Designing tools for 3D content authoring based on 3D sketching

Emilie Yu

UNIVERSITÉ CÔTE D'AZUR

ÉCOLE DOCTORALE
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

THÈSE DE DOCTORAT

Conception d'outils de création de contenu 3D basés sur le dessin 3D

Emilie Yu

Centre Inria d'Université Côte d'Azur
Équipe GraphDeco

Présentée en vue de l'obtention du grade de docteure en Informatique d'Université Côte d'Azur

Dirigée par : Adrien Bousseau
Soutenue le : 21 décembre 2023

Devant le jury composé de :

Wendy Mackay, Directrice de Recherche, Inria Paris Saclay

Hongbo Fu, Professeur, City University of Hong Kong

Damien Rohmer, Professeur, Ecole Polytechnique

Martin Hachet, Directeur de Recherche, Inria Bordeaux

Hui-Yin Wu, Chargée de Recherche, Centre Inria d'Université Côte d'Azur

J. Andreas Bærentzen, Professeur, Technical University of Denmark

Adrien Bousseau, Directeur de Recherche, Centre Inria d'Université Côte d'Azur

Inria

CONCEPTION D'OUTILS DE CRÉATION DE CONTENU 3D BASÉS SUR LE DESSIN 3D

DESIGNING TOOLS FOR 3D CONTENT AUTHORING BASED ON 3D SKETCHING

Jury:

Présidente du jury / President of the jury
Wendy Mackay, Directrice de Recherche, Inria Paris Saclay

Rapporteurs / Reviewers
Hongbo Fu, Professeur, City University of Hong Kong
Damien Rohmer, Professeur, Ecole Polytechnique

Examinateur·ices / Examiners
Martin Hachet, Directeur de Recherche, Inria Bordeaux
Hui-Yin Wu, Chargée de Recherche, Centre Inria d'Université Côte d'Azur

Visiteur / Visitor
J. Andreas Bærentzen, Professeur, Technical University of Denmark

Directeur de thèse / Thesis supervisor
Adrien Bousseau, Directeur de Recherche, Centre Inria d'Université Côte d'Azur

Acknowledgements

> *Ce ne sont que "femmes puissantes" qui se sont "débrouillées seules" pour "s'en sortir". On les érige en icônes, ces femmes qui "ne se laissent pas faire", notre boulimisme d'héroïsme est le propre d'une société de spectateurs rivés à leur siège, écrasés d'impuissance. Être fragile est devenu une insulte. Qu'adviendra-t-il des incertaines ? De celles et de ceux qui ne s'en sortent pas, ou laborieusement, sans gloire ?*
>
> —*Lola Lafon, Chavirer*

Completing a PhD is a collective endeavor.

I was lucky to work with many collaborators, and each taught me something unique. I thank Rahul Arora, for taking me under his wing from the start of my research journey, and being always a source of support, care and motivation ; I thank Tibor Stanko for sharing with me invaluable math and geometry processing intuition that served me throughout my PhD. For my time at Adobe Research, I thank Cuong Nguyen for helping me wrangle the dreaded camera-model-conversion problem and Kevin Blackburn-Matzen for building a depth estimation pipeline that a novice in deep learning like me can use – the project would not have been possible without either of these things.

I have also encountered great mentors and supervisors. I thank Andreas Bærentzen for being with me from the start, always a source of positivity, inspiration and support at all levels of a project. I do not know if I would be doing research without our first project work during the master. I thank Karan Singh for his contagious enthusiasm and excitement about research, I have never been plagued by lack of motivation or interesting ideas in a project together. I thank Oliver Wang for being a VideoDoodles fan from the start, as well as for invaluable insights in this project. I thank Rubaiat Habib Kazi for patiently teaching me how to conduct interviews and other basic HCI knowledge that I lacked, as well as for being an inspiring mentor and overall great person and I thank Wilmot Li for pretty much handing off to me a great project idea and for taking me on as a summer intern – I wish I could have spent more in-person time together. I had the chance to spend time in the caring, supportive and joyful supervision of Fanny Chevalier, thank you for training me in the art of listening to people and understanding them – maybe the most valuable skill of the whole PhD – but also for encouraging me by example to become a great researcher and mentor. Finally, I acknowledge my chance of being supervised by Adrien Bousseau throughout the PhD. Thank you for leaving me space to do my own things while still always being available when I needed support. Our trusting and caring work relationship was invaluable in letting me find my own path in research. I have been inspired by your curiosity and passion for what you do in research,

that never come at the expense of other things and people in your life. I appreciate that this is a rare thing in the current academic world.

I deeply thank all past and present Graphdeco members. Each single person taught me something new or made me reflect on something that I never would have thought of otherwise. I am humbled by seeing each individual person going through the group evolve and grow. I am lucky to have been well advised by all those who graduated before me, in particular Siddhant, Stavros and Yorgos. I am also inspired daily by the energy of "young" PhD students and interns: Nicolás, Yohan Panagiotis, Petros, Aryamaan, Gilda, Vishal, Berend. I am excited to follow what you are up to and I will miss our time together in the lab. Doing a PhD is hard emotional work, and it would not have been possible without the care work provided to me by my labmates. Evolving in an environment where it is ok to say when something is not going well was vital for me. In particular, I want to acknowledge the invaluable support from David, Felix, Nicolas and Capucine. I deeply appreciate the time and effort we have put forth in supporting one another emotionally throughout the years. Thank you to Alban and Nicolas for our long discussions about academia, politics and everything in between. Thank you to Felix for always showing the purest support and enthusiasm towards my endeavors, yet always helping me question the things that matter. Thank you to David for teaching me so much about research and overall about how to be a good human being. I would also like to thank all those in the lab and around who shared and fueled my passion for bouldering and climbing – perhaps the only thing that kept me sane at times: Gilles, Cécile, Felix, Yorgos, Stef, Leo, Lorenzo, Antoine, Andreas, Panagiotis, Vishal, Alexandre. Lastly, I would like to thank George Drettakis, Adrien Bousseau and Guillaume Cordonnier for taking care of the life of the group and Sophie Honnorat for providing the best administrative support we could hope for. I would also like to thank Inria support staff in general, in particular Pascal Tempier from IT, HR and the canteen staff.

I also thank the members of dgp at University of Toronto whom I had the chance to meet during a summer stay. In particular I thank dgp-vis students: Eva, Anna, Moko, Book, Warren and Arnav. We had a lovely time both in and outside of the lab, and I am very grateful to each person in dgp-vis for fostering a caring and welcoming atmosphere. I would also like to thank all others who made my stay there amazing, Leen for being the best office mate I could ask for ; Blaine for making me discover a love for backpacking and hunting down new coffee shops ; Mian and Bogdan for being so welcoming and enthusiastic about everything food-related ; Kinjal for our small chats and snack breaks ; Mengfei for launching the best desk-side impromptu chats ; Sarah Kushner for sharing stories in Los Angeles ; also Kate, Bingjian, Yuta, Jiannan, Aravind and many others.

I thank my family for supporting me all my life and providing a good environment for me to find my way. And lastly I thank my friends outside of the small academic bubble, in particular thank you Charlotte for being always a source of energy and ideas throughout the years.

Résumé

L'accessibilité croissante du rendu 3D en temps réel a fait de la création de contenu 3D un moyen majeur d'expression et de communication. Mais la création de contenu 3D nécessite d'interagir avec des représentations numériques de la forme et de l'apparence qui sont compatibles avec les algorithmes de rendu et d'animation. Les maillages triangulaires, les modèles de matériaux paramétriques et les courbes d'animation sont bien adaptés aux opérations de rendu mais obligent les artistes à exprimer leurs idées en terme de commandes bas niveau qui doivent être apprises et mémorisées.

Dans cette thèse, nous explorons l'utilisation de *coups de crayon 3D* ou *courbes 3D* comme moyen pour les artistes d'exprimer leurs idées. Inspirés par la façon dont les artistes travaillent avec un pinceau et une toile, nous considérons ce geste de coup de crayon de l'artiste comme la principale commande d'entrée du système de création. Les courbes 3D sont des primitives flexibles qui peuvent être créées dans des interfaces utilisateur 2D ou dans des interfaces de réalité virtuelle (RV), et elles peuvent encoder une forme 3D ou l'apparence finale d'une peinture 3D. La conception d'outils qui considèrent les courbes 3D comme une représentation de la forme ou de l'apparence ouvre un espace vaste et passionnant à explorer.

Les designers peuvent utiliser les courbes 3D comme une représentation partielle de la forme 3D. Nous étudions comment convertir un croquis 3D clairsemé en un modèle de surface 3D. Étant donné que les courbes caractéristiques de l'objet sont un élément important de la forme designée et qu'elles sont représentées avec soin dans l'esquisse, nous reconstruisons une surface lisse par morceaux qui préserve ces courbes caractéristiques. En obtenant une surface à partir des courbes 3D non structurées, notre algorithme permet le rendu 3D de la forme décrite par l'esquisse.

Pour mieux comprendre comment les courbes 3D peuvent représenter non seulement la forme mais aussi l'apparence des objets, nous étudions la pratique de la peinture en RV au sein d'une communauté d'artistes qui travaillent avec un logiciel commercial de peinture en RV. Sur la base de cette étude, nous proposons une conception et une implémentation pour les "calques 3D", une nouvelle primitive d'interaction pour la peinture RV qui considère les courbes 3D comme représentation à la fois de la forme et de l'apparence 3D, tout en découplant l'édition de ces deux éléments. Inspirés par l'utilisation de la composition de calques en peinture numérique 2D, nous proposons un processus non destructif pour modifier l'apparence d'une peinture RV.

L'animation dessinée à la main est un moyen expressif de créer une animation avec des coups de crayons. Dans les animations de type "video doodles", les artistes créent un dessin animé qui semble bouger dans le même espace 3D qu'une vidéo filmée. Prendre en

compte les effets de perspective et les occlusions tout en dessinant des courbes 2D n'est pas une tâche facile, c'est pourquoi nous utilisons des techniques de vision par ordinateur pour placer les courbes dans l'espace 3D et les rendre en respectant le contexte de la vidéo. Nous concevons une interface utilisateur en 2D qui ressemble aux outils traditionnels d'animation en 2D, afin de permettre aux utilisateurs qui ne sont pas familiers avec les outils 3D de créer de telles animations.

Globalement, nous montrons que les courbes 3D sont une représentation puissante pour la création de contenu 3D en proposant trois systèmes qui exploitent les courbes 3D ou les dessins 3D en tant que primitives d'interaction pour des applications créatives allant de la création de forme à celle d'apparence et d'animation. Nous abordons la conception de ces systèmes sous deux angles complémentaires ; nous développons de nouveaux algorithmes pour interpréter les dessins et les commandes de bas niveau de l'utilisateur, et nous concevons des interactions qui permettent aux utilisateurs d'exprimer leurs intentions haut niveau.

Mots-clés: Dessin 3D – Réalité Virtuelle – Modélisation 3D – Outils de création

Abstract

The increasing accessibility of real-time 3D rendering hardware has made 3D content creation a major means of expression and storytelling. But authoring 3D content requires interacting with the digital representations of shape and appearance that are compatible with rendering and animation algorithms. Triangular meshes, parametric material models and animation curves, while well suited to downstream computation, require artists to convey their ideas in terms of low-level commands that need to be learnt and remembered.

In this thesis, we explore the use of *3D strokes* as a way for artists to express their ideas. Inspired by the way artists work with brush and canvas, we consider the artist's mark-making gesture as the main input to the authoring system. 3D strokes are flexible primitives that can be created in either 2D desktop user interfaces or in virtual reality (VR) interfaces, and they can encode a 3D shape or likewise the final appearance of a 3D painting. Designing tools that consider 3D strokes as a shape or appearance representation opens a large and exciting space to explore.

Designers can use 3D strokes as a partial representation of 3D shape. We investigate how to interpret an unstructured sparse 3D sketch into a 3D surface model. Since feature curves are a prominent part of the design and are finely depicted by the sketch, we recover a piece-wise smooth surface that preserves those sharp features. By obtaining a surface from unstructured 3D strokes, our algorithm allows to render the shape depicted by the sketch.

To better understand how 3D strokes can depict not only the shape but also the appearance of objects, we study the practice of VR painting among a community of artists that work with a commercial VR painting software. Based on this inquiry, we propose a design and implementation for 3D-Layers, a new interaction primitive for VR painting that embraces 3D strokes as the sole representation for both 3D shape and appearance, yet decouples edition of these two elements. Inspired by the usage of layer compositing in 2D digital painting, we support a non-destructive workflow to edit the appearance of a VR painting.

Hand-drawn animation is an expressive way to convey an animation with strokes. In "video doodles" animation, artists create an animated doodle that seems to live in the same 3D space as a captured video. Taking into account perspective effects and occlusions while drawing 2D strokes is not an easy task, so we leverage computer vision techniques to place strokes in 3D space and render them with respect to the video context. We design a 2D user interface that resembles traditional 2D motion design tools, to enable users unfamiliar with 3D tools to create such animations.

Overall, we show that 3D strokes are an expressive representation for 3D content cre-

ation by proposing three systems that leverage 3D strokes or 3D sketches as interaction primitives for creative applications spanning shape, appearance and animation authoring. We approach system design from two complementary perspectives ; we develop novel algorithms to interpret strokes and low-level user input, and we design interactions to provide new ways for people to express their high-level intent.

Keywords: 3D sketching – 3D modelling – Virtual Reality – Creativity Support Tools

Contents

Chapter I

Introduction

Contemporary to the work on this thesis, the technology industry has pushed to increase the adoption of virtual reality (VR) hardware in an attempt to drive social acceptance of the so-called "metaverse". Namely, in 2020 Meta has released a new VR headset that sold 10 million units [316], and in 2023 Apple announced a new augmented reality (AR) headset [17]. Corporate advertisements and technologists herald the start of a new era in computing, from the way we work to how we connect with friends and family, yet it is unclear for society at large whether this technology constitutes an advancement in quality of life or a new attempt by tech giants to capture attention and extract value from users [351]. In this rapidly evolving technology and social landscape, we choose to study the concrete possibilities that new input, display, and environment capture technologies present to artists and how software design might support or influence the authoring workflow. Should the forecasted adoption of VR hardware and the metaverse happen, creative applications that are accessible to novices can help democratize 3D content creation and ensure that the power to tell stories and design virtual surroundings is shared with the general public.

Developing software and algorithms to aid people in the authoring of three dimensional (3D) content is a core research question in computer graphics. Ivan Sutherland's seminal work on *Sketchpad* [300] in the 1960s exemplifies that effort of making shape edition on the computer easier and more accessible. While at the time, creating shapes on the computer was possible only through written commands, Sutherland showed that human and computer could instead "converse rapidly through the medium of line drawings", by using a pen interface to point at the screen and draw lines and curves. *Sketchpad* stands as an early example of a new way of designing interactions between human and computer, in which the human's physical actions – of moving the pen – yield rapid operations onto an object of interest – the drawing – whose graphical representation is present at all time on the screen and always up to date.

An interface presenting these qualities was described as a *Direct Manipulation* interface by Shneiderman [282], by opposition to the prevalent user interfaces at the time, which were based on text command inputs. Hutchins et al. [138] further characterize what makes an interface feel "direct", by conceptualizing an interface as a bridge over the gap between the state of the digital system and a user's goals. In particular, they describe the process of translating one's intent into a set of low-level actions on the computer's

data as bridging the *gulf of execution*. They describe how all interfaces can introduce two types of distance that describe how large that gulf is: *semantic distance* describes how close commands or actions available in a system are to how a person "thinks of the task", and *articulatory distance* measures how close the physical form of the input that the user performs matches the corresponding command in the system. This description of interactions is general and can be applied to all types of applications such as text processing, book keeping, and data tabulation. In particular, I claim that the concepts of direct manipulation interfaces are relevant to design *visual content authoring tools*, since reducing semantic and articulatory distances can support artists in quick externalization of ideas and promote exploration [255].

Despite tremendous progress in making 3D content creation accessible and the availability of quality consumer software, the craft and technical mastery of 3D creation remains daunting to many. Most professional 3D modeling software relies on a 2D mouse user interface, so they provide tools that let users edit 3D data by dragging geometric primitives along constrained 2D planes or 1D axes. In this interaction, the *articulatory distance* between the 2D gesture and the 3D editing operation is arguable quite large. Furthermore, many 3D modeling tools make users work with *low-level representations* of geometry such as polygonal meshes, since they are the de facto way to encode digital shapes. The authoring process consists in iteratively deforming this representation – eg extruding a polygonal face or moving a primitive shape in a CSG model. Working with those tools require the artist to bridge the *semantic distance* by learning how to convey their idea in terms of valid deformation operations on this strictly defined digital material. In this setting, bridging the *gulf of execution* [138] can feel prohibitively difficult for an artist that doesn't have enough experience working with such digital representations.

In this thesis, we explore the use of *3D strokes* as a "high-level language" between a person's goals and the 3D authoring system. Inspired by the way artists work with brush and canvas, we consider the artist's mark-making gesture across the domain as the main physical input action to the authoring system. Creating a stroke via a motion of the hand through space arguably presents a small articulatory distance. We embrace the inherent lack of structure and freeform nature of sketched strokes in our system design. Our algorithms and user interfaces do not expect artists to sketch in a particular way, which aims at bridging the semantic distance by letting the system interpret raw user input instead of asking the user to make their input conform to system requirements. In the following chapter, we define what 3D sketching is and we will see that 3D sketching in the broad sense maps out a large space of possible designs and applications. We will proceed by narrowing down on the particular topics we choose to study in this thesis.

