menus present even in the other views (perspective and balcony); the reason this had not been implemented was simply a restriction in programming complexity.

We made several important observations concerning realism. The realism offerred by image-based facade textures and point-based vegetation were identified as being important, allowing users to better understand the final effect of the placement of trees on the overall design. The ability to have the true high-quality 3D leaves when zooming in was singled out as being important. However, when explicitly asked in the questionnaire about the differences in realism between the different colored tent sessions, the differences between billboards and point-based trees or the presence/absence of shadows did not show up in the questionnaire ranking. However, in the interviews the users responded that they were concentrating on the task and when questioned further, and all identified the importance of shadows, for placement and sense of scale, but also in terms of appreciation of shadow/sun coverage in the square.

The presence of human figures was judged central, in particular as a marker of scale. As mentioned before, human figures are used in traditional drawings in this manner (Fig. 6.3). However, the specific representation used, which is based on low-resolution billboards, was judged insufficient.

The inclusion of spatialized 3D sound was judged as very important to evaluate the overall ambience and atmosphere created by a certain design, in the presence of fountains, the tramway and buses. This was the one item of realism whose presence/absence was noticed by 2 out of 3 participants.

### 6.6.2 Observations from field deployment

In the meeting at the Nice City Hall the dynamics of the working group session were very interesting. The group was large (more than ten people counting ourselves), and thus it was difficult to follow the power relationships which were occurring. In the beginning of the presentation, people expressed themselves quite freely and several interesting questions became apparent. Later in the meeting the hierarchical relationships among the participants became more evident, and those who report directly to the elected officials tended to dominate the discussion.

The first interesting observation was that several people at the meeting considered the representation to be *too realistic* and thus judged it to be inappropriate for public display. One reason advanced for this observation was that people would believe that this would be the exact design thus removing all freedom for the realisation of the final project; with drawings people are more aware of the level of abstraction of the design. Nonetheless, the group considered that this was an excellent tool to help with decision-making, and that with appropriate preparation could be used when showing the design to the elected officials. Their main concern was that all elements (e.g., colors used for the stone of the square etc.) be fined tuned within the VE, to avoid an incorrect detail that would discredit the entire presentation.

The fact that the system could interactively switch between the different types of trees, allowing the evaluation of shadow coverage on the square and the relative height of the trees was greatly appreciated. Several participants remarked that they have never until this moment understood the difference made by the different types of trees, nor the difference in shade provided by the tall trees.

The balcony view was also appreciated, since it gave a completely novel perspective of the overall design which was not previously appreciated by the participants who had, nonetheless, been working on the project for more than six months.

As far as the brainstorming session is concerned, the two architects and the designer worked mainly on the choice of trees, as was the case with the large working group. However, in contrast, these experts worked closely on detailed aspects of the design, for example the choice of ground elements around the trees, the material used to represent these elements (type of wood/stone etc.), or the spacing between the trees and the choice of their number (2 or 3 rows etc.)

The tool, as evidenced by their discussions, clearly helped the architects to understand the space and ambience created by their design in a manner which was not previously possible. They expressed a high degree of satisfaction at the end of the brainstorming session.

## 6.7 Conclusion

The above are preliminary results derived from an experiment with an unavoidably small user set. However, we argue that they are rich results because they involve an in-depth observation of a real work process, a VE design with the involvement of the actual non-IT expert users, and the in situ use in the decision making process of a real project.

The main conclusion of our work is that by using a participatory design approach and a

focused evaluation process, we were able to tailor the development of the VE to the real needs of our end-users and increase the validity of our environment as a practical yet impressive work tool.

Some problems remain, for example the interface issues noted by the users (presence of menus in all views, precision in the top-view etc.). None of these however are particularly hard to resolve, mainly requiring further software development. Other remaining issues are related to the practical difficulties of VE development for real applications cited in the introduction.

In terms of more specific conclusions, we observed that the use of multiple views (and in particular the balcony view), was greatly appreciated and judged very useful by the users. This shows the power and flexibility of realistic VE environments in a real design process. It is important to note that the successful design of these views was based on our observation (e.g., sketch in Fig. 6.2) and understanding of how the users actually work, and their continuous input throughout the design-development-evaluation cycles.

We were also pleased to see that our evaluation process indicates that all our realism enhancements to VEs were judged important. Realistic vegetation, shadows and spatialized 3D sound were cited as being important in judging the quality of an outdoor design, and made the system more useful. The presence of people in the scene was seen as having central importance as a marker of scale and of giving a feeling of life to the scene. Again, the inclusion of these elements came in part from our participatory design approach.

Although this approach of engaging users in the design is time and resource intensive, we consider it to be worthwhile and will continue to explore the application of HCI models into the design of VEs. We believe that this approach provides promise for the development of environments and meaningful tools that can be of real value to their users.

# Chapter 7

# Conclusion

The thesis is divided into two complementary parts. In the first part we presented two new techniques for Image-Based Modeling and Rendering, one for relatively simple objects and another for trees. As mentioned in the introduction, this thesis was funded by a CIFRE scholarship, which encourages focus on real world problems. We saw this an a opportunity to develop novel research solutions for real-world problems. In particular, the two above-mentioned techniques have been inspired by the work and the problems found during the development of the real world project of the Nice Tramway, in the context of the CREATE EU IST project. In the second part we have presented the applicative part of the thesis, that is, the more systems- and development-oriented work related to the creation of a usable virtual environment and it's evaluation.

The advantage of work in a real world project is that we were confronted with real problems, which are more complex and challenging compared to typical "toy examples" often used to illustrate novel research ideas. Our research has been directed and inspired by this problems, but they have a common point, Image-Based Modeling and Rendering that was our main task during the realization of the CREATE project.

One of the problems found at the beginning of the project was in the texturing of the models reconstructed from the images using the original input photographs. The texture extraction appeared to be very memory- and time- consuming due to the time needed by the artists to erase all the artifacts in the images. This led us to find an alternative solution, View-Dependent Layered Texture Maps. This new approach solves some of the problems found in the work of Debevec [17], that was the most appropriate previous technique for our problem.