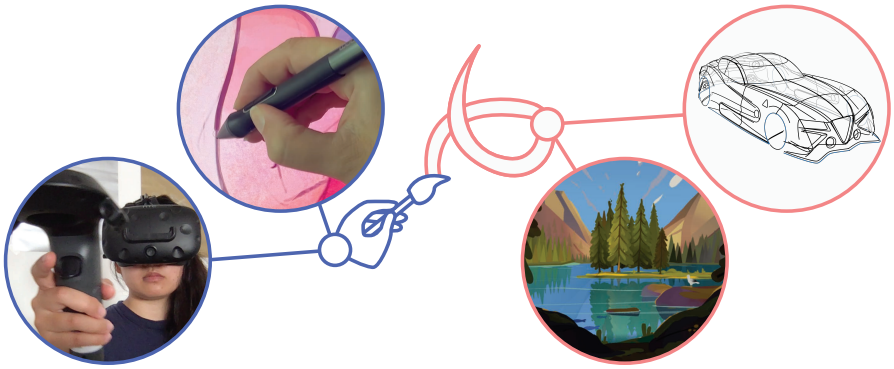


Fig. I.1: In the following section, we define what we mean by 3D sketching (middle), and in particular we look at input methods for 3D sketching (left) and define what the 3D strokes depict in different use cases (right). Second image from the left ©Lois van Baarle ; third image ©Nick Ladd ; fourth image ©James Robbins.

I.1 3D sketching as a creative medium

This work uses the term *3D sketching* to refer to the act of expressing an idea or concept by creating 3D strokes embedded in a 3D space that can be empty or contain some context to which the strokes relate. We use the term *3D strokes* to refer to 3D curves that have been "marked" in space [53] by the direct gesture of an artist's hand – usually holding an appropriate tool. This definition is voluntarily broad, and opens up a large space to explore in this thesis. The following section delves into the specifics of the input methods used to create 3D strokes (Fig. I.1, left), and the different meanings that such a stroke may carry (Fig. I.1, right) ; each of these options comes with its own set of challenges. I conclude by narrowing down on which challenges we choose to explore more in depth in this thesis.

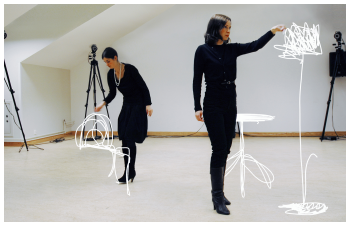
I.1.1 How to create 3D strokes?

2D input Traditionally, user interfaces for display and for input are two dimensional (2D), like the computer display and light pen used by Sutherland. Sketching on a tablet is very close to the action of sketching with pen on paper, making it easy to transition to, and just as expressive as sketching on paper. One way to create 3D strokes with pen and tablet input is for a user to draw a 2D stroke and "lift" it to 3D following some rule determined by the interface designer – often by making the assumption that the 2D stroke depicts a projection of the 3D stroke onto a camera plane. Such a tool for 3D content creation lets the artist's hand gesture input directly affect the shape of a stroke upon creation.

Challenges As interface designers we are left with the responsibility to decide how the user's 2D strokes should be lifted to 3D space. This problem is ill-posed since one 2D

stroke can be lifted to many 3D strokes: solving it requires choosing suitable heuristics or designing ways to let the user lift the uncertainty. If the user wishes to sketch in an existing 3D context, the challenge becomes to decide how to relate the strokes to that context.

3D input Another way to create 3D strokes is to record an artist's hand gesture in 3D space, and to let that 3D gesture be the direct source of creation of a 3D stroke, as if the artist were holding a small LED light that forms a 3D light trail in a long exposure photograph (see inset). With early hardware for 3D input and hand-tracking, Galyean and Hughes [112], Schkolne et al. [267] among others introduced that idea of creating shape through such a process, referred to as analogous to "squeezing toothpaste" or more formally as "repeated marking". With the increasing availability of virtual reality (VR) hardware that put 6 DOF tracked controllers in the hands of many, commercial applications such as TiltBrush and Quill have adopted this idea of a hand-held 3D brush tool that leaves digital 3D marks in space. Creating 3D content with such a VR paint brush compared to a 2D pen and tablet bridges the gap in degree of integration, as we have a 3D input device for 3D content creation.


©FRONT Design [107]

Challenges Performing gestures in 3D space with a high degree of accuracy is an activity that is difficult to do without extensive practice. Compared to marking 2D paper, for which we have trained since our early childhood doodles, sketching in 3D space is a novel activity to most people and presents a steep learning curve. This leads to a challenge for users in terms of creating accurate and intentional strokes, as shown by experimental studies [19, 211].

I.1.2 What do 3D strokes depict?

Just as physical drawing media can be used to depict a sphere in very different ways, 3D strokes are a flexible primitive that can be used with a variety of depiction styles (Fig. I.2).

Shape In an analogy of the line drawing in Fig. I.2a, an artist can depict a 3D surface by placing a few strokes in 3D space. For example the sphere drawing in Fig. I.2c is composed of very sparse strokes, yet the viewer's brain can understand that this represents a sphere. Line drawings are a popular way to express design concepts in early stages of design, in part because of their remaining uncertainty and ambiguity, which stimulates the viewer's creativity while filling the gaps. Expressing a design with a few lines and curves is also a valuable tool for product or automotive design, where *feature lines* of an object are often a deliberate part of crafting its final look. In 3D sketching, depicting a 3D surface with few strokes is appealing for the same reasons and additionally presents an opportunity for

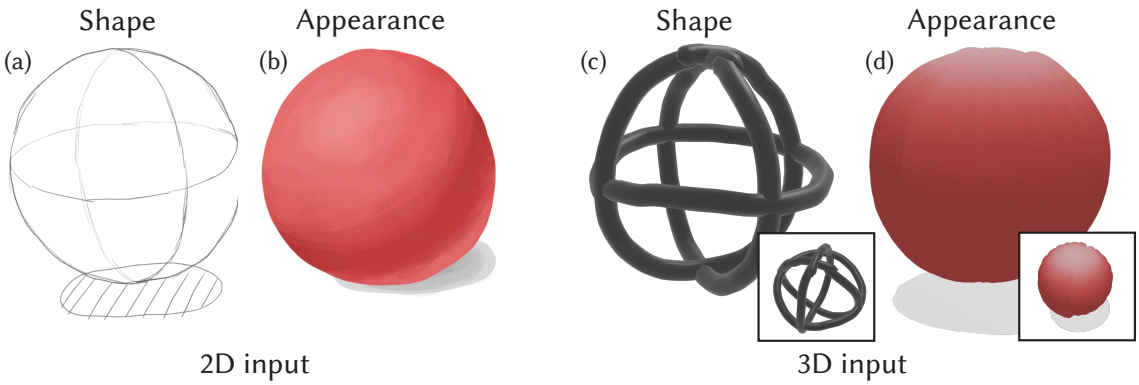


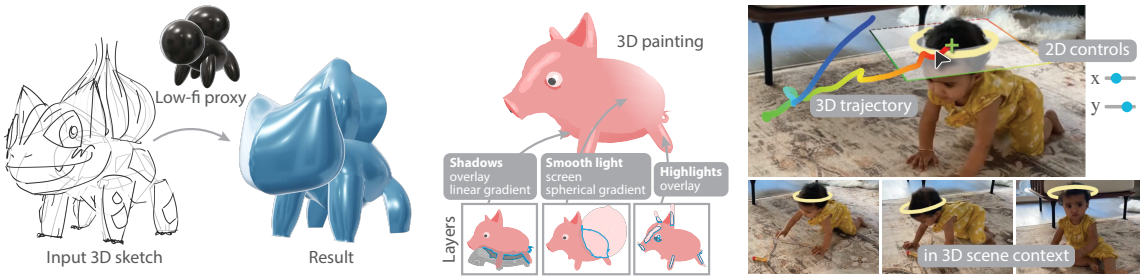
Fig. I.2: A sphere can be depicted with 2D strokes by sketching on a pen tablet (a, b), or with 3D strokes by sketching in VR (c, d). The sphere can be depicted by a suggestion of its *shape* (a, c), or by directly rendering its *appearance* (b, d). Sketching with 3D strokes yields a 3D artwork that can be viewed from another viewpoint (inset).

the 3D sketch to serve directly as a rough starting point for a more defined 3D modeling step.

Challenges Just like with the sphere example shown earlier, observing a 3D sketch attentively is often enough for a human to perceive a reasonable approximation of the 3D surface depicted. However, from the perspective of a computer, the sparse set of strokes of a 3D sketch form an uneven sampling of the underlying surface, which makes it very difficult to infer what this surface might be. While the 3D strokes are related to each other through near-intersections or other geometric criteria such as symmetry, these relations do not form a definite and clear structure suitable for analysis. Nevertheless, obtaining a well-defined 3D surface from such an unstructured 3D sketch might be a valuable starting point to other downstream design steps, such as rapid prototyping, or CAD modeling.

Appearance While the thin and sparse pencil strokes of Fig. I.2a depict a sphere by suggesting its *shape*, the thick dense and colored brush strokes of Fig. I.2b depict a sphere by representing its *appearance*. Similarly, 3D strokes can be used to directly represent appearance (Fig. I.2d), and by arranging many colored brush strokes, skilled artists can depict complex 3D scenes. Such "3D paintings" open unique possibilities in the space of 3D content creation: artists can create scenes that present all the expressivity and hand-crafted aesthetic of a traditional painting, yet can be explored by free viewpoint navigation as any 3D scene.

Challenges While the design of 2D digital painting applications has been a topic of research for decades, how to design and implement practical and expressive "3D painting"



(a) Recover piecewise-smooth surfaces from an unstructured 3D sketch

(b) Layered appearance depiction in VR painting

(c) Hand-drawn animation in 3D with a 2D user interface

Fig. I.3: We present three possibilities to leverage 3D sketching as an authoring tool. (a) From 3D strokes sketched in VR we can recover a piecewise-smooth surface that fits the sketch. In ambiguous cases, a user-defined proxy surface (a, top) can be used to guide surface fitting. (b) We augment VR painting with a data structure and rendering algorithm to support organizing strokes in stacked layers, enabling a more precise and non-destructive coloring workflow to hand-paint 3D appearance. (c) We facilitate authoring hand-drawn animation on top of existing videos by combining a 2D keyframing and drawing interface with a 3D tracking method.

tools is still an open problem. It is not clear how well-established tools and paradigms from 2D painting can be ported to this different domain. Unlike 2D paintings, 3D paintings do not rely on the concept of a *substrate* onto which paint strokes are accumulated by layering and mixing – eg the oil painter's canvas, or Photoshop's background layer. A lot of features in 2D digital painting are designed around this mental model: layering and compositing of strokes, textured brushes that simulate traditional media applied onto a canvas, cutting regions and masking layers. Current commercial 3D painting softwares lack analogous functionalities and struggle to support artist workflows.

I.1.3 Our work in this space

In this large and exciting space of what 3D sketching can be, we choose to explore three possibilities during this thesis. First we delve into how VR hardware could support 3D sketching creative tools, by studying both how sparse 3D sketches could be surfaced (Chapter III) and how to improve the design of 3D painting tools by designing new interaction primitives inspired from 2D digital painting tools (Chapter V). Then we turn to the case of pen and tablet 2D interfaces, and how these can be used to support creating 3D doodles in the context of an existing video (Chapter VI).

3D sketching is used by communities of artists that develop ad-hoc methods to achieve a diverse set of creative goals, such as industrial 3D design, illustration, game asset design, and motion design. By studying tutorials, conducting interviews and browsing created artifacts, we aim to better understand *how* artists work with 3D sketching, in which scenarios they encounter the challenges that come with 3D sketching, and what work-arounds they have found to overcome those challenges. Adopting this methodology that

is commonplace in human-computer interaction research [212] helps us design solutions to the challenges outlined previously (Section I.1.1 and Section I.1.2).

First, in our work on surfacing VR sketches (Chapter III), we tackle imprecision and ambiguity of the description of a shape through a freehand 3D sketch (see Fig. I.3a). We formulate well-posed objectives for our surfacing algorithm by observing common depiction strategies in VR sketches, and reduce the overall complexity of the problem by letting the user control the coarse look of the final shape by a rough 3D model. Then we seek a better understanding of how VR artists depict appearance in 3D paintings by conducting interviews with 4 participants (Chapter IV). This study led us to propose a new way to organize 3D paintings to give finer control to the artist over colors while relaxing the need for precise motions (Chapter V. see Fig. I.3b). Finally we study how people create animated doodles on videos in order to design a user interface that puts the power of deep 3D scene understanding in service of a controllable and playful animated doodles creation workflow (Chapter VI, see Fig. I.3c). By leveraging 3D scene understanding, we can "lift" 2D animated doodles to the 3D context of a captured video.

I.2 Outline

The rest of the thesis is organized as follows. Chapter II presents previous work on tools for 3D content authoring based on direct manipulation interaction. We will show that developing tools for 3D shape, appearance and animation authoring that feel direct has a long history, with rich contributions from both a technical and interaction design point of view.

Chapter III explores how 3D strokes can be used as a representation of *shape*. We show that a sparse 3D sketch can be converted to a piecewise-smooth 3D surface that preserves characteristic feature lines of the sketch. Our key insight is that 3D strokes encode not only samples on the 3D surface, but can also indicate where sharp features lie on the surface.

3D strokes can encode not only shape ; they can represent final *appearance* of a 3D scene. In VR painting, artists use colored brush strokes to directly depict appearance and create complex scenes that have a unique visual style, hard to achieve with traditional 3D authoring tools. We study the emergent practice of VR painting in Chapter IV through 4 semi-structured interviews with practicing VR artists, and provide a preliminary analysis.

In our formative study, we find out that artists use brush strokes to depict *both* shape and appearance in their painting, with a strong coupling between these two. While this enables artists to directly manipulate both, this coupling makes editing appearance without affecting shape difficult, and hinders making changes in appearance to a VR painting. In Chapter V, we propose a new interaction primitive inspired by 2D bitmap layers in digital painting to alleviate this challenge. We showcase two complex workflows

achieved with our prototype, and propose a preliminary plan for a user study to test our design.

In Chapter VI, we show that 3D strokes can also be helpful in the context of hand-drawn *animation*. We focus on the creation of video doodles: simple hand-drawn animation overlaid on a video, in which the drawn doodles have to look as if they were in the captured scene. We propose a user interface that enables novice users to create video doodles, by decomposing the authoring process into on one hand 3D trajectory estimation of a "scene-aware 3D canvas", and on the other hand sketching multiple frames.

Finally, we conclude this thesis by proposing future research directions in Chapter VII.

I.3 Publications

The work of Chapter III has been published:

> Emilie Yu, Rahul Arora, J. Andreas Bærentzen, Karan Singh, and Adrien Bousseau. 2022. *Piecewise-smooth surface fitting onto unstructured 3D sketches.* In *ACM Transactions on Graphics* (Proceedings of SIGGRAPH), volume 41-4, pages 1-16.

The work of Chapter VI has been published:

> Emilie Yu, Kevin Blackburn-Matzen, Cuong Nguyen, Oliver Wang, Rubaiat Habib Kazi, and Adrien Bousseau. 2023. *VideoDoodles: Hand-Drawn Animations on Videos with Scene-Aware Canvases.* In *ACM Transactions on Graphics* (Proceedings of SIGGRAPH), volume 42-4, pages 1-12.

Chapter IV and Chapter V present preliminary results of ongoing work that has not been published yet.

Chapter II

Background and Related Work

This chapter provides background for the thesis as a whole, highlighting common themes that span all three of our more specific projects, and situating them in the current landscape of research and industry tools for 3D digital content authoring. Each of the core chapters (Chapters III to VI) will give background concerning the particular authoring scenario that the chapter focuses on.

In this chapter we map out the landscape of tools and efforts to make digital 3D content authoring more direct and more expressive. Digital 3D content serves varied applications, from creating frames in a feature animated movie, to generating the commands that a laser cutting machine will use to make parts for a kitchen table. All digital 3D content destined to be visualized on a computer is composed of different data structures with particular values, that a rendering algorithm interprets to form an image. Three core aspects that define digital 3D content are *shape*, *appearance* and *motion*.

For a more concrete example, let us look at this crow character, created for a small video game project (Fig. II.1). The *shape* (Fig. II.1a) defines what space the crow occupies in 3D, it defines the outline of the crow character from any given viewpoint, as well as what kind of creases, kinks and bumps the crow presents. Shape can be authored in many different ways as we will see in Section II.1, but in the end it is typically an assembly of geometric primitives such as 2D polygons – triangles, in the case of the crow. Knowing the shape of the crow is enough to draw its outline, but by defining *appearance*, we can convey that this crow character is purple and we can make it match the flat visual style of the game (Fig. II.1b, bottom), or instead make it look like a plastic toy (Fig. II.1b, top). The appearance of the crow encodes some information about how a computer program should render the crow to a 2D frame. It defines visual aspects about the surface such as color, and how the surface reacts to virtual incoming light. It also encompasses information about the environment in which the crow lies, such as the lighting setup and surrounding objects that might affect the appearance of the crow by casting shadows or reflecting light [84]. Appearance authoring is a vast topic, in Section II.2 we focus on methods to author appearance by painting strokes. Finally, the crow character can be made to appear in *motion* (Fig. II.1c) by authoring motion via animation. We end this landscape of authoring tools by giving in Section II.3 an overview of animation methods aimed at novice users, as they, in particular, strive to design more direct interaction techniques for animation.

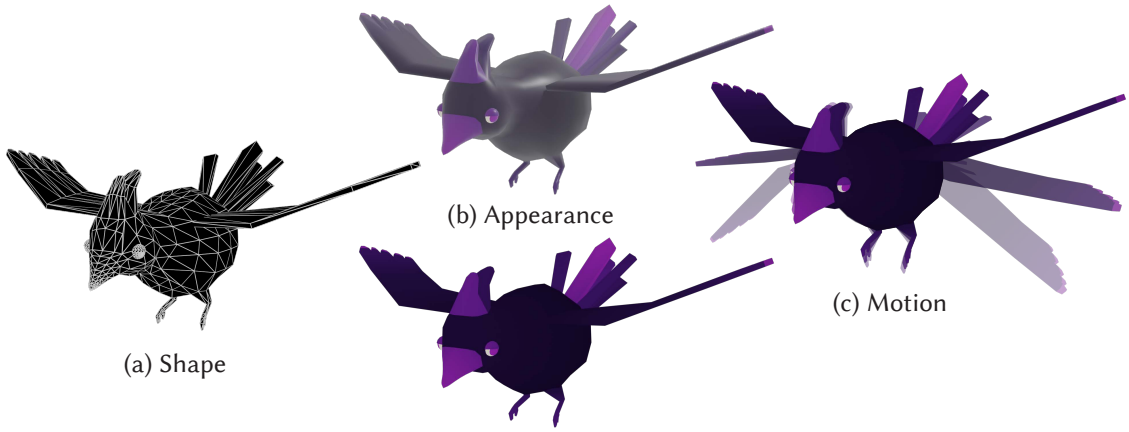


Fig. II.1: Authoring digital 3D content encompasses many aspects, among which we focus on authoring 3D shape (a), appearance (b) and motion (c).

VR, AR and other 6-DOF input devices have driven a lot of research in 3D content authoring systems. We discuss those contributions across all sections of this chapter, as we will see that they can be well classified along the same categories as 2D desktop systems.

II.1 Shape authoring

Methods and user interfaces to create and edit 3D geometry cover a vast body of academic and industry efforts, and have been designed to cater to a variety of use cases, such as industrial design, 3D game asset creation or digital fabrication. For brevity, this section dwells on the landscape of work that most closely matches our effort of making tools that let users author shape by direct manipulation. We organize related work in this field by the kind of *representation* that they let users manipulate. The choice of shape representation in a particular authoring tool is sometimes driven purely by practical aspects of performance or compatibility concerns, yet the representation introduces different trade-offs and constraints. This affects whether a given tool supports a particular artist's workflow, just like how choosing to work with vector curves will support a different process than working with bitmap brushes [191].

> If it is true that the way we think about something is shaped by the vocabulary we have for talking about it, then it is important for the designer of a system to provide the user with a good representation of the task domain in question.
>
> —Hutchins et al. 1985, Direct manipulation interfaces

II.1.1 Manipulating surfaces

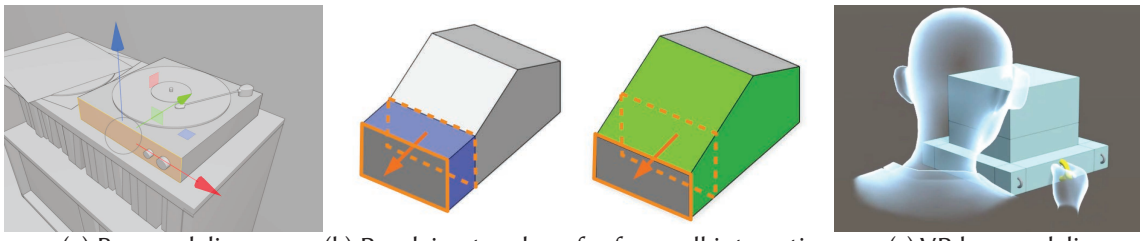
A popular representation for 3D geometry authoring is to encode a shape as a surface that defines its outer boundary. This surface is discretized as a *polygonal mesh*, for example composed of quadrilaterals or triangles. The user can visualize the discretization elements such as vertices, edges and faces, and those elements are the main object of interest in the interaction ; the user can select them and apply operations onto them.

Box modeling

A complex surface can be created by starting from a simple primitive shape such as a cube, and iteratively applying operations onto the elements of the polygonal mesh – a process referred to as box modeling [315] (see Fig. II.2a). By extruding faces, subdividing face loops and pulling on edges and vertices, the artist can create a *low-polygon* mesh that captures the rough shape they envision [38]. This low-polygon mesh can optionally be subdivided to obtain a smooth looking surface. This workflow gives the artist a high level of control over the *topology* of the polygonal mesh they are forming – ie how the discretization elements are assembled.

While this control enables skilled modelers to build a surface with few discretization artifacts and deliver an optimized mesh, working at the polygon level widens the "gulf of execution" by letting the user figure out how a desired shape modification translates to topology modifications. Previous work has sought to simplify this process by automating topology modifications for simple operations [200] (see Fig. II.2b), or by converting an arbitrary surface into a quadrilateral mesh with a topology that facilitates this modeling workflow [238]. Doing box modeling in VR can make grabbing and pulling on primitives in 3D quicker and more direct than with a 2D desktop interface (see Fig. II.2c) [25, 246, 287], but the user is still in charge of constructing a valid shape in terms of mesh topology, or is limited to interacting with shapes via rigid transformations [299]. This extra cognitive load is not appropriate in early stages of the design process, when sketching out an idea must be a "quick" and "inexpensive" process [52].

Thus we see artists start the box modeling process with references from earlier stages of design, which can be 2D images or sketches that they "trace" over from a particular viewpoint [252]. In particular in VR modeling tools we observed artists using a 3D rough sketch done in VR as a visual scaffold to help the modeling step [287]. Our work in Chapter III explores how this kind of ideation 3D sketch can be used to automatically model a 3D shape, and the other projects presented in this thesis (Chapters III and V) consider only strokes as interactable geometric primitives, hiding the complexity of surface discretization away from the user.



(a) Box modeling in Blender

(b) Resolving topology for face pull interaction [Lipp et al. 2014]

(c) VR box modeling [Bærentzen et al. 2019]

Fig. II.2: In box modeling, users apply operations onto elements of a polygonal mesh such as faces (a) via gizmos [38]. Ensuring correct mesh topology updates matching a desired modification (orange arrow) is not always trivial (b) [200]. VR interfaces can make mesh manipulation easier (c) [25].

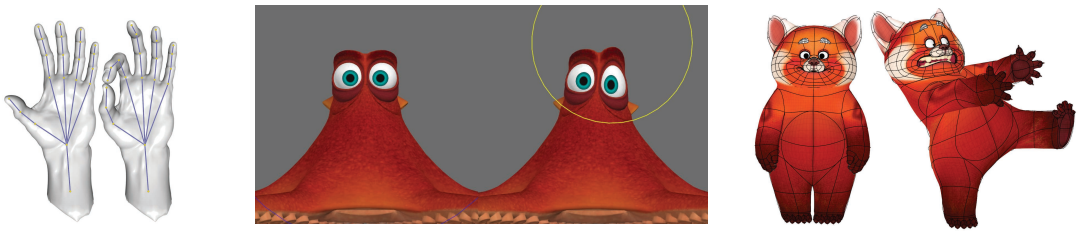
Surface deformation

Digital surface deformation operations are designed as analogies to modeling actions that take place with physical media, such as clay sculpting, in order to build on users' pre-existing understanding of the non-digital world [151]. The idea of taking inspiration from the actions in sculpting can be found in the early work of Coquillart [67] that uses a 3D lattice as handle to deform the 3D surface. More recent work on surface deformation made manipulation simpler by allowing more flexibility in the handle construction and achieving real-time deformation [153] (see Fig. II.3a). The user can also grab vertices as handles and see the surface deformed in a way that matches "naive physics" expectations [151], by solving a variational problem with a well-chosen energy that models surface stretching and bending [44] or rigidity [291].

Another popular deformation user interface consists in exposing a "brush" tool that defines a 3D deformation field corresponding yet again to sculpting analogies such as grab, twist, pinch [75] (see Fig. II.3b). Such modeling tools are implemented in commercial desktop software [39, 215] and for VR hardware to enable 6-DOF manipulation of deformation brushes [7].

While modeling by deformation is a very popular and efficient method for 3D shape editing, it is more suitable to refine an already-existing coarse version of the desired shape than to create a shape from scratch. We see these techniques as complementary to our work, since we focus more on the initial stage of design when the idea must be externalized from a blank slate.

Specifically, some surface deformation techniques use on-surface curves or curve networks as meaningful handles, motivated by the observation that profile curves and sharp feature curves are strong shape descriptors that artists want to have explicit control over. Singh and Fiume [286] show the expressive power of using a few on-surface curves to deform a surface, and later work explored the use of networks of curves that are either



(a) Skeleton handles [Jacobson et al. 2011]

(b) Sculpting brushes [De Goes and James 2017]

(c) Curve network handles [De Goes et al. 2022]

Fig. II.3: Surfaces can be deformed by a user through different interaction metaphors. The system can expose different kind of handles such as a skeleton (a) [153], or a curve network (c) [76]. Alternatively, the user can use a "brush" that displaces the content inside by simulating an elastic material being pulled (b) [75].

hand-authored [76] or automatically generated given a shape [111] to enable deformation of complex character meshes (see Fig. II.3c) and regular man-made shapes respectively. We push the idea of using curves for shape definition further, by using curves not only for editing an existing surface but also for defining that surface from scratch in Chapter III.

Composing surfaces

Finally, a last notable interaction metaphor that can power direct manipulation interfaces is to consider how multiple surfaces can be composed together. A closed surface defines a solid volume, and multiple such volumes can be composed with boolean operations. This method – called Constructive Solid Geometry (CSG) – enables the construction of complex shapes via the manipulation of simple parametrized primitive shapes such as boxes, spheres, or tori [277]. More generally, both these primitive shapes and more complex organic shapes can be described by an implicit surface representation: the surface is the set of 3D points in which a scalar function evaluates to a fixed value [41]. In a user interface for CSG or implicit surface modeling, the user can edit the primitive shapes themselves by changing their parameters or by moving them in 3D space, and they can define how the shapes are composed together by choosing a boolean operation and a blending mode [15, 135]. This type of shape authoring workflow has been implemented in commercial software, both for 2D desktop interfaces and for VR [96, 130, 219]. Making CSG edition easier for users is an active field of research, for example to enable the conversion of other shape representations to CSG [89] ; to define variants of CSG representation that allow more powerful edits [90] ; or to let users manipulate a complex CSG program with direct manipulation paradigms [218].

Composing surfaces to obtain complex effects is something we explore in Chapter V, where 3D strokes can act on the color of their interior. This interaction metaphor is directly inspired by CSG interfaces [96, 130] ; and the implementation we choose to render the coloring effect is related to similar ideas in CSG rendering [339].

II.1.2 Manipulating strokes

The question of how to facilitate conversation between human and machine "through the medium of line drawings" from Sutherland's thesis remains open to this day and has inspired many researchers to look into how *hand-drawn strokes* can be interpreted by a computer for diverse applications. We are in particular interested in how strokes can be used for the purpose of 3D content authoring, and in this section we present previous work on using strokes to define 3D geometry.

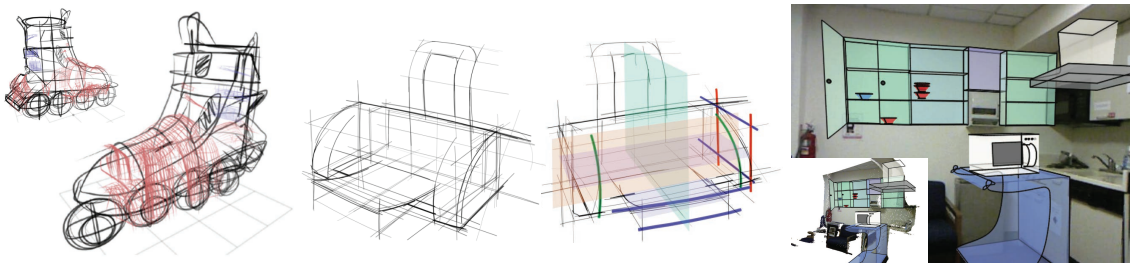
2D strokes to 3D strokes

A simple pen drawing as in Fig. I.2(a) is readily perceived as a 3D sphere by a human observer. Perhaps because of how simple that process feels to us, a large body of research called "sketch-based modeling" has studied how computers could be programed to understand 2D line drawings as 3D shapes [42, 232]. This line of work considers 2D line drawings as representations of 3D shape that have been projected to some image plane. The goal of sketch-based modeling systems is to inverse that projection such that from an input 2D stroke we obtain a 3D stroke, or such that from an input sketch we obtain a 3D surface. There is inherent ambiguity in this process since the "depth" of a stroke in the virtual 3D space before projection is unknown. There are multiple strategies developed in prior work to better constrain the problem, anchored in observations on how people draw [65, 331].

Some work has considered the design of interactive systems to facilitate converting 2D strokes into 3D strokes. In this setting, the depth ambiguity can be resolved by designing workflows to let the user specify it explicitly, for example by letting them provide multiple views of the same stroke [23, 161], sketch by projection on a 3D plane [23, 85, 196, 334], or on a transient 3D surface [24, 77, 172, 173, 231] (see Fig. II.4a). To reduce the need for multiple changes of viewpoints while sketching and to allow users to sketch more complex non-planar curves, Schmidt et al. [271] and Krs et al. [179] rely on geometric priors such as minimizing total curvature and on the spatial relationships between a stroke and its surrounding 3D context. They formulate such priors as terms in an optimization, and spatial relationships as hard constraints.

In the case where only one sketch from a single viewpoint is provided by the user, Xu et al. [333] showed that if the sketch follows industrial design sketching principles, formulating these same principles as objectives that the final 3D sketch should minimize makes it possible to reconstruct a complex 3D sketch. Recently, by using similar design principles and expressing them in robust optimization frameworks, Gryaditskaya et al. [120] and Hähnlein et al. [124] have shown that it is possible to deal with more complex sketches with messy oversketched strokes [121] (see Fig. II.4b).

Some work in this field has studied how to leverage real-world 3D context to lift 2D sketches to 3D [196, 236] (see Fig. II.4c). We are interested in this particular case in



(a) Sketching on transient surfaces [Kim et al. 2018]

(b) Lifting a single-view sketch to 3D [Hähnlein et al. 2022]

(c) Sketching in a 3D context [Li et al. 2017]

Fig. II.4: Creating 3D strokes via a 2D input device can be done by letting users draw on transient 3D curved surfaces (a) [172] ; or by reconstructing a plausible 3D interpretation of the 2D strokes by analyzing intersections and symmetries (b) [124]. To design in a real-world 3D context (c), partial 3D geometry reconstructed from a RGB-D capture can be used to place strokes in 3D [196].

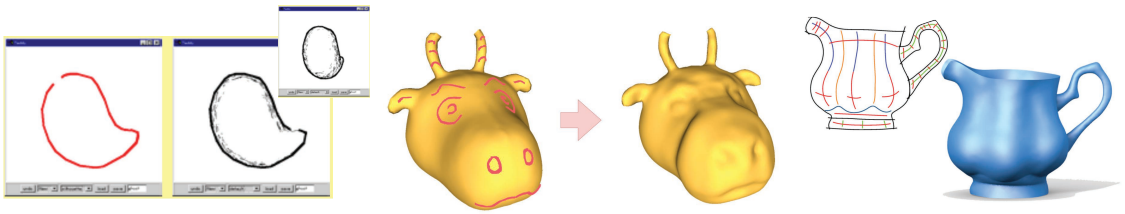
Chapter VI, where our context consists of a casually captured video, and our goal is to help the user place animated 2D doodles in this 3D context.

2D strokes to 3D surfaces

To let users create 3D surfaces by sketching, a simple and effective idea was first developed in *Teddy* [142] based on the assumption that a closed 2D curve represents the silhouette of a smooth round surface (Fig. II.5a). This idea was pushed further by Nealen et al. [226] that introduced more robust optimization framework to solve for the surfaces, and others explored how implicit surfaces could be used as surface representation in this problem [73, 162, 272]. While these methods typically rely on the user iteratively sketching multiple 2D contours from different viewpoints, others developed methods that work with a single sketch from one viewpoint by leveraging annotations [94, 114, 159, 189] (see Fig. II.5c), or by applying learnt priors from synthetic data [190, 208] (see Fig. II.5b). In the first core chapter of this thesis (Chapter III), we are inspired by the premise of sketch-based modeling and seek to explore a closely related question: if a user could sketch 3D strokes, how could those be used to create a 3D surface? While such a sketch is already well defined in 3D space, it shares some of the characteristics of the 2D sketches studied in sketch-based modeling, such as being a partial depiction of the surface, presenting oversketching, imprecisions and gaps between strokes.