This new approach solves the geometric subdivision problem and the hole filing problem that was done automatically without giving the chance to the user to modify the result and which

resulted in blurred, low quality textures in the interpolated regions. Occlusions are the main problem since, in the VDTM approach occlusion is solved subdividing the geometry and filling missing textures with automatic interpolation. In our approach, we solve the problem at the image level, and not at the geometry level as before, dividing the image into layers corresponding to the different levels of occlusion. In this way the geometry is not modified and the holes produced by occlusion can be edited in a standard image-editing tool. Since the result of the algorithm is an standard image with layers, the algorithm can easily be integrated into any Image-Based Modeling application and directly interfaced with an Image Editing tool. Also, since we know the depth information, this can be integrated into an Image-Editing program to be able to use techniques like that presented by Oh et al. [66].

However, some problems still remain. The geometry needs to be as precise as possible and currently this step still involves a lot of manual "clicking" even when using a modelling-by-images system such as RealViz or Facade. Another task that needs to be done by hand is hole filling. This new approach gives complete control to the artist to add or erase information, but we lack of methods to help fill the holes produced by occlusions. Although there are some solutions to this problem, giving to the artist the chance of filling the hole automatically and to accept or to refuse the result. Missing information in one image can be recovered from other view points giving the chance to the artist to use this information to fill the holes. Since we know the geometry, we can copy paste parts of an image to another one applying the needed deformation to fit exactly where they correspond. An algorithm of Inpainting [7, 25] can be used filing holes automatically, but always leaving the control to the artist to correct filled regions if necessary. An implementation of the work of Drori et al. [25] can be very powerful, if adapted to be guided with information coming from other views, for example.

Another problem encountered during the work for CREATE, was the capture and rendering of trees. In the context of urban planning projects, it is necessary to capture the existing tree, rather than using generic synthetic models. Maintaining the shape and the appearance of the real tree is thus very important. We developed a technique inspired by medical imaging tomography to reconstruct a volume of opacities, enhanced with billboard textures, that reproduce exactly the captured tree when viewed from the same view-points where the pictures where taken and create a plausible interpolation between them. The resulting billboard-enhanced volume can be used in a real-time application.

Compared to existing techniques to model trees, our method is able to create a volumetric representation of an existing tree and our result is immediately adapted to real-time rendering. The

time needed to capture a tree is relatively fast, compared to detailed manual modeling in a standard package such as Maya or 3DSMax. Our capture technique could also be used in other contexts, for example Tree Physiology, where capture often involves manual clicking of hundreds or even thousands of points, using a magnetic sense.

One of the problems of our new method is the lack of semantic information. Since we reconstruct a volume, we do not differentiate between trunk, branches and leaves. This can lead to a very interesting future work; using this semantic information could allow us to analyze the structure of the tree and generate new instances of the same kind of tree with a different configuration of branches and leaves. This can help to accelerate rendering, replacing the known parts by geometry, creating for example and hybrid representation where trunk and branches are geometry and leaves are volumes enhanced with billboard textures as in the original approach.

Another problem, is the heavy use of the texture memory needed to represent a tree. However we think that the textures used are easily compressible using PCA techniques or simply generating random textures in the parts of the tree where the noise due to high frequencies can make this randomness imperceptible. When demonstrating this work to a games company, there was great interest in the quality of the trees, but the memory overhead was considered order(s) of magnitude too high.

The real-time rendering can be optimized using the hierarchical structure used in the approach. Once we have the billboard textures of the lowest level of the hierarchy we can relatively easily generate billboard textures for all the intermediate levels of the structure, generating in this way a multi-level representation of the tree, and a level-of-detail algorithm for rendering can thus be easily developed.

We have been contacted by biologists that are already interested by the method to be able to quickly capture an existing tree and take measurements of it. Similar methods exists in the Tree Physiology research field, like the one presented by Phattaralerphong et al. [71], and we think our approach can be used in this way too, with results of much higher quality in this context.

Last but not least, we have presented the creation and evaluation of an Image-Based Virtual Environment used in real world project. This is the part of the thesis that is more application oriented. We expect that the design and system choices made here will be of interest to other real world projects of the same kind.

Due to the constraints of timing and man-power for CREATE, the two techniques developed in the first part of the thesis were not actually integrated in the working prototype used for the evaluation of the virtual environment with the architects. Nonetheless, we integrated two advanced

techniques (view-dependent textures and sky-dome lighting), in this prototype as an exercise of applying research into a real-world virtual environment application. The methodology developed in this context however, should permit relatively straightforward integration of our novel techniques in the future.

Looking back on the development of the VE system, we realize that a major limitation in our work were the restrictions imposed by the use of older hardware for the large installations, i.e., the CAVE's and RealityCenters of the project. These systems use Performer under IRIX to optimize for multipipe projection and multiple displays. Given the evolution of graphics hardware and systems, using a modern graphics API with lower overall overhead, and is better adapted to the new features of modern GPUs, would greatly improve the quality of the VE's which we could offer for the application context. Even though we believe that most of our design choices (portability, abstraction, extensibility via scripting, etc.) are still valid in such a novel context, some choices would probably have been different.

The evaluation was done with a small number of users, nonetheless we consider that we have rich results because of the in-depth observation done of a real work process. The system was used in the decision making for the real project, with very positive feedback. Our conclusion is that by using a participatory design approach and a focused evaluation process, we can develop a VE to solve real needs of our end-users.

The users of the system found some problems, such as the interface where menus where present only in the top view or the precision when placing objects. But these problems can be fixed with additional software development. We observe that multiple views are really appreciated by the users, especially the balcony view, and they were judged as very useful. This was something expected because the views where inspired by sketches done by this same users, and also shows the flexibility of VE compared to the sketch design. The users judged all the realism enhancements as very important. Vegetation, shadows and sound were cited as being important in judging the quality of outdoors design. The crowds were also seen as having importance as a marker of scale and giving life to the VE.

We believe that this approach provides promise for the development of environments and meaningful tools that can be of real value to their users.