Mid-air 3D sketching

Despite the recent burst of interest in virtual reality (VR) hardware, the core ideas of VR user interfaces have been developed by researchers for decades. In 1968, Sutherland [301] demonstrated the possibility of a head-mounted display that renders a virtual 3D scene from the viewpoint of the spectator's head. In the 90s, more practical hardware



(a) Contour inflation [Igarashi et al. 1999]

(b) Surface detailing with normal map prediction [Luo et al. 2021]

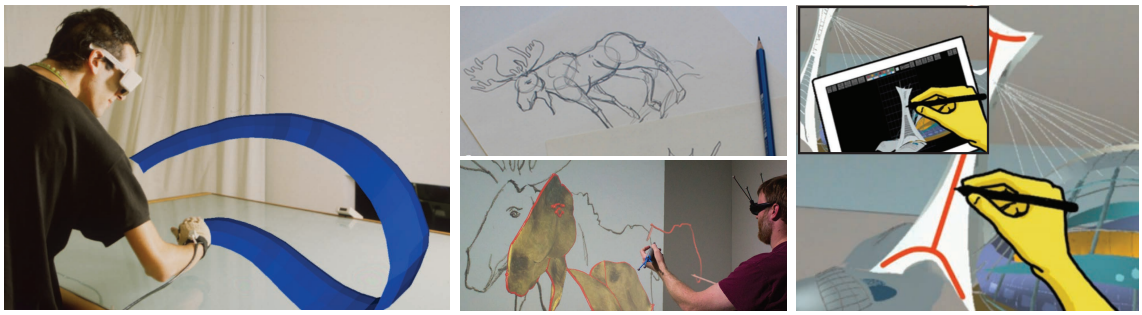
(c) Annotated strokes define curvature [Li et al. 2017]

Fig. II.5: Sketch-based modeling research investigates how to let users create and edit 3D shapes by sketching in 2D. A closed 2D curve can be inflated to a smooth surface (a) [142] ; a 2D pre-trained normal map estimation network can help interpret detailing 2D strokes as 3D edits (b) [208] ; and annotated 2D strokes can convey rich information about local surface curvature (c) [189].

systems for VR and AR such as the *CAVE* [69] and the *Responsive Workbench* [180] supported researchers in exploring the use of head-mounted display and tracked hand-held controllers for authoring applications. *HoloSketch* proposed to use 3D position input from such a hand-held device to create primitive objects or free-form tubes that follow the hand motion of the user [79]. *Surface Drawing* enabled users to create and deform surfaces in mid-air with a sweep of the hand [267] (see Fig. II.6a). In *FreeDrawer*, the 3D model was created on the *Responsive Workbench* by sketching a sparse network of 3D strokes, from which the system inferred surfaces that could later be deformed [322]. The ideas from these seminal papers are still very relevant to modern VR sketching applications. OpenBrush [140] and many other VR creative applications [228, 287, 289] employ a mid-air sketching interaction similar to the ideas presented in *HoloSketch* and *Surface Drawing*.

Israel et al. [148] reported that 3D sketching with tracked hand-held controllers could be useful for designers, by helping them externalize ideas and foster creativity. However, VR sketching interfaces present new challenges to users compared to 2D sketching. When sketching simple strokes like a straight line or a circle, users – especially novices in 3D sketching – achieve poor accuracy when trying to match a target [19, 211]. The lack of a supporting surface induces the need for better motor control in a space with more degrees of freedom than traditional 2D sketching. Adding to that, low spatial ability can inhibit users, making them more prone to errors when positioning strokes relative to each other. We refer the interested reader to a recently published book chapter that covers more thoroughly the opportunities and challenges of 3D sketching [21].

Motivated by those studies on the challenge of *geometric accuracy* when trying to achieve a particular shape with mid-air drawing gestures, a variety of interfaces have been proposed. One idea is to reduce the degrees of freedom when drawing a stroke mid-air. *3-Draw* [263] did so by decoupling the act of drawing the curve to define its shape from



(a) Painting mid-air [Schkolne et al. 2001]

(b) Lifting 2D strokes to 3D [Jackson and Keefe 2016]

(c) Combining 3D mid-air and 2D pen input [Arora et al. 2018]

Fig. II.6: Creating 3D strokes can be done using hardware with 6 degrees-of-freedom tracking. The motion of the user's hand mid-air can create a ribbon (a) [267] ; the user can grab and lift strokes drawn in 2D to 3D (b) [150] ; or 3D strokes can be created in two steps, by first using 3D mid-air sketching to define 3D canvases, then sketching on a canvas with a pen tablet (c) [18].

indicating its position and orientation in the overall drawing. Keefe et al. [166] use haptic feedback from a Phantom device and a 2-hand interaction metaphor inspired by tape drawing [26] to separate drawing the curve from indicating its tangent direction. Other methods avoid direct 3D freehand sketching in the creation process. Jackson and Keefe [150] proposed to use curves from a 2D sketch as a basis for VR creation (see Fig. II.6b), a feature also available in the commercial software Gravity Sketch [287] along with other curve editing operations based on control points.

Another line of work proposes to let artists sketch on a supporting surface. Arora et al. [18] and Drey et al. [87] use a 2D tablet on which the artist can sketch precise strokes that are mapped to a proxy 3D surface defined by a few freehand 3D strokes (see Fig. II.6c), and Jiang et al. [157] use the artist's non-dominant hand as a transient physical support for sketching. In a mobile AR context, using the phone as both a 3D tracked input device and a tactile screen to draw on can help make the best of both input conditions [182]. In an augmented reality context, physical 3D objects can also serve as support to sketch more accurately [318, 336], or to provide tactile feedback similar to sketching on paper [101, 187]. In lack of a good support surface, vibrotactile or pneumatic feedback can also provide some help in understanding pen position with respect to virtual surfaces [95].

Inspired by ideas from the domain of 2D sketch beautification [102, 141, 240], some work explored how beautifying, snapping or generally regularizing user's 3D strokes could help draw regular shapes [74, 210], curve networks [337], and complex 3D curves with precise control over positions and tangent directions [338]. In Chapter III and Chapter V, we propose methods that are resilient to imprecisions in 3D sketches, while avoiding sketch modification behaviors. We preserve the freedom of unconstrained freehand sketching by assuming from the get go that input sketches in our surfacing and coloring applications might be imprecise.

Finally, recent work proposed to overcome the need for geometric accuracy while sketching by instead considering the 3D sketch as a query into a learnt distributions of 3D shapes [205, 206]. This allows retrieving the closest 3D shape to the sketch or generating new shapes in this distribution that resemble the sketch, while being robust to ambiguity and imprecisions of sketches.

II.2 Appearance authoring

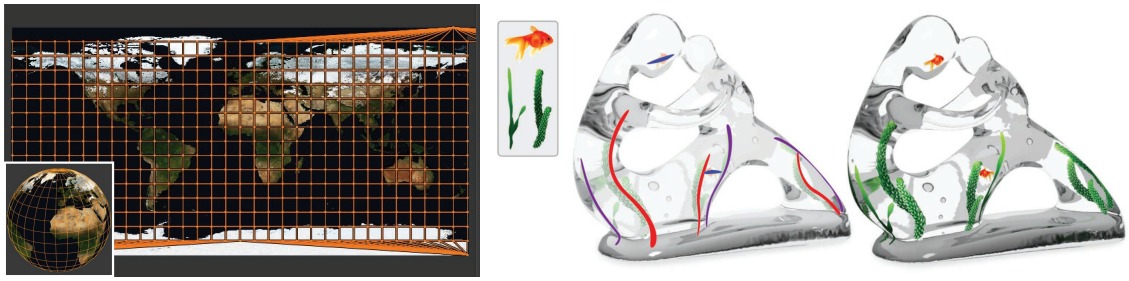
Appearance authoring is yet again a vast topic in computer graphics. We provide a scoped overview of appearance authoring by focusing on methods that give a high level of *direct control* over the final visual appearance. Since our work focuses on using 3D strokes to create appearance, we are de facto more interested in stylized or painterly appearance, which are styles that benefit most from providing the artist control via strokes. For example, we forgo discussing physically-based material models [84], or material capture techniques, as those do not give the opportunity to the artist to control appearance via mark-making with strokes. In the following section, we first give an overview over methods to paint on the surface of a 3D shape, then we look at methods to paint brush strokes that float mid-air in 3D.

II.2.1 Painting on a 3D shape

Since most objects are fully opaque and that we visualize them solely from their outside, they can be modeled as 3D surfaces with appearance described everywhere on this surface. A 3D surface is "in a certain sense, two-dimensional" [82], and for us that means that a straightforward way to describe appearance on this surface is to consider defining appearance on the familiar 2D plane – as if drawing on a canvas – and then find a valid mapping from each point of the 3D surface to the 2D plane. Most models in computer animation and computer games are defined in this way – artists paint on a 2D texture, and they define a mapping function between the 3D surface and the 2D texture space.

From 3D surface to texture

Finding a mapping from a 3D surface to 2D texture space is not an easy task in the general case. 3D surfaces with non-zero Gaussian curvature cannot be mapped to the plane without introducing distortions, just as one cannot wrap a soccer ball with gift wrapping paper without introducing creases and bunched up areas. 3D surfaces such as spheres that are not simply connected two-dimensional manifolds with boundary also need to be cut in some way to ensure that they can be flattened out. In other terms, the soccer ball gift wrapping must present at least one seam. Finding a good mapping for surface texturing thus requires multiple considerations, such as choosing appropriate cuts and defining how the surface patches are flattened out – a process called parametrization, in order to minimize visual artifacts of stretching or discontinuities once the texture is



(a) Global texture mapping in Blender

(b) Local texture mapping along sketched strokes [Sun et al. 2013]

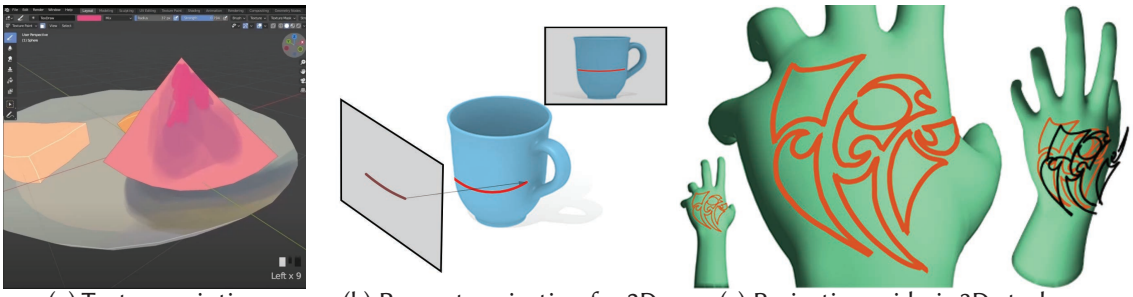
Fig. II.7: We can paint on a 3D shape by finding out how to map 2D paint strokes onto the surface. This can be done by finding a global texture mapping (a), that defines where each point of the sphere (inset) reads its color on the 2D planisphere [40] ; or by finding local mappings along sketched strokes (b) [298].

mapped to the 3D surface.

Previous work has proposed effective solutions to create mappings with low distortion. This can be done by introducing seams to reduce areas of high curvature, while taking care to hide the seams in less visible parts [279] ; by alternating user edits with automatic optimization based on a geometric distortion measure [248] ; or by automatically optimizing for both parametrization and cut placements [193]. Commercial software such as Blender [40] and ZBrush [216] provide implementations of some algorithms for texture mapping – also referred to as UV unwrapping. They also provide a direct manipulation interface where artists can edit how polygons from their 3D models are laid out in 2D (see Fig. II.7a). Because it is challenging to mentally predict how a change in 2D texture space affects the final look of the texture on the 3D surface, Gingold et al. [115] propose a user interface that enables the user to edit texture maps while working on the 3D surface.

These texturing approaches create a global mapping from the 3D surface to a 2D texture space. Other approaches have considered instead creating local maps on the surface in order to decorate it with decals [241], just as one might put a small sticker onto a surface. Schmidt et al. [269] describe how this can be achieved efficiently by approximating the exponential map of the surface at a point in the discrete setting, and they show how decals can be used to achieve interactions similar to 2D image processing but on the surface – eg copy and pasting, moving and deforming decals. This approach can be extended to support generating exponential maps along a sketched curve on the surface [298], in order to let users paint long strokes on the surface which can be textured with images – eg to simulate painting on the surface with an ink brush (see Fig. II.7b).

Lastly, DeBry et al. [78] introduced a different texturing paradigm: texture can be stored in 3D space in a sparse adaptive octree, instead of being stored on a 2D plane. This entirely bypasses the problem of finding an appropriate texture mapping for a surface before painting on it. In Chapter V, we take inspiration from this vision by encoding color



(a) Texture painting in Blender

(b) Raycast projection for 2D texture painting

(c) Projecting mid-air 3D strokes to a 3D surface [Arora and Singh 2020]

Fig. II.8: Painting textures can be done by applying strokes directly on the 3D surface (a) [40, 125]. When painting with a 2D input device, strokes are mapped to the 3D surface by raycasting each point (b). However, when painting mid-air in VR (black strokes), special precautions must be taken to ensure the resulting strokes on the surface (orange) are smooth (c) [22]. Note: (a) ©Craft Reaper ; (b) figure from [22].

texture as 3D strokes. This enables us to support coloring extremely messy geometry composed of many strokes.

Painting textures

Assuming we have obtained a global texture mapping for a 3D surface, the question of how to let artists paint the texture remains. Since the texture is a 2D image, it is trivial to paint on it but that remains a rather indirect way to define the appearance of the 3D surface, since the mapping from 2D texture to 3D surface might introduce transformations and distortions. In 1990, Hanrahan and Haeberli [125] proposed instead to let artists paint on the 3D surface directly. Brush stroke points are mapped from 3D to 2D texture space, and the texture image is painted in this way (see Fig. II.8a). This simple and very effective method was used to create complex 3D environments with a painterly style in Tarzan [70] ; and it is still in use today, as many material and texture painting software uses this interaction metaphor of painting directly on the 3D surface [5]. Painting on a 3D surface using a 2D input device – pen tablet or mouse – introduces challenges: how to map the strokes painted by the user in image space to the 3D surface? Typically, this is done by sampling the stroke and raycasting every sample to the closest point on the surface [125] (see Fig. II.8b). But this can make it difficult for artists to paint a complex 3D shape with self-occlusions, and requires many viewpoint changes to paint the whole surface. Previous work has explored how to help artists sketch on complex 3D shapes with many self-occluding layers by resolving depth ambiguities [108] and how to automate camera placement in order to find disoccluded views for painting [234, 304].

Using a 3D input device such as a VR controller, it becomes possible to paint on a surface with 3D motion. Agrawala et al. [9] used a tracked stylus to let people paint a texture by moving the stylus in 3D around a physical version of the digital 3D model they are

working on. The tracked stylus applies paint to the digital surface in a small volume around its tip. When trying to paint with a hand-held device in 3D space without such a convenient physical support, people can suffer from poor geometric accuracy, which makes painting smooth lines on a surface rather difficult. Arora and Singh [22] proposed a new method to map mid-air 3D stroke point to on-surface points, while ensuring the sketched strokes are smooth (Fig. II.8c).

Finally, while hand-painted textures are beautiful and expressive, they cannot react to changes in lighting or viewing conditions – eg a painted highlight will not move when viewed from a different point of view. Previous work has proposed techniques that augment hand-authored shading strokes or shapes with computational control, enabling hatching shading strokes to follow a shaded region [160], or letting artists define how shading shapes should react to changes in lighting [244].

II.2.2 Painting in 3D space

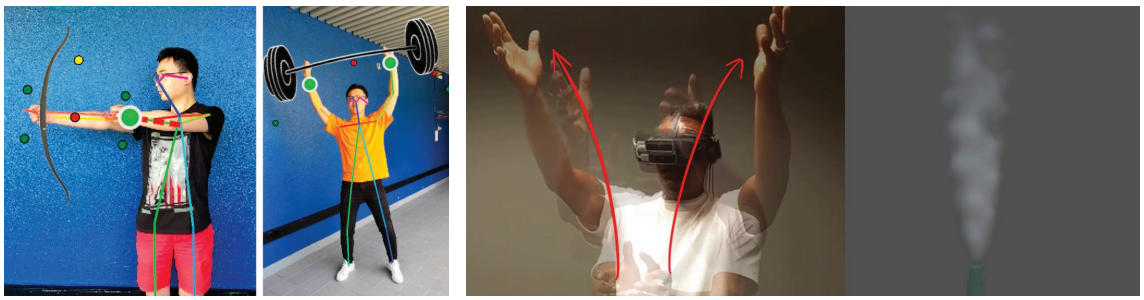
Limiting the application of paint to a pre-defined 3D surface limits the expressivity of painting, as it will always appear to be "stuck" to the underlying surface. *Overcoat* overcomes this by letting artists define 3D strokes in the vicinity of a base 3D surface [28, 268]. Strokes are defined in 3D space, and they are rendered with brush splats after being projected to image-space. This technique supports a painterly style with strokes that create a fuzzy volume around the 3D surface.

Painting on a surface by projection, like in "WYSIWYG texturing" [125], can also be used to create the impression of off-surface strokes by simply hiding the supporting surface in the final result. Only the painted strokes remain, and they appear to float in 3D space [36, 85, 144]. We use this technique in Chapter VI: 2D doodles are mapped to 3D by using the texture mapping from an invisible 3D canvas plane.

All previous techniques require the existence of a well-defined proxy surface to paint on, which means the user has to provide it before they can start painting. In VR painting, the artist can create 3D brush strokes mid-air directly, without an explicit support [140, 219, 228, 267, 289]. We study further the opportunities and challenges associated with this new medium in Chapter IV.

II.3 Motion authoring

We end our tour of digital 3D content authoring research by looking briefly at the topic of motion authoring. Supporting motion authoring or animation covers a vast spectrum of research, to cite just a few examples: how to pose a 3D character with a high level of control over the final shape [27, 76, 313], how to control large crowds [64], how to capture a live actor's performance [61], how to edit 2D graphics animation [341] or even how to make tangible stop-motion animation [2, 181]. Our work in this thesis dwells on



(a) Animating clip arts based on tracked pose [Liu et al. 2020]

(b) Controlling smoke emission with mid-air hand gestures [Arora et al. 2019]

Fig. II.9: Direct manipulation interfaces for animation can leverage captured motion content such as tracked human pose from a video (a) [201] ; or use the principle of authoring by demonstration, for example letting the user create and control effects with gestures of their hands (b) [20].

animation from the point of view of casual users with limited time and resources who would like to create short animated doodles (Chapter VI). Therefore we will focus in this section on previous work that looked at how to support similar practices: first, we survey techniques to author animation through direct manipulation, then we look at techniques catered to hand-drawn animation.