In conclusion, we presented in this thesis two novel contributions in Image-Based Modeling and Rendering, which were inspired by problems encountered in the context of a real-world project. We also presented the design development of a complete realistic VE system for use in urban planning, and a first attempt at its evaluation. The fact that the work was conducting in

the context of a real world project is a big constraint, but at the same time a very rich source of inspiration.

# Bibliography

[1] Foundation of the Hellenic World, September 1993. http://www.fhw.gr/fhw.

[2] E. H. Adelson and J. R. Bergen. The plenoptic function and the elements of early vision. *M. Landy and J. A. Movshon, (eds) Computational Models of Visual Processing*, 1991.

[3] Adobe. Adobe Photoshop. www.adobe.com/photoshop.

[4] Adobe. Canoma. www.canoma.com.

[5] Alias. Maya. www.alias.com.

[6] A. Andersen and A. Kak. Simultaneous algebraic reconstruction technique (sart): A superior implementation of the art algorithm. *Ultrasonic Imaging*, 6(81), 1984.

[7] Marcelo Bertalmio, Guillermo Sapiro, Vincent Caselles, and Coloma Ballester. Image inpainting. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 417–424. ACM Press/Addison-Wesley Publishing Co., 2000.

[8] J. F. Blinn. Light reflection functions for simulation of clouds and dusty surfaces. *Proc. SIGGRAPH'82*, pages 21–29, 1982.

[9] Jeremy S. De Bonet and Paul A. Viola. Roxels: Responsibility weighted 3d volume reconstruction. In *Proc. ICCV-99*, pages 418–425, 1999.

[10] Doug Bowman, Joseph L. Gabbard, and Deborah Hix. A survey of usability evaluation in virtual environments: Classification and comparison of methods. *Presence: Teleoperators and Virtual Environments*, 11(4):404–424, 2002.

[11] Chris Buehler, Michael Bosse, Leonard McMillan, Steven J. Gotler, and Michael F. Cohen. Unstructured lumigraph rendering. In *Proc. SIGGRAPH 2001*, pages 425–432, 2001.

[12] Bardford Chamberlain, Tony DeRose, Dani Lischinski, David Salesin, and John Snyder. Faster rendering of complex environments using a spatial hierarchy. In *Proc. Graphics Interface'96*, 1996.

[13] Shenchang Eric Chen. Quicktime vr: an image-based approach to virtual environment navigation. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 29–38. ACM Press, 1995.

[14] Yung-Yu Chuang, Brian Curless, David H. Salesin, and Richard Szeliski. A bayesian approach to digital matting. In *Proc. of IEEE CVPR 2001*, volume 2, pages 264–271, December 2001.

[15] Fred D. Davis. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly*, 13(3):318–340, 1989.

[16] P. de Reffye, C. Edelin, J. Franson, M. Jaeger, and C. Puech. Plant models faithful to botanical structure and development. In *Proc. SIGGRAPH 88*, pages 151–158, 1988.

[17] Paul Debevec, Yizhou Yu, and George Boshokov. Efficient view-dependent image-based rendering with projective texture-mapping. Technical Report CSD-98-1003, 20, 1998.

[18] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. *Proc. SIGGRAPH'96*, pages 11–20, 1996.

[19] Oliver Deussen, Carsten Colditz, Marc Stamminger, and George Drettakis. Interactive visualization of complex plant ecosystems. In *Proc. IEEE Visualization 2002*, October 2002.

[20] Oliver Deussen, Pat Hanrahan, Bernd Lintermann, Radomir Mech, Matt Pharr, and Przemyslaw Prusinkiewicz. Realistic modeling and rendering of plant ecosystems. *Proc. SIGGRAPH'98*, pages 275–286, 1998.

[21] Discreet. 3ds max. www.discreet.com/3dsmax.

[22] Robert A. Drebin, Loren Carpenter, and Pat Hanrahan. Volume rendering. In *Proc. SIGGRAPH'88*, pages 65–74, aug 1988.

[23] George Drettakis, Maria Roussou, Manuel Asselot, Alex Reche, Alexandre Olivier, Nicolas Tsingos, and Franco Tecchia. Participatory design and evaluation of a real-world virtual environment for architecture and urban planning. Technical Report RR-5479, INRIA, 2005.

[24] George Drettakis, Maria Roussou, Nicolas Tsingos, Alex Reche, and Emmanuel Gallo. Image-based techniques for the creation and display of photorealistic interactive virtual environments. In Sabine Coquillart and Martin Goebel, editors, *Eurographics Symposium on Virtual Environments*, page to appear, Grenoble, France, 2004. ACM.

[25] Iddo Drori, Daniel Cohen-Or, and Hezy Yeshurun. Fragment-based image completion. *ACM Trans. Graph.*, 22(3):303–312, 2003.

[26] Florent Duguet and George Drettakis. Flexible point-based rendering on mobile devices. Technical Report RR-4833, INRIA, REVES/INRIA Sophia-Antipolis, May 2003.

[27] Alexei A. Efros and Thomas K. Leung. Texture synthesis by non-parametric sampling. In *Proc. IEEE ICCV 1999*, pages 1033–1038, Corfu, Greece, September 1999.

[28] K. Anders Ericsson and Herbert A. Simon. *Protocol Analysis: Verbal Reports as Data*. MIT Press, Cambridge, MA., 1985.

[29] O. D. Faugeras. *Three-Dimensional Computer Vision: A Geometric Viewpoint*. The MIT Press, Cambridge, Mass., 1993.

[30] Olivier Faugeras et al. 3-d reconstruction of urban scenes from image sequences. *cvgip-iu*, 1997.

[31] Jr. Frederic P. Brooks. Walkthrough: A dynamic graphics system for simulating virtual buildings. In F. Crow and S. M. Pizer, editors, *Workshop on Interactive 3D Graphics*, pages 9–21, 1986.

[32] T. Funkhouser, J-M. Jot, and N. Tsingos. Survey of methods for modeling sound propagation in interactive virtual environment systems. *accepted for publication in Presence.*, 2003.