II.3.1 Direct manipulation interfaces for animation

The trajectory of a point or object through space is a characteristic component of motion. Previous work has leveraged this idea to help novices define animations by drawing a trajectory curve that controls the 2D translation and rotation of a 2D image along the curve [72]. Thorne et al. [310] considers sketched trajectories as being composed of a vocabulary of gestures that can be interpreted as different character animation segments to be synthesized. With control over a 3D character through a detailed skeleton, the character itself can be deformed to match the shape of a space-time trajectory, to follow a curved trajectory closely [122]. Motion trajectories were also used by previous work as a way to navigate videos [86, 117, 227], and simultaneously navigate and animate properties of objects [265]. In Chapter VI, we provide a visualization of an animated doodle's trajectory through 3D space as a way to help the user visually understand the animation and quickly spot and fix errors in our tracking results – the user can click on the trajectory to scrub the video in time.

Another line of work on animation tools for novice users leverages already captured content to drive an animation. Motion capture data or the video of a person acting a motion can be abstracted to a skeleton animation, which can be used to drive the deformation of a sketch that shares a similar structure [288, 295], or to add clip arts of virtual objects onto the video that follow the person's motion [201] (see Fig. II.9a). Our

work (Chapter VI) follows this idea to create complex 3D motion trajectories by analyzing scene motion from a video.

Finally, a very simple way to author animation is by demonstration – just like a child "animating" their plush toy by grabbing it and making it move. Previous work has used video capture of paper cutouts to create 2D animations [29], hand gestures to control 3D particle effects [20] (Fig. II.9b), recorded 3D controller motion to make strokes move [289], or a tracked tablet to create prototypes of augmented reality in-situ animations [188].

II.3.2 Animating sketches

Sketches are easy to create for novice users or children, so they are often used as a starting point in animation systems targeted at novices. Animating a sketch can be done at the sketch-level: by creating a suitable triangulation, Smith et al. [288] show that they can make a child drawing animate by deforming the triangulated sketch with ARAP [143]. Su et al. [295] follow a similar process to animate doodles, with the addition of a stroke-rigidity preservation term to the ARAP formulation. Sketched elements can also be animated procedurally, for example by cloning example sketched elements and applying simple procedures like emission from a source, or an oscillating effect [165].

A higher level of control over the look of an animation can be achieved by drawing more than one sketch. Indeed, many iconic hand-drawn animation effects cannot be reproduced by deforming a single sketch, such as when the shape of a character's head changes drastically between a frontal and side view. At an extreme, it is possible to draw every single frame of an animation. Since this is quite time-consuming, previous work has proposed methods to reduce the amount of frames to be drawn to obtain the full animation. This can be done by generating interpolating frames in-between sketched keyframes [8, 99], or by predicting how other frames should be modified given new strokes in one frame [332]. Lastly, one idea is to consider strokes to be 3D or "2.5D", since they are typically depicting the projection of a 3D object. Rivers et al. [257] propose to solve the problem of interpolating keyframes of a cartoon character animation by combining the effects of point interpolation in 3D and stroke interpolation in 2D. This is achieved by associating each stroke with a 3D position. A similar idea is to place strokes on a 3D canvas, and to define keyframes both in terms of canvas 3D shape and position, and in terms of strokes [110]. We take inspiration from this idea in Chapter VI, where we show that introducing the notion of a 3D canvas animated in 3D space simplifies the creation of animated doodles, making it possible to create complex animation effects by drawing just a few frames.

Chapter III

VR Sketching For 3D Surface Modeling

III.1 Motivation

As discussed earlier in Chapter I, sketching in 3D can serve as a way for artists to convey their intent for 3D shape authoring tasks. Creating 3D sketches composed of sparse strokes (Fig. III.1a) is a quick and inexpensive way to externalize and reflect on an idea, which can be helpful to designers for exploratory phases of design [148, 255].

However, 3D sketches are still only a partial representation of a 3D shape, and just like 2D design sketches they can be hard to parse visually due to the clutter created by occlusion when viewing the sketch as a 2D projection [120]. Communicating a design to a wider audience and other downstream tasks such as physical prototyping require designers to create a well-defined 3D surface model (Fig. III.1e). This is typically done with 3D modeling tools where users gradually create and refine polygonal meshes (Fig. III.1(b-d)). While the 3D sketch can be used as visual guidance or as a snapping target to align polygon vertices to 3D strokes, this process still involves a significant amount of manual work. Having the option to quickly convert a sparse 3D sketch into a well-defined 3D

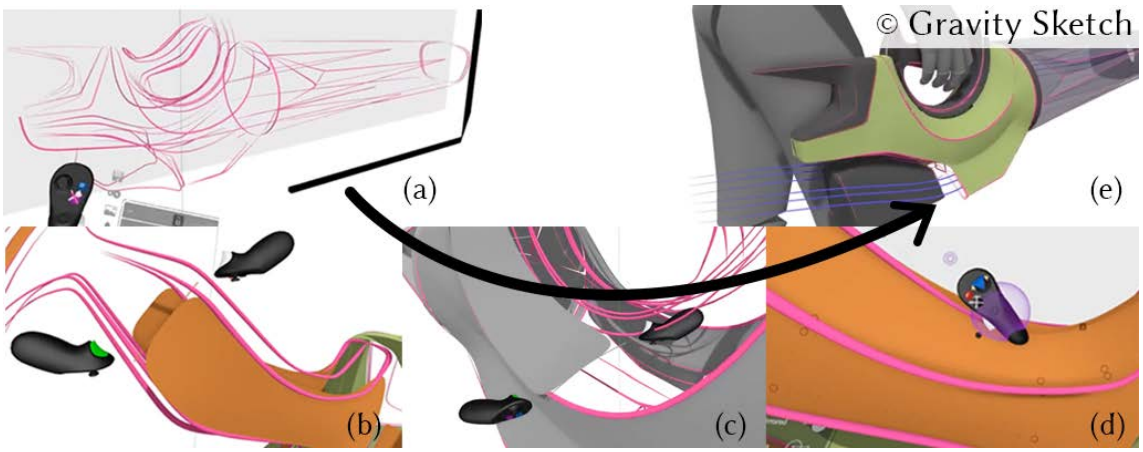


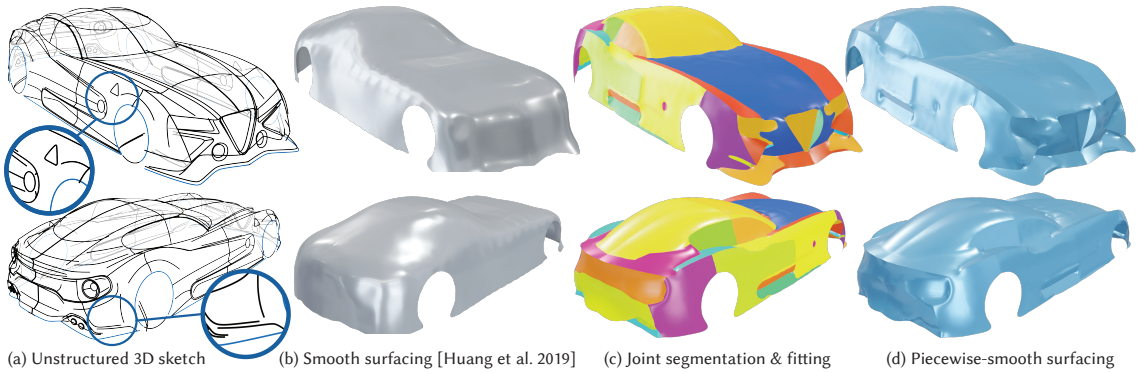
Fig. III.1: Sparse 3D sketches are a quick and inexpensive way for artists to externalize an idea (a), before eventually working on modeling a well-defined surface model (b-d), which can be used for downstream applications such as visualizing the design in-context to evaluate it (e). ©Gravity Sketch, used with permission.

surface can enable novel workflows with a quick back and forth between loose sketching and detailed 3D surface modeling.

III.2 Piecewise-smooth surface fitting onto unstructured 3D sketches

In this project, we make a first step towards this potential workflow by proposing, to our knowledge, the first approach to transform such 3D sketches into piece-wise smooth 3D surface models that preserve salient sketched features (Fig. III.2). We design our method based on several key characteristics of 3D sketches, highlighted as insets in Fig. III.2(a).

- Artists often depict piecewise-smooth surfaces by strategically placing strokes at sharp surface discontinuities, consistent with traditional sketching practices of representing an object through its feature curves – ridges and valleys [65, 121].
- While the main patches that form the envisioned surface are often delimited by strokes, other strokes lie within individual patches to depict sub-parts and decorative details.
- 3D sketches are often imprecise, exhibiting over-sketching or gaps and missing intersections between strokes.



(a) Unstructured 3D sketch (b) Smooth surfacing [Huang et al. 2019] (c) Joint segmentation & fitting (d) Piecewise-smooth surfacing

Fig. III.2: 3D sketches created in Virtual Reality (VR) or with sketch-based modeling systems often depict piecewise-smooth surfaces (a), but lack proper inter-stroke connectivity to detect the individual surface patches, as illustrated in the insets where pen strokes do not intersect precisely, and where detail strokes lie on the imaginary surface without being connected to other strokes. State-of-the-art surfacing algorithms only produce smooth surfaces from such sparse and unstructured 3D data (b). Our algorithm segments such an initial smooth surface into regions aligned with the pen strokes to produce a piecewise-smooth surface that better captures the intended shape. ©James Robbins, used with permission.

While the relative importance of these characteristics in 3D sketches may vary by artist and the drawing tool used, they underlie both AR/VR sketching and traditional sketch-based 3D modeling systems [23, 173, 333], making our approach well-suited in general to surfacing any sketches, that are provided as an unstructured collection of 3D curves.

Existing surfacing methods only partly account for these characteristics. Point cloud surfacing algorithms target densely sampled surfaces and fail on sparse stroke clouds [132, 164] or cannot capture sharp features of the sketch (Fig. III.2b).

Methods dedicated to 3D drawings focus on well-connected curve networks [1, 34, 233, 264, 349], where surface patches are bounded by closed cycles of curve segments. Due to their strong requirements on curve network topology, such methods cannot process the imprecise, unstructured stroke clouds we target.

To reconstruct piecewise-smooth surfaces from inaccurate 3D sketches, we must jointly determine which parts of the sketch correspond to different surface patches, and recover the geometry of these patches away from the strokes.

A first challenge is to determine where surface patches should lie in the empty space between the strokes. This task is particularly ambiguous for sparse sketches, since the inter-stroke Euclidean proximity does not necessarily denote geodesic proximity on the envisioned surface. We reduce this ambiguity by complementing the input sketch with a low-fidelity *proxy surface*, which we obtain by applying the smooth point cloud surfacing method of Huang et al. [137], or in cases where this automatic method fails, by asking users to create a simple proxy with low-poly modeling tools (see Section III.6). The proxy surface provides us with a manifold domain on which we project nearby strokes, such that our problem of locating the surface patches amounts to *segmenting* the proxy into regions roughly bounded by strokes.

Our second challenge is to shape each proxy region into a surface patch that represents well the geometry of the strokes that project in that region, which is especially difficult when the strokes themselves are sparse and approximate. We address this challenge by representing each patch as an implicit surface defined as the zero level-set of a low degree polynomial – which we refer to as an *implicit polynomial surface*. Such implicit surfaces offer multiple benefits in our context. They have infinite support, allowing the surface regions they capture to grow or shrink arbitrarily as the algorithm progresses, and they are fast to evaluate and fit to stroke points.

Equipped with the concepts of a proxy surface and implicit polynomial surface patches, we cast finding the piecewise-smooth surface that best represents the sketch as a *multi-model fitting* problem [147]. In a nutshell, our algorithm alternates between refining a segmentation of the proxy into regions well represented by a set of implicit surfaces, and improving the implicit surfaces by fitting them over the strokes within each region. After convergence, we generate the final 3D triangle mesh by projecting the proxy surface

onto the zero-level sets of the implicit polynomial surfaces, effectively recovering sharp features at the intersections between patches.

While our method automatically produces piecewise-smooth surfaces aligned with the input strokes, it also supports user indications to further improve the quality of the surface; for instance, to force the creation or removal of surface discontinuities. We also rely on user indications to discard strokes that are not supported by our approach, notably internal strokes that lie inside the outermost depicted surface, small disconnected parts, and so-called *skeletal strokes* that depict thin cylindrical features (see Fig. III.18).

In summary, we present the first surfacing method for 3D sketches that produces piecewise-smooth meshes while being oblivious to stroke connectivity. We evaluate our method by surfacing more than 50 sketches created with a variety of VR and sketch-based modeling systems, by comparing to surfacing methods tailored to well-connected curve networks, by measuring deviation from ground truth surfaces from which synthetic sketches were generated, and by studying the impact of the main components of our algorithm. Finally, we provide an interactive visualization of all our results as supplemental materials for close inspection.

III.3 Related work

Surfacing curve networks The problem of surfacing 3D sketches emerged with the advent of practical user interfaces, algorithms and devices to create such sketches. Early 2D interfaces deduce the depth of the strokes from the intersections they form with other strokes [271, 333] or with sketching planes [23], effectively producing well-connected *curve networks* by construction. As a consequence, a number of surfacing algorithms strongly rely on the connectivity of the curve network to identify closed cycles delimiting surface patches [1, 233, 264, 349]. Each such patch can then be surfaced by propagating geometric information from the boundary curves [34, 237, 293]. Curve connectivity information provides strong geometric hints, since surface normals can be estimated at each intersection – as done by Pan et al. [237] to detect sharp features and determine which curves are flow lines.

However, the reliance on accurate curve network topology prevents these methods to work on raw, *unstructured stroke clouds* as typically produced via freehand VR sketching [140, 287, 289].

Recent studies have shown that precise sketching in VR is more challenging than in 2D due to the lack of a supporting surface for the hand and the need for finer motor control to position strokes in 3D [19, 211]. In a preliminary work to this thesis, we prototyped a VR sketching system that automatically neatens the 3D sketch as the user is sketching to form a well-connected curve network despite inaccuracies [337], and other work also looks at correcting user's strokes to enforce intersections or geometric regularity [210, 338]. However, we found in our user study for *CASSIE* [337] that people sometimes

felt constrained in what they could achieve by being forced to think of their designs in terms of a curve network.

Processing 3D sketches to form well-connected curve networks is both challenging and often undesirable. First, many strokes in 3D sketches are not meant to connect to others – see the details on *author1_bulbasaur* (Fig. III.19), and on the car (Fig. III.2 insets). Simply ignoring such disconnected strokes can alter the original design intent (Fig. III.15, red strokes). Secondly, 3D sketches may exhibit oversketching and imprecisions (*vr_controller* Fig. III.19), which make forming clean curve networks from 3D strokes a challenging problem, akin to 2D vectorization. We instead propose an approach that is oblivious to stroke connectivity, achieving high robustness to inaccuracy and to detail strokes that lie on the envisioned surface but are not connected to other strokes.

Our work also differs from interactive systems designed to model 3D surfaces by sketching in 2D [93, 94, 189, 226]. Users of these systems draw in dedicated interfaces and provide annotations of surface discontinuities. In contrast, we take as input 3D sketches created with a variety of tools, and we propose a multi-model fitting algorithm to automatically locate sharp features where and only where they are needed.

Surfacing point clouds The 3D sketches we consider can easily be converted to point clouds by sampling points along each stroke, which would allow leveraging the wealth of methods developed for surfacing such unstructured 3D data [33]. Unfortunately, most existing methods have been designed to process dense point clouds acquired using 3D scanning technology [132, 164], and are doomed to fail on 3D sketches that only provide a very sparse, non-uniform sampling of the envisioned surface. While some methods are robust to missing data, they only fix holes that are relatively small compared to the scale of the overall surface [133], or guide surface completion by fitting geometric primitives on otherwise dense parts of the point cloud [273, 303]. In contrast, 3D sketches are dominated by large holes, and only contain 3D information along thin, 1-dimensional subspaces.

The recent *VIPSS* algorithm of Huang et al. [137] achieves impressive resilience to sparsity and non-uniformity of the point cloud, and has even been demonstrated on unstructured stroke clouds similar to our target sketches. But this robustness is obtained thanks to a global smoothness energy that misses sharp surface discontinuities. Nevertheless, we use *VIPSS* to initialize our method, and focus on the problem of segmenting its result into individual patches forming a more faithful reconstruction of the input sketch.

Our approach relates to algorithms that recover piecewise-smooth surfaces from point clouds by identifying locally-smooth patches and by detecting sharp discontinuities as the intersections of these patches [104, 136, 156]. However, these methods rely on a dense sampling of the surface *away* from sharp features, while 3D sketches are mostly empty in such regions and are only densely sampled *along* feature curves.

We also take inspiration from methods that reconstruct or approximate 3D shapes as a collection of parametric geometric primitives, such as planes [63, 220, 245], cones, cylinders and spheres [195, 326], generalized cylinders [348], and quadric surfaces [335]. Similarly to these methods, we cast the discovery of representative surface patches as a multi-model fitting problem. Our originality is to leverage specifics of 3D sketches to guide this optimization, notably by encouraging large, uniform patches bounded by the input strokes, and by representing free-form surfaces as 4th-order implicit polynomial surfaces.