[33] Joseph L. Gabbard, Deborah Hix, and J. Edward SwanII. User-centered design and evaluation of virtual environments. *IEEE Computer Graphics and Applications*, pages 51–59, 1999.

[34] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. *Computer Graphics*, 30(Annual Conference Series):43–54, 1996.

[35] Jens Herder. Optimization of sound spatialization resource management through clustering. *The Journal of Three Dimensional Images, 3D-Forum Society*, 13(3):59–65, September 1999.

[36] Dorit S. Hochbaum and David B. Schmoys. A best possible heuristic for the *k*-center problem. *Mathematics of Operations Research*, 10(2):180–184, May 1985.

[37] Mike Houston, Chris Niederauer, Manesh Agrawala, and Greg Humphreys. Visualizing dynamic architectural environments. *Communications of the ACM*, 47(8):55–59, August 2004.

[38] Henrik Wann Jensen. *Realistic image synthesis using photon mapping*. A. K. Peters, Ltd., Natick, MA, USA, 2001.

[39] Tadamura Katsumi, Kaneda Kazufumi, Nakamae Eihachiro, Kato Fujiwa, and Noguchi Takao. A display method of trees by using photo images. *Journal of Information Processing*, 15(4), 1992.

[40] Martin Kemp. *Leonardo on Painting*. Yale University Press, August 2001.

[41] S. Krishnan et al. A hardware-assisted visibility-ordering algorithm with applications to volume rendering. In *Data Visualization 2001*, pages 233–242, 2001.

[42] K. Kutulakos and S. Seitz. A theory of shape by space carving. In *Proc. ICCV-99*, volume I, pages 307–314, Los Alamitos, CA, Sep 20–27 1999.

[43] P. Larsson, D. Västfjäll, and M. Kleiner. Better presence and performance in virtual environments by improved binaural sound rendering. *proceedings of the AES 22nd Intl. Conf. on virtual, synthetic and entertainment audio, Espoo, Finland*, pages 31–38, June 2002.

[44] S. Laveau and O. D. Faugeras. 3-d scene representation as a collection of images. In *Twelfth International Conference on Pattern Recognition (ICPR'94)*, volume A, pages 689–691. IEEE Computer Society Press, October 1994.

[45] Jason Leigh, Andrew E. Johnson, Christina A. Vasilakis, and Thomas A. DeFanti. Multi-perspective collaborative design in persistent networked virtual environments. In *VRAIS*. IEEE, 1996.

[46] M. Levoy and T. Whitted. The use of points as a display primitive. In *CS Department, University of North Carolina at Chapel Hill*, TR 85-022, January 1985. http://www-graphics.stanford.edu/papers/points/.

[47] Marc Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, may 1988.

[48] Marc Levoy and Pat Hanrahan. Light field rendering. *Computer Graphics*, 30(Annual Conference Series):31–42, 1996.

[49] A. Likas, N. Vlassis, and J.J. Verbeek. The global k-means clustering algorithm. *Pattern Recognition*, 36(2):451–461, 2003.

[50] R. Likert. *The human organization: Its management and value*. McGraw-Hill, NY, 1967.

[51] Dani Lischinski and Ari Rappoport. Image-based rendering for non-diffuse synthetic scenes. In George Drettakis and Nelson L. Max, editors, *Rendering Techniques*, pages 301–314. Springer, 1998.

[52] Céline Loscos et al. The create project: Mixed reality for design, education, and cultural heritage with a constructivist approach. In *Proc. of ISMAR 2003 (Poster)*, 2003.

[53] Wendy E. Mackay and Anne-Laure Fayard. Hci, natural science and design: A framework for triangulation across disciplines. In *Designing Interactive Systems*, pages 223–234, Amsterdam, 1997.

[54] William R. Mark, Leonard McMillan, and Gary Bishop. Post-rendering 3d warping. In *SI3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 7–ff. ACM Press, 1997.

[55] Wojciech Matusik, Hanspeter Pfister, Addy Ngan, Paul Beardsley, Remo Ziegler, and Leonard McMillan. Image-based 3d photography using opacity hulls. *ACM Trans. on Graphics (Proc. SIGGRAPH 2002)*, 21(3):427–437, jul 2002.

[56] Nelson Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, jun 1995.

[57] Nelson Max. Hierarchical rendering of trees from precomputed multi-layer z-buffers. In *Proc. 7th EG Workshop on Rendering*, 1996.

[58] Nelson Max and Keiichi Ohsaki. Rendering trees from precomputed z-buffer views. In *Proc. 6th EG Workshop on Rendering*, 1995.

[59] Leonard McMillan and Gary Bishop. Plenoptic modeling: an image-based rendering system. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 39–46. ACM Press, 1995.

[60] Alexandre Meyer and Fabrice Neyret. Interactive volumetric textures. In *Proc. 9th EG Rendering Workshop 1998*, jul 1998.

[61] Alexandre Meyer, Fabrice Neyret, and Pierre Poulin. Interactive rendering of trees with shading and shadows. In *Proc. 12th EG Workshop on Rendering, 2001*, Jul 2001.

[62] Helen Neale, Sue Cobb, and John R. Wilson. A front-ended approach to the user-centred design of ves. In *IEEE Virtual Reality 2002*, pages 199–206, Orlando, FL, 2002. IEEE Computer Society.

[63] M. E. Newell et al. A solution to the hidden surface problem. In *Proc. of the ACM Nat. Conf.*, pages 443–450, 1972.

[64] Fabrice Neyret. Modeling animating and rendering complex scenes using volumetric textures. *IEEE Trans. on Visualization and Computer Graphics*, 4(1):55–70, Jan – Mar 1998.

[65] Jakob Nielsen. *Usability Engineering*. Academic Press, Boston, 1993.

[66] Byong Mok Oh, Max Chen, Julie Dorsey, and Frédo Durand. Image-based modeling and photo editing. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 433–442. ACM Press, 2001.

[67] Manuel M. Oliveira et al. Relief texture mapping. In *Proc. of SIGGRAPH 2000 (New Orleans, La)*, July 2000.

[68] Dave Pape, Tomoko Imai, Josephine Anstey, Maria Roussou, and Tom DeFanti. Xp: An authoring system for immersive art exhibitions. In *4th International Conference on Virtual Systems and Multimedia (VSMM)*. Gifu, Japan, 1998.

[69] Sergey Parilov and Wolfgang Stuerzlinger. Layered relief textures. *Journal of WSCG*, feb 2002.

[70] Gary Perlman. Web-based user interface evaluation with questionnaires.

[71] Jessada Phattaralerphong and Hervé Sinoquet. A method for 3d reconstruction of tree canopy volume from photographs: assessment from 3d digitised plants. In C. Godin et al., editor, *Proceedings of the Functional-Structural Plant Models*, pages 36–39. UMR Cirad-Cnrs-Ephe-Inra-Inria-Ird-Université de Montpellier II, June 2004.

[72] Thomas Porter and Tom Duff. Compositing digital images. In *Proc. of SIGGRAPH'84*, pages 253–259, 1984.

[73] Pierre Poulin et al. Interactively modeling with photogrammetry. In *Eurographics Rendering Workshop 1998*, pages 93–104, June 1998.

[74] A. J. Preetham, Peter Shirley, and Brian E. Smits. A practical analytic model for daylight. In Alyn Rockwood, editor, *Siggraph 1999, Computer Grahics Procedings*, pages 91–100, Los Angeles, 1999. Addison Wesley Longman.

[75] P. Prusinkiewicz and A. Lindenmayer. The algorithmic beauty of plants. *Springer, New York*, 1990.

[76] Kari Pulli, Michael Cohen, Tom Duchamp, Hugues Hoppe, Linda Shapiro, and Werner Stuetzle. View-based rendering: Visualizing real objects from scanned range and color data. In Julie Dorsey and Phillipp Slusallek, editors, *Rendering Techniques '97 (Proceedings of the Eighth Eurographics Workshop on Rendering)*, pages 23–34, New York, NY, 1997. Springer Wien.

[77] X. Qin, E. Nakamae, K. Tadamura, and Y. Nagai. Fast photo-realistic rendering of trees in daylight. In *Proc. of Eurographics 03*, pages 243–252, Sep 1–6 2003.

[78] RealViz. ImageModeler and Stitcher, 2001. http://www.realviz.com/.

[79] Alex Reche, Ignacio Martin, and George Drettakis. Volumetric reconstruction and interactive rendering of trees from photographs. *ACM Transactions on Graphics (SIGGRAPH Conference Proceedings)*, 23(3), July 2004.

[80] Maria Roussou, George Drettakis, Nicolas Tsingos, Alex Reche, and Emmanuel Gallo. A user-centered approach on combining realism and interactivity in virtual environments. In *IEEE VR 2004*, pages 251–252, Chicago, IL, 2004. IEEE.

[81] Maria Roussou, Athanasios Sideris, Céline Loscos, Andrea Dettori, George Drettakis, Jean-Christophe Lombardo, Florent Coudret, Cristian Bianchi, and Franco Tecchia. Requirements analysis on cultural heritage - education and urban - architectural planning and design case studies. Technical Report RN/04/09, University College London, 2004.

[82] S. Rusinkiewicz and M. Levoy. QSplat: A multiresolution point rendering system for large meshes. In *Proc. ACM SIGGRAPH*, pages 343–352, 2000.

[83] M. Ruzon and C. Tomasi. Alpha estimation in natural images. In *Proc. IEEE CVPR'2000*, pages 18–25, Los Alamitos, June 13–15 2000.

[84] Hartmut Schirmacher, Ming Li, and Hans-Peter Seidel. On-the-fly processing of generalized Lumigraphs. In *Proc. Eurographics 2001*, volume 20 of *Computer Graphics Forum*, pages C165–C173;C543. Eurographics Association, Blackwell, 2001.

[85] S. Seitz and C. Dyer. Photorealistic scene reconstruction by voxel coloring. In *Proc. IEEE CVPR 1997*, pages 1067–1073, 1997.

[86] SGI. OpenGL Performer. www.sgi.com/software/performer.

[87] Jonathan Shade, Steven Gortler, Li wei He, and Richard Szeliski. Layered depth images. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 231–242. ACM Press, 1998.

[88] Peter Shirley and Allan Tuchman. A polygonal approximation to direct scalar volume rendering. *Computer Graphics*, 24(5):63–70, nov 1990.

[89] Ilya Shlyakhter, Max Rozenoer, Julie Dorsey, and Seth Teller. Reconstructing 3d tree models from instrumented photographs. *IEEE Computer Graphics and Applications*, 21(3):53–61, May / Jun 2001.

[90] Francois X. Sillion and Claude Puech. *Radiosity and Global Illumination*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1994.

[91] John Snyder and Jed Lengyel. Visibility sorting and compositing without splitting for image layer decompositions. In *SIGGRAPH 98 Conference Proc.*, 1998.

[92] Henry Sowizral, Ian G. Angus, Steven Bryson, Stefan Haas, Mark R. Mine, and Randy Pausch. Performing work within virtual environments (panel session). In *SIGGRAPH '95:*

*Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 497–498. ACM Press, 1995.

[93] Marc Stamminger and George Drettakis. Interactive sampling and rendering for complex and procedural geometry. In *Rendering Techniques 2001*, 12th EG workshop on Rendering, 2001.

[94] Marc Stamminger and George Drettakis. Perspective shadow maps. In John Hughes, editor, *Proceedings of ACM SIGGRAPH 2002*, Annual Conference Series. ACM Press/ ACM SIGGRAPH, July 2002.

[95] Eos Systems. PhotoModeler. www.photomodeler.com.

[96] Richard Szeliski and Polina Golland. Stereo matching with transparency and matting. In *Proc. ICCV-98*, pages 517–526, jan 1998.

[97] Camillo J. Taylor and David J. Kriegman. Structure and motion from line segments in multiple images. *IEEE Trans. on Pat. Analysis and Mach. Intelligence*, 17(11):1021–1032, 1995.