Note that the resulting segmentation is purely driven by geometry and does not carry any semantic meaning other than when semantic parts happen to be well-represented by different smooth patches – eg, the hood of the car and the windshield (Fig. III.2c).

Surfacing stroke clouds Our work focuses on *sparse* 3D sketches, which contrast with the *dense* VR paintings targeted by Rosales et al. [261] that are created by covering the surface with large, overlapping ribbons that provide position and orientation samples all over the surface [140, 260].

In the computer vision community, a few methods have been proposed to reconstruct *curve clouds* by matching edges in a multi-view stereo algorithm [100]. Usumezbas et al. [312] proposed a surfacing algorithm dedicated to such unstructured 3D data, where candidate surface patches are lofted between pairs of curves. But this method largely relies on the availability of multiple photographs of the shape to select valid candidate patches based on occlusion reasoning.

Closest to our problem statement, Batuhan Arisoy et al. [30] surface sparse and imprecise 3D sketches by smoothly deforming an initial low-fidelity surface of correct topology, using a discrete guidance vector field that points towards the closest stroke point. This approach produces globally smooth surfaces and requires user intervention to specify strokes that should be inserted into the mesh as sharp edge polylines (see Fig. 17 and 18 in their paper for a visual comparison of their results on similar sketches to ours). In contrast, our multi-model fitting formulation produces piecewise-smooth surfaces automatically.

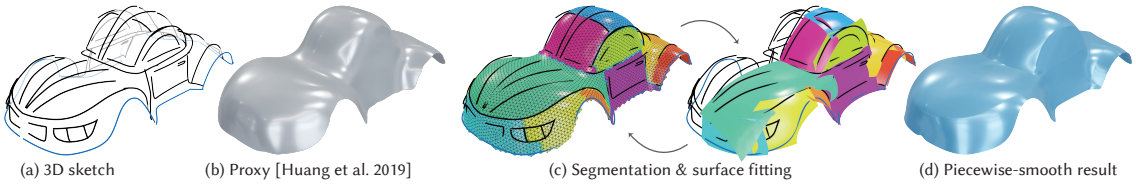


Fig. III.3: *Overview of our method.* We take as input a 3D sketch (a) and a proxy mesh (b), for example obtained with *VIPSS* [137]. We iteratively improve a segmentation of the proxy mesh (c, left) and parameters of a set of surface models (c, right) to obtain a decomposition of the surface into smooth patches that approximate the input strokes well. We represent each surface model as the zero level set of an implicit polynomial, which offers infinite spatial support and a good balance between expressivity and complexity. Finally, we project the vertices of the proxy mesh onto the surface models to recover the final piecewise-smooth surface with sharp features (d). When appropriate, we trim the proxy mesh to match user-annotated boundary strokes (a, blue), and we leverage sketch symmetry by surfacing only half of the sketch and mirroring the result.

III.4 Overview

Fig. III.3 illustrates the main steps of our method, which takes as input a 3D sketch, along with an approximate *proxy surface* obtained with an automatic surfacing algorithm [137] or an existing low-poly modeling tool (see the accompanying video for a demonstration of this workflow). After projecting the sketch onto the proxy, our goal is to segment the proxy into regions, each associated with a smooth *surface model*, such that the resulting piecewise-smooth surface satisfies the following desiderata.

- **Reproducing stroke geometry.** The surface models should run close to the input strokes, both for strokes that depict sharp surface discontinuities and for strokes that depict details within smooth areas.
- **Aligning patch boundaries with strokes.** The boundary between neighboring regions should lie along strokes that depict sharp surface discontinuities, yet not all strokes in the sketch depict a discontinuity.
- **Keeping the reconstruction simple.** The surface should be composed of a small number of smooth patches, rather than many intricate patches that would overfit to inaccuracy in the input strokes.

We formulate these competing requirements as energy terms in an optimization. A first energy term measures the distance between each stroke and the surface model it is assigned to. A second term measures the smoothness of the segmentation away from the strokes, encouraging the transitions between models to occur along strokes. This smoothness term also penalizes small, isolated regions, which contributes to satisfy our third desiderata. Finally, a third term measures the complexity of the reconstruction by counting the degrees of freedom of the models used.

While each of our terms has an intuitive interpretation, their combination yields a challenging optimization problem that combines discrete variables (which surface model should be assigned to which region of the proxy) and continuous variables (parameters of these models). We tackle this challenge by expressing our problem within *PEARL*, a general algorithm to solve multi-model fitting problems [147]. In our context, the algorithm alternates between optimizing the discrete variables describing the segmentation while keeping the model parameters fixed, and optimizing for the continuous model parameters while keeping the segmentation fixed (Fig. III.3c).

After convergence, we obtain a segmented proxy, as well as a set of surface models represented by implicit polynomial surfaces. The last step of our method aims at converting this representation into a triangle mesh suitable for downstream applications (Fig. III.3d). Since the proxy provides us with a mesh of correct topology relatively close to the surface models, we recover the final surface by projecting this mesh onto the zero level-sets of the polynomials.

III.5 Method

We now define the variables and the energy terms of our optimization problem, before describing how we solve this problem by alternating segmentation and fitting. In what follows, we use the terms *points* and *segments* to refer to the stroke polylines, and the terms *vertices* and *edges* to refer to the triangles of the proxy mesh.

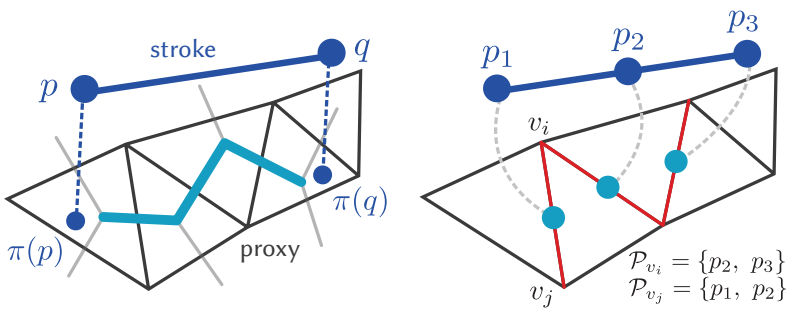
III.5.1 Encoding stroke information on the proxy domain

While our segmentation algorithm relies on the proxy mesh as a computational domain, several of our energy terms are defined as functions of the stroke points. To reason about stroke geometry on the proxy domain, we project each stroke to its nearest location on the proxy and associate it with nearby mesh elements.

Specifically, we start by projecting each stroke point to its closest face on the proxy mesh. We then trace out stroke segments as geodesics lying on the mesh surface. Note that we do not need to trace precise geodesics on the mesh, since we are only interested in detecting which mesh edges are crossed by a given stroke segment. Therefore, we approximate the geodesics with shortest paths in the dual graph of the mesh (light blue path in Fig. III.4a). This gives us the list of primal mesh edges that are crossed by the projected stroke segment.

We can then associate stroke points to these crossed edges (Fig. III.4, red) by sampling additional points on the stroke segment (Fig. III.4, blue). If n edges are crossed while tracing a segment, we sample n regularly-spaced points on that segment. Each of these points is then associated to the corresponding mesh edge.

Finally, we associate to each vertex v_i the stroke points associated with each edge of its

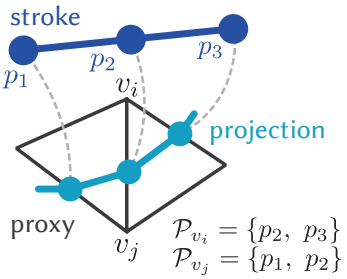


(a) Projection (b) Association

Fig. III.4: Associating stroke information to the proxy graph. (a) Stroke points are projected onto the mesh and we find the shortest path (light blue) between the mesh faces on the dual graph of the mesh (in grey). (b) The shortest path yields a list of edges crossed by the projected segment (red). We associate stroke points sampled from the stroke segment with neighboring vertices of each crossed edge.

1-ring. If there are multiple stroke points associated to a single edge, we only associate the closest point to v_i.

Thanks to this association, our segmentation algorithm assigns one surface model per mesh vertex, yet can fit several surface models to a single stroke point – which is critical for positioning boundaries of surface patches along strokes. Denoting $\mathcal{P}$ the set of all stroke points, we denote $\mathcal{P}_v \subset \mathcal{P}$ the subset associated with a given mesh vertex $v \in \mathcal{V}$, which can contain zero or many stroke points (see inset).



We also leverage the association of stroke points to mesh edges by assigning a weight w^e to each edge $e \in \mathcal{E}$ of the mesh, using a low weight $w^e = 1$ if the edge is crossed by a projected stroke segment (red edges in Fig. III.4b), and a high weight $w^e = 100$ otherwise. The values given here we are in case all edges have approximately the same length. In practice, we introduce mesh simplification to reduce our algorithm's runtime (Section III.6), and use a scaling term to adjust edge weights to edge lengths in the case of anisotropic meshes.

III.5.2 Problem formulation

Our goal is to assign each mesh vertex to a surface model f_α, associated with the label $\alpha \in \mathcal{L}$. We denote the corresponding vertex labeling as $l\colon \mathcal{V} \to \mathcal{L}$. Furthermore, we denote as θ_α the vector of real-valued parameters of the surface model f_α. Our surface model representation, detailed in Section III.5.3, is the zero level-set $\mathcal{Z}(f_\alpha)$ of an implicit polynomial. Finally, we denote as $\Theta = \bigoplus_{\alpha\in\mathcal{L}} \theta_\alpha$ the concatenation of the parameter vectors θ_α for all $\alpha \in \mathcal{L}$.

Given these definitions, we cast our problem as finding the labeling and the associated model parameters that minimize

$$\begin{aligned} E_{\mathcal{L}}(l,\Theta) = & E_{\text{fidelity}}(l,\Theta) \\ & + w_{\text{smoothness}} \cdot E_{\text{smoothness}}(l) \\ & + w_{\text{simplicity}} \cdot E_{\text{simplicity}}(l), \end{aligned} \tag{III.1}$$

where E_{fidelity}, $E_{\text{smoothness}}$, and $E_{\text{simplicity}}$ capture the three desiderata listed in Section III.4. Note that the number of labels required for a given sketch is also unknown, since we do not know how many smooth patches are necessary to faithfully represent the 3D surface a priori. We next describe each of the three energy terms.

Fidelity to input strokes We seek a piecewise-smooth surface that reproduces well the input stroke geometry. We measure this property by summing over all mesh vertices v the distance between their surface model $\mathcal{Z}\left(f_{l(v)}\right)$ and each of their associated stroke points $\mathrm{p} \in \mathcal{P}_v$:

$$E_{\text{fidelity}}(l,\Theta) = \frac{1}{\epsilon^2 N_p} \sum_{v \in \mathcal{V}} \sum_{\mathrm{p} \in \mathcal{P}_v} dist\left(\mathcal{Z}\left(f_{l(v)}\right), \mathrm{p}\right)^2, \tag{III.2}$$

where the normalization term N_p is the total number of stroke points involved in the summation. We detail in Section III.5.3 how to compute $dist\left(\mathcal{Z}(f), \mathrm{p}\right)^2$ efficiently. The scale factor ϵ controls the sensitivity of E_{fidelity} to small deviations of the strokes from the surface models. We experimentally set this parameter to 1% of the input sketch's bounding-box diagonal for all results presented in the paper, but we show the impact of alternate values in Fig. III.12.

Smoothness of the labeling Our smoothness term seeks to concentrate changes of labels along strokes, such that sharp surface discontinuities appear at these locations when transitioning from one surface model to another. Inspired by classic work on edge-aware image segmentation [49], we encourage neighboring vertices to share the same label, unless they are separated by a stroke. We achieve this goal by leveraging edge weights of the proxy mesh as penalties: each change of label along an edge incurs as penalty the weight of the edge, which we define based on whether that edge is crossed by a stroke segment or not (see Section III.5.1). Specifically, we define the smoothness energy:

$$E_{\text{smoothness}}(l) = \frac{1}{W} \sum_{\{u,v\} \in \mathcal{E}} w^{uv} \left(1 - \delta\left(l(u), l(v)\right)\right), \tag{III.3}$$

with δ being the Kronecker delta function, $\delta(i,j) = 1$ if $i = j$, $\delta(i,j) = 0$ otherwise. The normalization term W is the sum of all edge weights. We set $w_{\text{smoothness}} = 10$.

Surface simplicity To be resilient to the typical inaccuracy of 3D sketches [19], we encourage the surface reconstruction to be as simple as possible. We measure the complexity of a solution as the total number of surface parameters involved. Denoting $\dim(\theta_\alpha)$ the number of parameters of a surface model f_α, we define the simplicity energy as:

$$E_{\text{simplicity}}(l) = \frac{1}{D} \sum_{\alpha \in \{l(v) | v \in \mathcal{V}\}} \dim(\theta_\alpha), \tag{III.4}$$

where we set $D = 35$, the number of parameters of a degree 4 model, as a normalizer. Note that by summing over assigned models only, this energy also pushes for using a small number of models to explain the 3D sketch, to the point where a single model can be used to represent multiple non-adjacent regions of the surface. For all results, we set $w_{\text{simplicity}} = 0.01$ (and demonstrate varying values in Fig. III.11).

III.5.3 Energy minimization

Algorithm 1 adapts the general *PEARL* multi-model fitting algorithm [147] to estimate the labeling l and surface model parameters Θ that locally minimize Equation III.1. After an initialization phase, the algorithm alternates between improving the labeling and refining the model parameters, and terminates when the labeling l no longer changes. We next detail each step.

Segmentation. At each iteration, we first optimize the current labeling l while keeping all surface model parameters Θ fixed. We perform α-expansion [48] – which finds the optimal change of labeling where some nodes are assigned a label α – for every label $\alpha \in \mathcal{L}$ to efficiently find a local minimum of $E_{\mathcal{L}}$ [1]. While $E_{\text{smoothness}}$ and $E_{\text{simplicity}}$ are straightforward to compute for a given labeling l, E_{fidelity} requires computing the distance $dist\,(\mathcal{Z}(f), \mathrm{p})^2$ between each surface model and the corresponding stroke points. Since the exact distance between a point and the zero level-set of an implicit polynomial surface cannot be computed exactly with a direct method, we employ the first-order approximation proposed by Taubin [308],

$$dist\,(\mathcal{Z}(f), \mathrm{p})^2 \approx \frac{f(\mathrm{p})^2}{\|\nabla f(\mathrm{p})\|^2}, \tag{III.5}$$

which can be computed in linear time with respect to the number of stroke points p. Since the resulting labeling might leave some labels unassigned, we update $\mathcal{L}$ to only keep the selected models.

[1] We use the C++ multi-label optimization library of Delong et al. [80] https://vision.cs.uwaterloo.ca/code/

Algorithm 1: PEARL algorithm [147] applied to our problem

(0) Initialize variables

Propose m vectors of parameters $\{\theta_1^0, \cdots, \theta_m^0\}$ as initial surface models

$\Theta \longleftarrow \bigoplus_{i=1}^{m} \theta_i^0$ // Concatenated surface parameters

$\mathcal{L} \longleftarrow \{1, \ldots, m\}$ // Set of existing labels

$l \longleftarrow l\colon v \in \mathcal{V} \mapsto 1$ // Initialize with single label

repeat

  $l_{\text{prev}} \longleftarrow l$

  (1) Optimize labeling

  $l \longleftarrow \alpha$-expansion [48] $\forall \alpha \in \mathcal{L}$ to optimize previous labeling l_{prev} for energy (Equation III.1), given the current parametric surface models Θ

  $\mathcal{L} \longleftarrow \{\alpha = l(v) \mid v \in \mathcal{V}\}$ // Remove unassigned labels

  $\Theta \longleftarrow \bigoplus_{\alpha \in \mathcal{L}} \theta_\alpha$ // And the corresponding inactive models

  (2) Optimize parametric surface models

  for $\alpha \in \mathcal{L}$ **do**

    $\mathcal{V}_\alpha \longleftarrow \{v \in \mathcal{V} \mid l(v) = \alpha\}$ // Vertices with label α

    $\theta_\alpha \longleftarrow$ Best fit for points in $\mathcal{P}_\alpha = \bigcup_{v \in \mathcal{V}_\alpha} \mathcal{P}_v$

  end

  (3) Propose new models

  Propose new labels $\mathcal{L}_{\text{new}}$ and models Θ_{new}

  $\mathcal{L} \longleftarrow \mathcal{L} \cup \mathcal{L}_{\text{new}}$

  $\Theta \longleftarrow \Theta \cup \Theta_{\text{new}}$

until $l = l_{\text{prev}}$;

Model representation and fitting. After each segmentation step, we improve the surface model of each label $\alpha \in \mathcal{L}$ by optimizing its parameters θ_α to best fit the stroke points $\mathrm{p} \in \mathcal{P}_\alpha$ associated with the vertices $v \in \mathcal{V}_\alpha$. We define each surface model as the zero level-set of a polynomial $f\colon \mathbb{R}^3 \to \mathbb{R}$,

$$\mathcal{Z}(f) = \{(x, y, z) \mid f(\theta_\alpha; x, y, z) = 0\},$$

with f expressed as

$$f(\theta_\alpha; x, y, z) = \mathrm{X}(x, y, z)^T \theta_\alpha,$$

where $\theta_\alpha = \left[\theta^1 \ldots \theta^t\right]^T$ is the $t \times 1$ vector of coefficients and $\mathrm{X}(x, y, z)$ a $t \times 1$ vector of monomials,

$$\mathrm{X}(x, y, z) = \begin{bmatrix} 1 & x & y & z & x^2 & \ldots & x^d & y^d & z^d \end{bmatrix}^T.$$

To avoid overfitting to approximate strokes, we limit the expressivity of the surface models by keeping their degree d low. In practice we set d to be at most 4, which corresponds to $t = 35$ parameters. We describe at the end of this section how we introduce lower-degree models at the end of each iteration. Fig. III.5 shows how surface models of different degrees capture shapes of varying complexity.