[98] The GIMP Team. The Gimp. www.gimp.org.

[99] Nicolas Tsingos, Emmanuel Gallo, and George Drettakis. Perceptual audio rendering of complex virtual environments. In *Proc. ACM SIGGRAPH 2004 (Special Issue of ACM Transactions on Graphics)*, August 2004.

[100] VRCO. CAVELib. www.vrco.com/CAVELib.

[101] M. Wand, M. Fischer, I. Peter, F. Meyer auf der Heide, and W. Starsser. The randomized z-buffer algorithm: Interactive rendering of highly complex scenes. In *Proc. ACM SIGGRAPH*, 2001.

[102] Lee Westover. Footprint evaluation for volume rendering. *Proc. SIGGRAPH'90*, pages 367–376, 1990.

[103] Lance Williams. Casting curved shadows on curved surfaces. In *Computer Graphics (SIGGRAPH '78 Proceedings)*, volume 12, pages 270–274, aug 1978.

[104] Daniel N. Wood, Daniel I. Azuma, Ken Aldinger, Brian Curless, Tom Duchamp, David H. Salesin, and Werner Stuetzle. Surface light fields for 3d photography. In *Proc. SIGGRAPH 2000*, pages 287–296, 2000.

[105] S. Yamazaki, R. Sagawa, H. Kawasaki, K. Ikeuchi, and M. Sakauchi. Microfacet billboarding. In *Proc. 13th EG Workshop on Rendering*, 2002.

# Appendix A

# Technical explanation of XP nodes

## A.1 View Dependent Display Algorithm

We introduce an **XPnode xpVDObject** which implements the view dependent specification. The **XP** interface for this is as follows:

C++ class name: **xpVDObject** ; parent class: **xpNode**

Scene file name: *vdobject*

A view dependent model.

Scene file attributes:

- *cleanModels* - if true remove all objects without textures from the loaded models. The default is 1 (true)
- *mergeModels* - if true the models are considered as entire sceneGraphs. At the **xpVDObject** creation all models will be merged in one replacing all nodes that exist in more than one model (nodes with the same name) by a **pfVDObject**. If false only one **pfVDObject** will be created with the different models as children. The default is 1 (true)
- And a series of file/camera pairs using:
  - *file* - one of the model files to load; the value is the name of the file - e.g. *file*=house.pfb

- And one of the following:
  - *cameraFile* - file containing the camera information. A cameraFile needs to be a VRML file and a camera is coded as a Viewpoint in VRML - e.g. *cameraFile*=camera1.wrl
  - *cameraName* - name of the camera. Only possible if the last loaded file is a VRML and contains this camera - e.g. *cameraName*=camera1
  - *camera* - camera parameters to generate the camera. The parameters are the position X/Y/Z, the Euler angles H/P/R or the rotation axis and the rotation angle in degrees AX/AY/AZ/Angle, the largest field of view between Xfov and Yfov in degrees and the aspect ratio (weight/height) - e.g. *camera*="0 0 0 0 0 0 30 1"

Example:

```
vdobject(cleanModels=0, file="car.pfb", camera="10 0 1 10 10 5.5 30 1",
                        file="car2.iv", cameraFile="cameraIV.wrl",
                        file="car3.wrl",cameraName="camera")
```

Load file car.pfb and use camera at position "10 0 1" with Euler angles "10 10 5.5" fov "30" and aspect ratio "1". Then load car2.iv and get its camera values from cameraIV.wrl where only one camera is defined and then load car3.wrl and get its camera values from the viewpoint "camera" which is defined inside car3.wrl.

The actual implementation of the **xpNode** is contained in the new Performer node **pfVDObject**. The specification of the Performer node follows:

### A.1.1 pfVDObject

The function of this Performer node is to choose the appropriate representations of an object depending on the current point of view and blend them into one unique representation, the best one for the current point of view. This node is derived from a **pfGroup** node that allows the object to have several representations of the same object.

***pfVDObject** method function specification*

| | | |
|---|---|---|
| | **pfVDObject::pfVDObject** | (); |
| int | **pfVDObject::addChild** | (pfNode* child, pfCamera * camera); |
| void | **pfVDObject::setNumBlendPass** | (int sampleDistance); |
| int | **pfVDObject::getNumBlendPass** | (); |

*Description of method functions*

*The name **pfVDObject** is short for Performer View Dependent Object.*

**new pfVDObject** creates and returns a handle to a **pfVDObject**. It sets a pre-callback function for the cull traversal of this node. This function select which ones of the *N* representations are the best ones for the current point of view and computes a percentage of importance for each one of the selected representations. This percentage is used by each one of the children as the blend factor. This percentage is stored in a **pfFlux** because it is computed in the cull process and used in the draw process. The fact that all the computations are done in the pre culling function is justified because at the end of the computation process the **pfVDObject** will choose the correct representations and then the cull process will choose if they need to be rendered or not. Then the percentage of importance is given to each one of the children using a **pfFlux**. The pre cull function is defined as this:

- Compare the current view position with all the known cameras associated with the child representations of the current node.
- Choose the *N* best representations, according the criteria defined. Where *N* is the number of blend passes, or the number of children if it is smaller than the number of passes required.
- Compute a blending factor depending on the importance of each representation.
- Enable the cull and draw of each chosen representation and disable the others.

- Continue the cull process.

In this manner at each draw of the **pfVDObject** a maximum of *N* representations will be used, blending them with a weight that represents the level of importance of each representation.

**pfVDObject::addChild** adds a child to the **pfVDObject** and associates a camera that represents the point of view where this representation is best. This function sets the *pre* and *post draw* callback functions of the new child. The *predraw* function does the following things:

- Determine if this is the first object to be drawn; if this is the case it clear the zone that the object fills.
- Initialize OpenGL/Performer blending parameters using the percentage of importance given by the **pfVDObject** node (parent of the current node).
- Continue the draw process.

The *postdraw* function simply resets the state.

**pfVDObject::setNumBlendPass** sets the number of blend passes used during the draw process. Each blend pass is one of the representations of the object. By default the number of blend passes is two.

**pfVDObject::getNumBlendPas** gets the number of blend passes.

An example **pfVDObject** is shown in the figure A.1. This **pfVDObject** has three representations and it uses only two of them during the draw process.

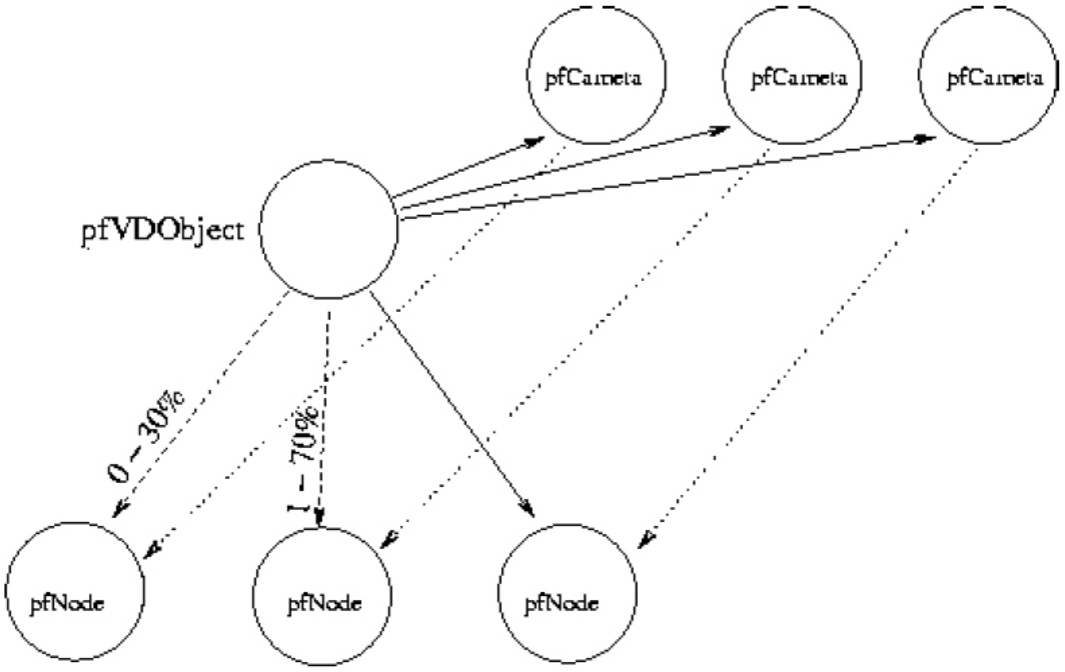



Figure A.1: Diagram of pfVDObject structure.

## A.2 Lighting

Since we are treating outdoors scenes, the computation of realistic and consistent lighting requires a relatively sophisticated sky and sun light model. We have based our work of Preetham et al [74]. We implement this via the **XPnode xpDayLight**.

C++ class name: **xpDayLight** ;

Scene file name : *dayLight*

Create and control sun and sky lighting.

Scene file attributes :

- *lattitude* - define latitude (0-360)
- *longitude* - define longitude (-90,90) south to north

- *standardMeridian* - define standard meridian, is actually a timezone number (east to west, zero based.)
- *julianDay* - define julian day (1-365)
- *timeOfDay* - define time of day (0.0,23.99) 14.25 = 2:15PM
- *turbidy* - define Turbidity (1.0,30+) 2-6 are most useful for clear days.

Example use:

```
dayLight( lattitude=43.70, longitude=7.27,
          standartMeridian=0, julianDay=130,
          timeOfDay=16.00, turbidy=4 )
```

The actual implementation of the daylighting operations is defined by the following Performer class, **pfSkyModel**, which is inserted into the Performer scene graph.

***pfSkyModel*** *method function specification*

| | | |
|---|---|---|
| static pfSkyModel* | **pfSkyModel::getInstance** | () |
| pfVec3 | **pfSkyModel::getSpectralRadiance** | ( const pfVec3 & normal); |
| pfVec3 | **pfSkyModel::getSkySpectralRadiance** | ( const pfVec3 & dir); |
| float | **pfSkyModel::getLatitude** | (); |
| float | **pfSkyModel::getLongitude** | (); |
| int | **pfSkyModel::getStandardMeridian** | (); |
| int | **pfSkyModel::getDayOfYear** | (); |
| float | **pfSkyModel::getTimeOfDay** | (); |
| float | **pfSkyModel::getTurbidity** | (); |
| void | **pfSkyModel::setLatitude** | ( float latitude ); |
| void | **pfSkyModel::setLongitude** | ( float longitude ); |
| void | **pfSkyModel::setStandardMeridian** | ( int standardMeridian ); |
| void | **pfSkyModel::setDayOfYear** | ( int julianDay ); |
| void | **pfSkyModel::setTimeOfDay** | ( float timeOfDay ); |
| void | **pfSkyModel::setTurbidity** | ( float turbidity ); |
| pfVec3 | **pfSkyModel::getSunPosition** | () const; |
| pfVec3 | **pfSkyModel::getSunSpectralRadiance** | () const; |

*Description of method functions*

**pfSkyModel::getInstance** returns instance of the skyModel since constructors are private (there is only one instance of skyModel).

**pfSkyModel::getSpectralRadiance** returns Spectral radiance as an XYZ color using the light equation:

$$\frac{4\pi}{n}\sum_{i=0}^{n} L_i^{sky} * Vis_i^{sky} * \overrightarrow{V_i^{sky}} \cdot \overrightarrow{N} + \frac{S}{m}\sum_{j=0}^{m} L_j^{sun} * Vis_j^{sun} * \overrightarrow{V_j^{sun}} \cdot \overrightarrow{N}$$

where $\overrightarrow{N}$ is the specified vertex normal, $\overrightarrow{V_j^{sun}}$ and $\overrightarrow{V_i^{sky}}$ are $m$ and $n$ random vectors in the sky and sun directions, $L_j^{sun}$ and $L_i^{sky}$ are the sky and sun spectrum radiance in the direction of $\overrightarrow{V_j^{sun}}$ and $\overrightarrow{V_i^{sky}}$, $Vis_j^{sun}$ and $Vis_i^{sky}$ are the visibility factor, $S$ is the solid angle of the sun and $4\pi$ is the solid angle of the sky.