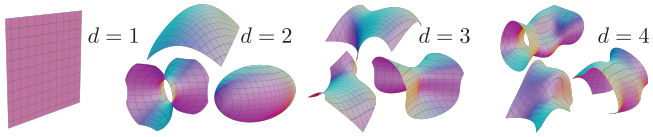


Fig. III.5: We represent surface patches as zero level-sets of implicit polynomials of degree up to 4. Low-degree polynomials can represent planes, developable and simple doubly-curved surfaces, while higher-degree polynomials can represent complex freeform surfaces.

Given the set of stroke points $\mathrm{p} = (x, y, z) \in \mathcal{P}_\alpha$, we seek the vector of coefficients θ_α that minimize the geometric error between the stroke points and the zero level set of the function f. However, the geometric distance from a point to the zero level-set of a polynomial has no closed form expression, and minimizing it requires an iterative approach [10]. Similarly, minimizing Equation III.5 is costly since its derivatives are non-linear with respect to the model parameters θ. A common alternative consists in minimizing the *algebraic distance* to the implicit polynomial surface, which yields a linear least-squares regression:

$$\underset{\theta}{\operatorname{argmin}} \sum_{\mathrm{p} \in \mathcal{P}_\alpha} \left(\mathrm{X}(\mathrm{p})^T \theta \right)^2 .$$

As shown by Tasdizen et al. [307], this minimization problem can be made more stable by also encouraging the gradient of the implicit function to align with prescribed normals at a set of points $\mathcal{P}'_\alpha$:

$$\underset{\theta}{\operatorname{argmin}} \sum_{\mathrm{p} \in \mathcal{P}_\alpha} \left(\mathrm{X}(\mathrm{p})^T \theta \right)^2 + \mu \sum_{\mathrm{p} \in \mathcal{P}'_\alpha} \left(1 - \mathrm{n}_\mathrm{p} \cdot \nabla X(\mathrm{p})^T \theta \right)^2 ,$$

where $\nabla X(\mathrm{p})$ is a $t \times 3$ matrix denoting the gradient of the monomial vector $X(\mathrm{p})$, and n_p denotes the normal vector at the point p. Since we do not have normal information at the stroke points, we encourage alignment with the proxy mesh normals at the mesh vertices. Furthermore, we exclude normals from vertices that lie close to the strokes, since strokes often denote surface discontinuities for which the smooth proxy mesh is only a crude approximation. We thus define the set of points for the alignment term as $\mathcal{P}'_\alpha = \{v \in \mathcal{V}_\alpha \mid \mathcal{P}_v = \emptyset\}$. Finally, we also include an L^2 regularization term to penalize large polynomial coefficients [307], yielding:

$$\hat{\theta}_\alpha = \underset{\theta}{\operatorname{argmin}} \sum_{\mathrm{p} \in \mathcal{P}_\alpha} (\mathrm{X}(\mathrm{p})^T \theta)^2 + \mu \sum_{\mathrm{p} \in \mathcal{P}'_\alpha} (1 - \mathrm{n}_\mathrm{p} \cdot \nabla X(\mathrm{p})^T \theta)^2 + \lambda \left\| \theta \right\|^2 , \tag{III.6}$$

with $\mu = 0.1$ and $\lambda = 1$. Fig. III.6 illustrates the impact of each of these regularization terms.

From Equation III.6, we obtain the parameters $\hat{\theta}_\alpha$ that best fit the stroke points by solving the linear system

$$(M^T M + \mu M_{\text{align}}^T M_{\text{align}} + \lambda I)\ \hat{\theta}_\alpha = \mu M_{\text{align}}^T . \tag{III.7}$$

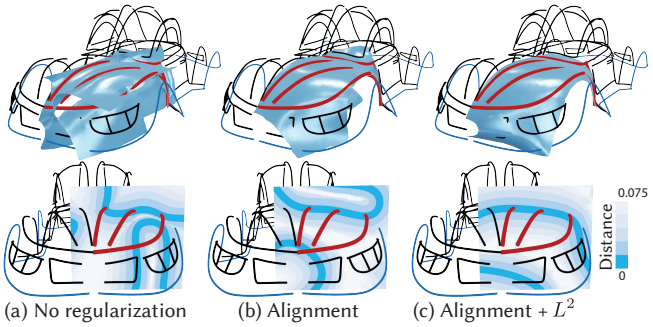


Fig. III.6: Fitting an implicit polynomial surface to a subset of strokes (thick red strokes). We visualize a slice of the unsigned distance field in the second row (distances consider a sketch bounding box diagonal of unit length). (a) Minimizing the algebraic distance without regularization yields level sets with spurious turns close to the data points. (b) By encouraging the gradient to align with the proxy normals [307], the implicit function is well behaved in a neighborhood around the data points. (c) Adding L^2 regularization avoids overfitting to noise in the data points and yields a more stable zero level-set.

M is a $(|\mathcal{P}_\alpha| \times t)$ matrix composed of rows of the monomial vector $X(\mathrm{p})$ for every point in $\mathcal{P}_\alpha$. And given the set of points $\mathcal{P}'_\alpha$ for the alignment term, M_{align} is a $(1 \times t)$ vector defined as:

$$M_{\text{align}} = \sum_{\mathrm{p} \in \mathcal{P}'_\alpha} \mathrm{n_p}^T \nabla \mathrm{X}(\mathrm{p}). \tag{III.8}$$

To keep the L^2 regularization position and scale independent, and resilient to non uniform scaling in the point set $\mathcal{P}_\alpha$, we apply a transformation to the space of monomials composed of a translation – to center the data – and a non uniform scaling – to normalize it. In practice, we compute the column-wise mean values μ and standard deviations S of M, and consider the transformed polynomial function:

$$f(\theta_\alpha; x, y, z) = \left(\frac{\mathrm{X}(x, y, z) - \mu}{S} \right)^T \hat{\theta_\alpha},$$

where the division by the vector S is a column-wise division. As a consequence, the vectors $X(\mathrm{p})$ that compose matrix M are transformed by subtracting μ and dividing by S, and the vector M_{align} is divided by S.

Note that $\hat{\theta}_\alpha$ does not necessarily decrease E_{fidelity} (Equation III.2) given a fixed labeling, since it minimizes the algebraic distance augmented with regularization terms – and not the geometric distance. To guarantee convergence to a local minimum of Algorithm 1, we only update the parameter vector θ_α if this yields a decrease of E_{fidelity} for the vertices labeled as α. If not, we keep the previous parameter vector and introduce the new parameter vector as a different model.

Initialization. Starting the algorithm with a good initial guess leads to higher quality solutions (see evaluation in Fig. III.10). We obtain this initial guess by leveraging the observation that some of the strokes depict boundaries of smooth surface patches. Assuming that this is the case for all strokes, we compute an over-segmentation of the proxy surface by running spectral clustering [280] on the mesh, using the same edge weight w^e as in $E_{\text{smoothness}}$. We then fit an implicit polynomial surface to the strokes associated to the vertices of each cluster to obtain our initial set of models.

Spectral clustering requires the number of clusters to be specified in advance. Yet, the appropriate number varies among sketches, depending of their level of details. We address this challenge by running spectral clustering with different numbers of clusters (in practice, 20, 30, 40, and 50 clusters), which produces surface models of different scales.

For this initial surface fitting, we boost the regularization weights μ and λ by a factor 10 to limit complexity of the models and prevent the appearance of sub-optimal large models across sharp features.

New model proposal We end each iteration by proposing new surface models, which helps decrease the energy $E_{\mathcal{L}}$ in subsequent iterations [147]. First, we introduce models of lower degree by fitting a polynomial of degree $d-1$ to the set of stroke points $\mathcal{P}_\alpha$ for each active model f_α of degree $d > 1$. Second, we propose new models by merging pairs of neighboring regions and fitting a polynomial to the union of their stroke points. We prioritize merging regions that are separated by edges with a high weight w^e, as this strategy is more likely to yield a decrease in $E_{\text{smoothness}}$. In practice, we select three pairs of regions to be merged per iteration.

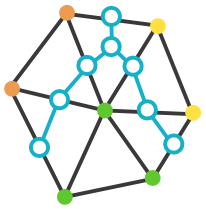
III.5.4 Extracting the surface mesh

The outcome of our multi-model fitting algorithm is a set of implicit surfaces that extend infinitely beyond the strokes they capture (Fig. III.3c). Our goal is now to extract a triangular mesh from this arrangement of surfaces. One option to reach this goal would be to trim each surface along its intersection with other surfaces, for example by meshing the isosurfaces, resolving intersections [57], and then selecting the set of patches that best cover the strokes while forming a closed manifold [31, 91]. Unfortunately, the halfspaces defined by our implicit polynomial surfaces are not guaranteed to bound the desired shape, as two surfaces that describe adjacent regions might not intersect. Even when neighboring surfaces do intersect, they can be nearly tangential to each other, which makes the detection of intersections prone to numerical inaccuracies. We bypass all these difficulties by leveraging the proxy mesh, as it provides us with a good estimate of the mesh we are looking for. We formulate our problem as the local minimization of an energy that projects the mesh towards the implicit surfaces assigned to each region,

regularized by a smoothness term:

$$E_{\text{mesh}} = E_{\text{project}} + E_{\text{regularize}} \tag{III.9}$$

Insertion of segmentation boundaries. Before attempting to minimize Equation III.9, we first need to insert edges that split the mesh triangles crossed by segmentation boundaries, so that these boundaries can emerge as sharp surface discontinuities in the optimized mesh. The inserted vertices (blue circles in inset) inherit the two or three labels of the regions they separate, yielding a multi-labeling function that we denote $\hat{l}$.



Projection to the implicit surfaces. We project each mesh vertex onto its associated surface patches by minimizing its distance to the zero-level sets of the corresponding polynomials:

$$E_{\text{project}}(\mathcal{V}) = \sum_{v \in \mathcal{V}} \sum_{\alpha \in \hat{l}(v)} \frac{1}{|\hat{l}(v)|} dist\left(\mathcal{Z}(f_\alpha), v\right)^2, \tag{III.10}$$

where we use the first-order approximation from Equation III.5 to compute the distance.

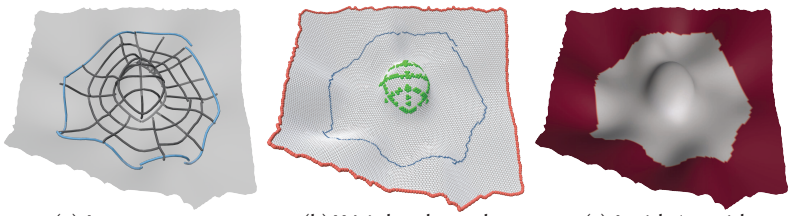
Regularization. Minimizing Equation III.10 sometimes pulls neighboring vertices in opposite directions, yielding a distorted surface. We achieve smoother results by regularizing the optimization with a 2D Laplacian term on the mesh triangles, and a 1D Laplacian to favor smooth sharp feature curves:

$$E_{\text{regularize}}(\mathcal{V}) = \gamma_{2D} \sum_{v \in \mathcal{V}} \mathrm{w}_v ||\Delta_{2D} v||^2 + \gamma_{1D} \sum_{v \in \mathcal{V}_{\text{seams}}} ||\Delta_{1D} v||^2, \tag{III.11}$$

where $\mathcal{V}_{\text{seams}}$ is the set of vertices inserted along segmentation boundaries, and Δ_{2D} and Δ_{1D} denote the discrete graph Laplacian operator for triangles and edges respectively [225]. We set $\gamma_{2D} = 100$ and $\gamma_{1D} = 500$.

The weight w_v controls the strength of the surface regularization, which we want to vanish along segmentation boundaries that correspond to sharp surface discontinuities. We set $\mathrm{w}_v = 1$ for vertices away from segmentation boundaries, and $\mathrm{w}_v = 0$ for boundary vertices that are shared by more than two segmentation regions. For vertices that lie in-between two regions, we adjust their weight according to the angle formed by the implicit surface normals on each side of the boundary, with $\mathrm{w}_v = 0$ when the angle is greater than $\pi/3$, $\mathrm{w}_v = 1$ when the angle is smaller than $\pi/8$, and w_v varies linearly in-between. Effectively, this preserves sharpness of the boundary between regions that intersect sharply, while favoring smoothness elsewhere.

While the Laplacian regularization in Equation III.11 favors smooth, regular meshes, it has the adversarial effect of shrinking open meshes. We prevent this effect by adding



(a) Input proxy & labeled strokes (b) Weighted graph & inside/outside nodes (c) Inside/outside labeling

Fig. III.7: To locate the boundary of an open surface, we ask the user to indicate which strokes correspond to boundaries in the input sketch (a, blue strokes). We perform a graph cut on the mesh to separate inside nodes (b, green) from outside nodes (b, red), while encouraging cuts to happen at edges crossed by boundary strokes (b, blue edges). The resulting labeling separates the inside of the surface (c, white) from the outside (c, red).

to E_{mesh} a term that constrains boundary vertices of open meshes to stay close to their associated stroke points

$$E_{\text{attract}} = \sum_{v \in \mathcal{V}_{\text{boundary}}} \sum_{\text{p} \in \mathcal{P}_v} ||v - \text{p}||^2 \tag{III.12}$$

where $\mathcal{V}_{\text{boundary}}$ is the set of vertices associated to the boundary strokes of the sketch.

However, we do not know a priori where an open surface should be trimmed to align with the sketch boundary. We thus ask users to annotate the boundary strokes of sketches depicting open surfaces (shown in blue in all sketches). After projecting these strokes over the proxy mesh, we segment the mesh into an interior and an exterior region separated by the boundary strokes (Fig. III.7c), and trim the exterior region to obtain a mesh whose boundary $\mathcal{V}_{\text{boundary}}$ aligns with the boundary strokes. We perform this segmentation with a graph cut, where we encourage cuts along edges crossed by boundary strokes by setting their cost to zero (Fig. III.7b). The graph cut source node is linked to all boundary vertices of the input mesh (red in Fig. III.7b) and the sink node is linked to vertices of the mesh that have associated stroke points but are geodesically far from the boundary strokes (green in Fig. III.7b).

For ease of comparison, we also apply this boundary cutting step to results of Huang et al. [137] against which we compare.

We minimize E_{mesh} with L-BFGS, by precomputing the gradient matrices for the quadratic terms.

III.6 Implementation details

Proxy creation Our method requires a proxy surface that has the same topology as the desired result, and that lies close to the envisioned surface, so that projecting the strokes onto the proxy yields the same embedding as it would on the envisioned surface.

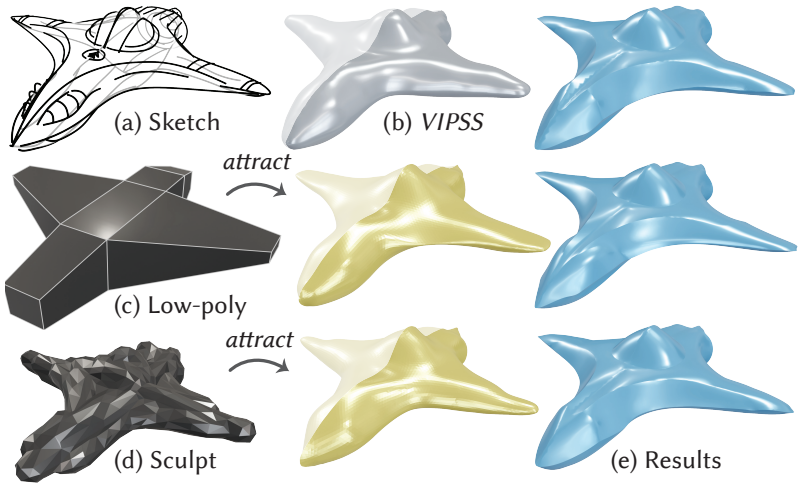


Fig. III.8: For the same sketch, we demonstrate how a proxy mesh can be created with *VIPSS* (b), low-poly modeling with *Blender* [38] (c), and rough VR sculpting in *Adobe Medium* [7] (d). We automatically improve the proxy mesh by remeshing, smoothing, and attracting it to the strokes (middle column). When the sketch is symmetric, we only process half of the proxy mesh. Note that despite a drastically different original appearance, all three proxy meshes yield a similar result (e), faithful to the input sketch (a).

The automatic *VIPSS* surfacing algorithm [137] produced suitable proxies for many of our results. We additionally re-meshed the output of *VIPSS* to obtain a high-resolution, uniform mesh [154]. However, *VIPSS* sometimes fails to surface complex sketches that exhibit many sharp features, a very sparse sampling, or thin features. In these cases, we experimented with multiple casual modeling tools to create our proxies in just a few minutes, including low-poly modeling with *Blender* [38] (Fig. III.8c), rough VR sculpting with *Adobe Medium* [7] (Fig. III.8d), assembly of simple geometric primitives (Fig. III.20 *author1_bulbasaur*). We provide an example modeling session in our accompanying video.

Manually-created proxy meshes are often approximate, and would not yield satisfying results if projected directly on the implicit surface models (Section III.5.4). We obtain better results by first attracting the proxy towards the strokes, which we achieve by projecting the stroke points onto the proxy and by using these points as control vertices for a *Least-Squares mesh* defined by the initial proxy connectivity [292]. Fig. III.8 shows how this attraction brings approximate proxy meshes much closer to the intended surface.