**pfSkyModel::getSkySpectralRadiance** returns spectral radiance, as an XYZ color of the sky in the specified direction using the same equation as above.

**pfSkyModel::setLatitude, setLongitude, setStandardMeridian, setDayOfYear, setTimeOfDay** and **setTurbidity** set latitude (0-360), longitude (-90,90) south to north, standard meridian ( time-zone number; east to west, zero based...), julian day (1-365), time of day (0.0, 23.99) 14.25 = 2:15PM and turbidity (1.0, 30+) 2-6 are most useful for clear days.

**pfSkyModel::getLatitude, getLongitude, getStandardMeridian, getDayOfYear, getTimeOf-Day** and **getTurbidity** return latitude, longitude, standard meridian, Julian day, time of day and turbidity respectively.

**pfSkyModel::GetSunPosition** returns the position (actually the direction ) of the sun. The return vector is normalized and South = +x, East = +y and up = +z.

**pfSkyModel::GetSunSpectralRadiance** returns spectral radiance, as an XYZ colour, of the sun.

An additional class is required to represent the dome of the sky for display, according to the parameters specified by the **xpDayLight** node. Note that this class differs from the **pfEarthSky** since it uses the Preetham et al. model and its parameters to compute colours at each vertex of the dome.

### A.2.1 SkyDome

C++ class name : **xpSkyDome** ; parent class : **xpStaticObject** ;

Scene file name : *skyDome*

Display a sky dome according to the dayLight specifications.

Scene file attributes :

- *texture* - optional argument to define a texture which is mapped to the dome. This is blended with the computed parameters of intensity based on the skylight model. Typically this is a (set of ) photograph(s) of the sky taken at the moment of capture.

Example use:

```
skyDome(texture="CapturedSky.jpg")
```

The dome is implemented by a Performer node which exists in the Performer scenegraph.

**pfSkyDome** is derived from the class **pfGeode**

*Method function specification*

**pfSkyDome::pfSkyDome**(char* textureFileName)

*Description of method functions*

**new pfSkyDome** creates and returns a handle to a **pfSkyDome**. It sets a pre-callback function to the draw traversal of this node. This function checks if the skyModel has changed and updates the appearance of the dome.

To actually include and control consistently illuminated objects in the scene, we need to add an specific **XPnode** for virtual objects which can be lit, this node is **xpLitVirtualObject**. This is a "wrapper" definition, which then allows the control of lighting.

### A.2.2 litVirtualObject

C++ class name: **xpLitVirtualObject** ; parent class : **xpNode**

Scene file name: *litVirtualObject*

Light virtual object with current dayLight

Scene file attributes: none

Example use:

```
litVirtualObject() {
        object (file=duck.pfb, grab=1, wall=0)
        object (file=duck2.pfb, grab=1, wall=0)
}
```

The actual lighting operations require a set of grouping nodes for virtual objects, so that lighting can be achieved.

**pfGroupVirtualObjs** is derived from the performer class **pfGroup**.

*Method function specification*

```
        pfGroupVirtualObjs::pfGroupVirtualObjs   ()
int     pfGroupVirtualObjs::addChild             (pfNode *child);
```

*Description of method functions*

**new pfGroupVirtualObjs** creates and returns a handle to a **pfGroupVirtualObjs**. It sets a *pre-draw* function of this node. This function checks if the sky model has been changed and also detects motion and updates the colours of the vertices of all children.

**pfGroupVirtualObjs::addChild** appends a child to the **pfGroupVirtualObjs** and increments the reference count of child. It changes emisive colour per vertex of each object in this way: diffuse

colour material * spectralRadiance of the skymodel. Specular colour for all objects is computed using the direction of the sun, as defined in the sun/sky model.

## A.3 Shadows

The **xpNodes** implementing perspective shadow rendering capabilities are defined as follows:

C++ class name: **xpShadowMap**;

Scene file name: *%xpShadowMap*

Messages accepted by *xpShadowMap*:

- *spot* - enable spot: "*spot* on" or "*spot* off"
- *lightposition* - set light position "*lightPosition* 0 0 -200"
- *lightDirection* - set light direction "*lightDirection* 0 0 1"
- *lightAngle* - set light angle "*lightAngle* 45.0"

There will be a flag to set any object as a shadow caster or receiver.

The performer node implementing this functionality is **pfShadowMap**. This class is derived from the class **pfChannel**.

***pfShadowMap*** *method function specification*

```
        pfShadowMap::pfShadowMap      (pfPipe *p);
void    pfShadowMap::setCamera        (pfChannel *channel);
void    pfShadowMap::setScene         (pfScene *scene);
void    pfShadowMap::updateNear       ();
void    pfShadowMap::setSpot          (bool spot);
void    pfShadowMap::setLightPosition (const pfVec3 &lpos);
void    pfShadowMap::setLightDirection(const pfVec3 &ldir);
void    pfShadowMap::setLightAngle    (const float angle);
void    pfShadowMap::initReceiver     (unsigned unit, pfScene *scene);
```

*Description of method functions*

**new pfShadowMap** creates a new **pfShadowMap** on the **pfPipe** identified by pipe. The new **pfShadowMap** will be rendered by the pipe. new **pfChannel** creates and returns a handle to a **pfChannel**.

**pfShadowMap::setCamera** sets a pointer to the current **pfChannel**.

**pfShadowMap::setScene** sets the caster scene.

**pfShadowMap::updateNear** updates the near value from the image rendered by the channel.

**pfShadowMap::setSpot, pfShadowMap::setLightPosition, pfShadowMap::setLightDirection, pfShadowMap::setLightAngle** sets spot light, light position, light direction and light angle (ignored if is not a spot light).

**pfShadowMap::initReceiver** sets texture and texgen attributes to all receiver objects of the scene. Unit points to the texture unit where attributes will be applied to.