Symmetric sketches Many VR sketching and sketch-based modeling tools provide a mirror plane to ease the creation of symmetric sketches. When this is the case, we only surface half of the sketch and mirror the result, which speeds up computation and yields surfaces that are symmetric by construction.

User control Most results shown in this chapter were obtained automatically. Nevertheless, an additional strength of our formulation is that it admits user control naturally. Fig. III.9 illustrates two scribble-based controls that we offer users to refine the segmentation:

- *Inserting a new region.* Users can scribble over a part of the mesh to trigger the insertion of a new region in that part (Figure III.9, yellow scribbles). This interaction is particularly useful to recover details that might have been over-smoothed.
- *Merging regions.* Users can scribble over multiple regions to merge them into a single one (Figure III.9, green scribbles). This interaction can be used to correct for over-segmentation.

We implement region insertion by introducing a new model, which we fit to the stroke points associated with the scribbled-on mesh vertices. We implement region merging in a similar manner, except that we fit the new model to the stroke points associated with every vertex that shares a label with any scribbled-on vertex and lies on the same connected component. For both edits, we also penalize the use of other models by adding a term to Equation III.1:

$$E_{\text{penalty}}(l) = \rho \sum_{v \in \mathcal{V}_{\text{scribbled}}} 1 - \delta\left(l(v), l_{\text{new}}\right) \tag{III.13}$$

where $\mathcal{V}_{\text{scribbled}}$ denotes the set of scribbled vertices, l_{new} denotes the label of the new model, and the penalty ρ is set to $(10\epsilon)^2$, with ϵ from Section III.5.2. We then update the solution by re-running Algorithm 1 from step (1) until convergence.

This combination of new model proposals and a penalties on other models effectively pushes the optimization towards a different local minimum that satisfies the user indications. We describe an edge-collapse algorithm in Section III.6 that reduces the complexity of the mesh on which we compute the labeling, making our method react faster to user edits.

Graph simplification Algorithm 1 has a complexity that increases linearly with the number of nodes in the graph to be labeled [48]. Yet, we note that only the vertices that are associated to stroke points contribute to E_{fidelity}, other vertices being solely determined by the smoothness and simplicity terms. We leverage this observation to simplify the graph away from the strokes, which we achieve by performing greedy edge collapses [113, 132]. In our context, we are not interested in preserving the 3D shape of the original mesh, but rather its connectivity. Therefore, we prioritize collapses that yield vertices of low valence, and stop collapsing edges whenever any further collapse would yield a vertex of high valence (> 12 in practice). We exclude edges that have a vertex associated with stroke points to preserve high resolution near the strokes.

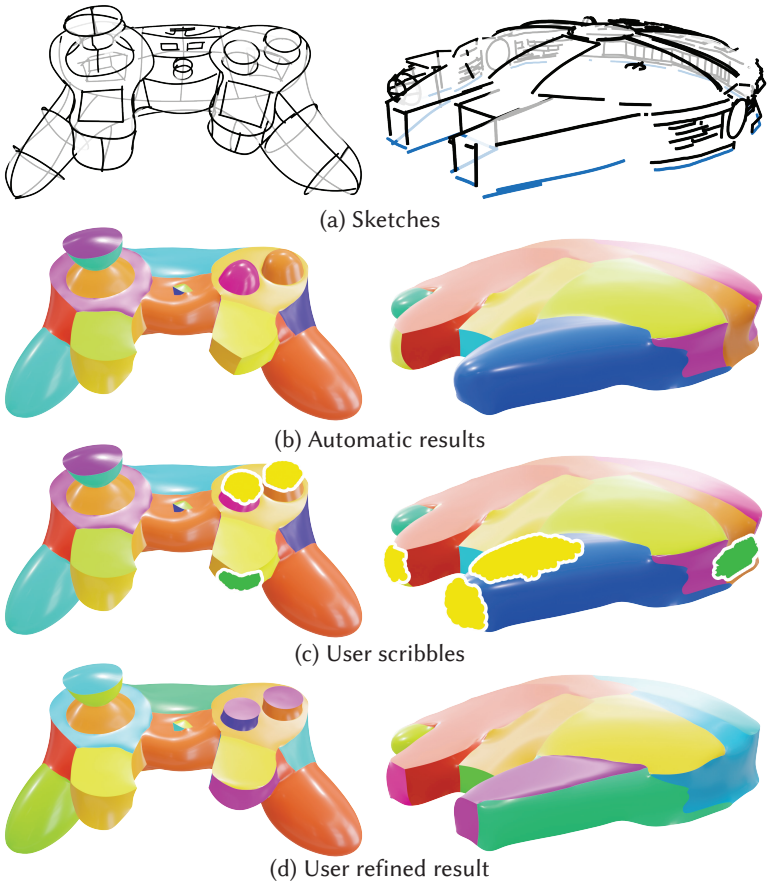


Fig. III.9: Our automatic method might produce over-segmented regions or miss geometric details (b). The user can indicate desired changes by scribbling on a preview of the result (c). Yellow scribbles trigger region insertion, green scribbles trigger region merging. Encouraging the segmentation to respect these constraints yields the desired result (d). Left column sketch ©Jacopo Colò, used with permission.

This simplification produces a mesh with spatially-varying resolution, which we account for by adjusting the edge weight to $\tilde{w}^e = w^e c^e$, where w^e is the original edge weight as defined in Section III.5.2, and c^e scales this weight according to the size of the triangles adjacent to the edge, as measured by the Euclidean distance between the vertices opposite the edge on the two adjacent faces.

After running Algorithm 1 on the simplified graph, we propagate the labeling to the vertices of the original graph based on proximity. This simple strategy speeds up Algorithm 1 by a factor of 2.9 on average, while yielding qualitatively similar results.

III.7 Evaluation and results

III.7.1 Algorithm evaluation

Initialization strategy

We initialize our method by over-segmenting the proxy mesh using spectral clustering (Section III.5.3). Fig. III.10 compares this strategy with two baselines, over multiple runs of the method. The first baseline strategy creates the initial surface models by sampling random sets of points among all stroke points in the sketch. The second baseline applies spectral clustering but keeps the default regularization weights rather than scaling their values during the initial surface fitting.

This evaluation reveals that our strategy is more stable, as it makes the optimization converge to approximately the same energy for different random seeds. In addition, our strategy is more effective, as evidenced by the lower or comparable median energy values at convergence. Finally, visualizing the segmentation produced by each strategy highlights the benefit of boosting the regularization when fitting the initial models, as it prevents the creation of complex models covering large regions of the sketch, which would be difficult to remove in subsequent steps of the optimization.

While our initialization strategies guarantees low variance and similar results regardless of random seed (see bottom two inset results Fig. III.10), for all subsequent results we reduce the influence of initialization by running our method 10 times and selecting the solution with the lowest energy.

Segmentation parameters

Fig. III.11 shows the effect of our simplicity term (Equation III.1) by varying its weight. The result is over-segmented when we omit the simplicity term altogether (Fig. III.11a). Increasing this parameter (Fig. III.11b, c) leads to smaller model counts m, and to models of lower degrees (see the spaceship cockpit), at the cost of an increase in mean fitting residual $\bar{\epsilon}$. The simplicity energy also encourages using the same model across disconnected regions (Fig. III.11c, yellow ellipsoid), which cannot be achieved with the smoothness energy alone.

The other tunable parameter of our method is the factor ϵ that defines the scale of the fitting error we tolerate (Equation III.2). Increasing this factor leads to results that follow the strokes more loosely, which can be useful if the input sketch is imprecise and needs to be smoothed (Fig. III.12).

Influence of proxy mesh

Fig. III.13 illustrates the influence of the proxy resolution and shape on our results. Since the proxy serves as a discrete domain for our segmentation algorithm, its resolution impacts the size of the regions we can detect. At low resolution, neighboring strokes will be lumped together and sharp features will be misplaced (Fig. III.13a, third row). In

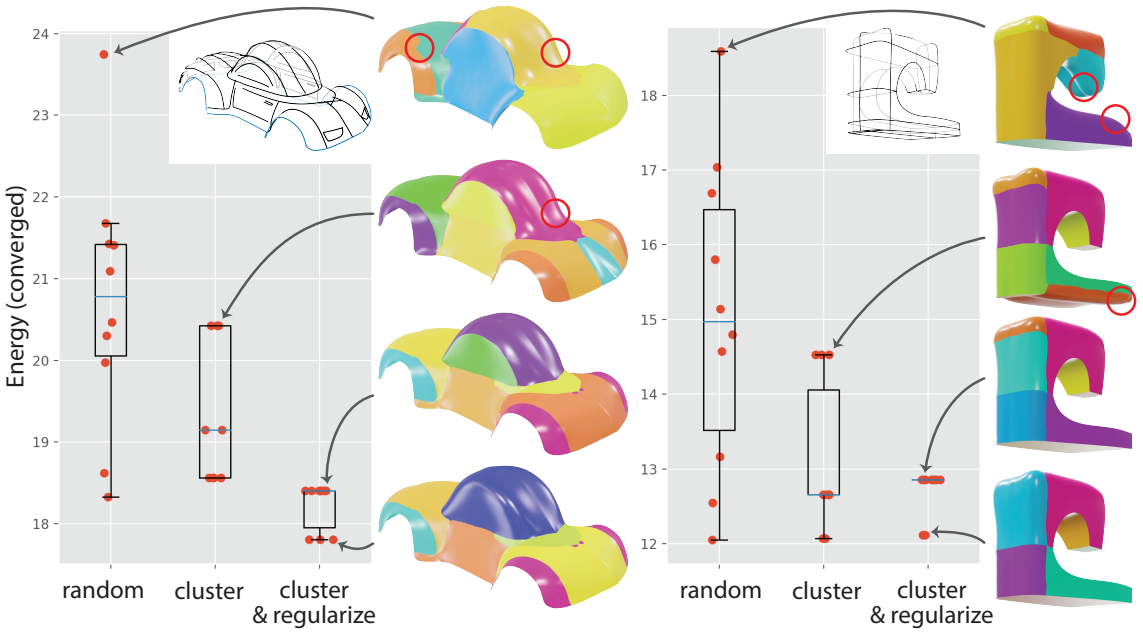


Fig. III.10: Comparison of different initialization strategies. Each plot shows the energy (Equation III.1) of the converged state over 10 runs with different random seeds. Initializing the surface models with a random selection of stroke points yields high variance across runs (left column of each plot). Spectral clustering identifies local patches that form good candidate models, but the default regularization weights used to fit these models can result in complex surfaces that cover large portions of the sketch (middle column of each plot, red circles in segmentation results). Boosting the regularization weights favors simpler models at the start of the optimization, yielding better and more stable results after convergence (right column of each plot).

addition, a low-resolution mesh might lack degrees of freedom to follow the curvature of the implicit surfaces, and is thus unable to reproduce the shape of smooth high curvature parts of the sketch (Fig. III.13a, fourth row).

The proxy also serves to embed the strokes into a manifold domain representative of the envisioned surface. When the proxy is too far from the desired surface, strokes that should be disjoint might end up projecting to the same location on the proxy and be approximated by the same surface patch (Fig. III.13b, top). A slightly more precise proxy mesh resolves the issue (Fig. III.13b, bottom).

III.7.2 Results and comparisons

Fig. III.19 and Fig. III.20 provide a gallery of results obtained by surfacing 3D sketches from varied sources, ranging from clean curve networks created with 2D sketch-based modeling interfaces [23, 333], to imprecise and over-sketched stroke clouds created with 2D and VR ideation tools [66, 173, 337]. All these results were obtained without user

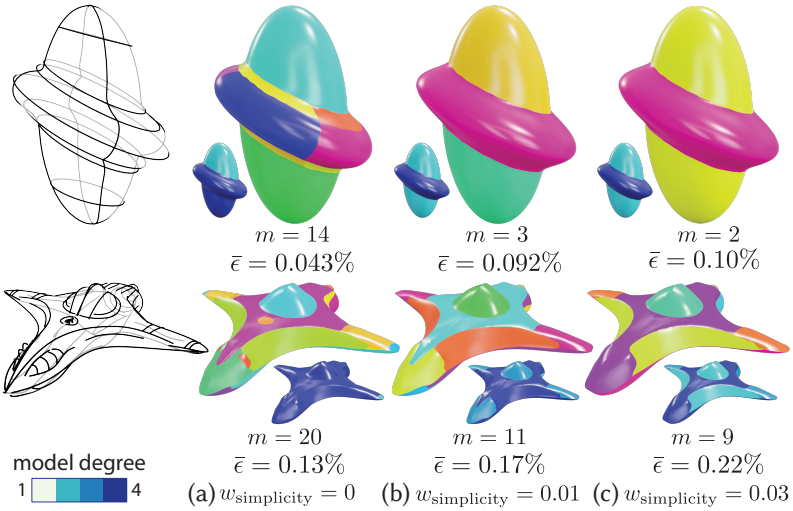


Fig. III.11: We demonstrate the impact of the simplicity energy by varying the weight $w_{\text{simplicity}}$ in Equation III.1 on two sketches. We measure the number m of models used in the segmentation, and the mean deviation $\bar{\epsilon}$ between stroke points and the models of the vertices they are associated with, as a % of the bounding box diagonal.

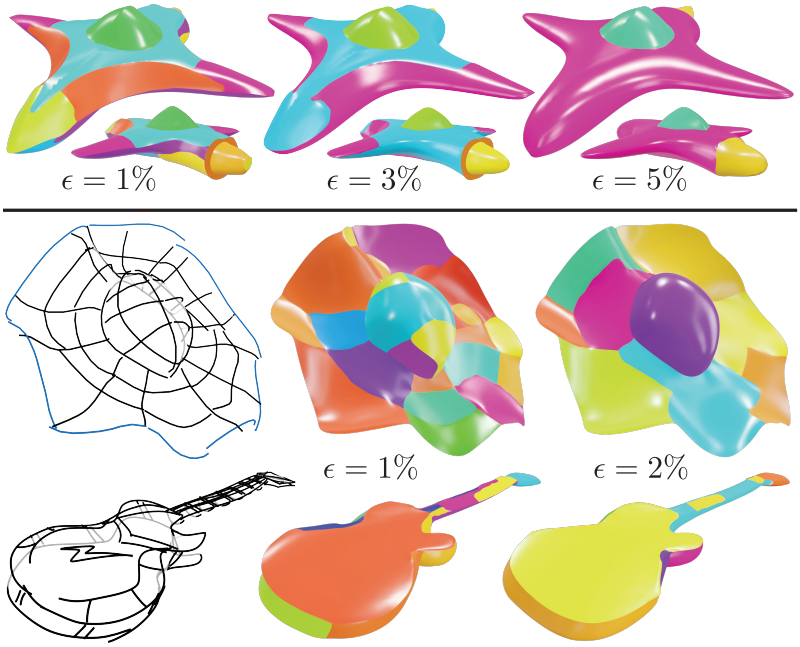


Fig. III.12: We demonstrate the impact of the fidelity energy by varying the error scale factor ϵ in Equation III.2 on the spaceship sketch (top row). In bottom rows, we show concrete examples of sketches with imprecise strokes where a higher ϵ gives better results compared to the default $\epsilon = 1\%$ parameter. The error scale factor is measured in % of the sketch bounding box diagonal.

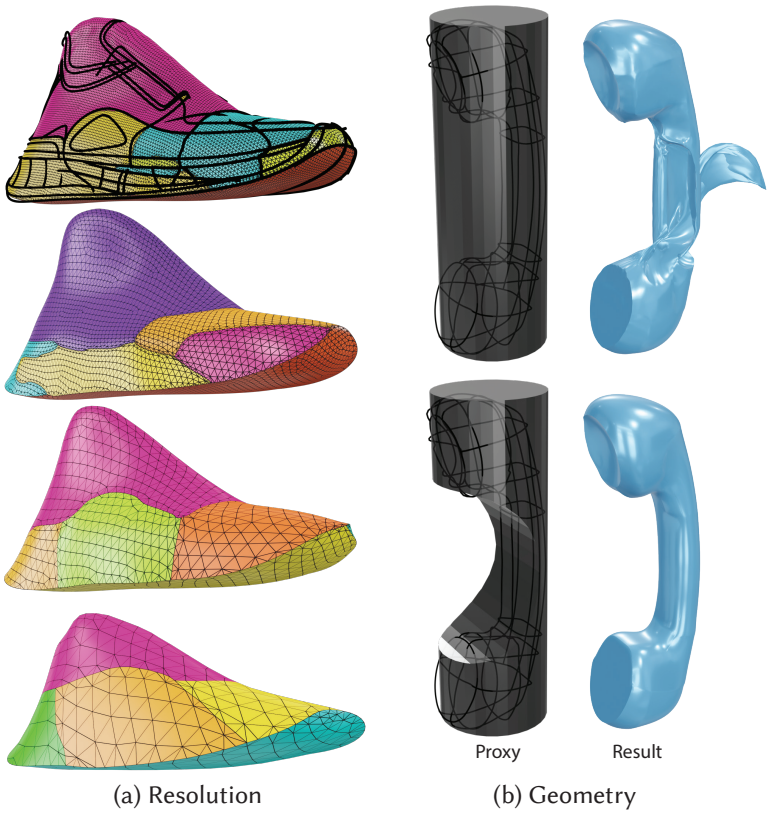


Fig. III.13: Our method depends on the resolution of the input proxy surface (a) as well as on its shape (b). The proxy should have sufficient resolution to capture fine details, and be close enough to the envisioned surface to avoid thin parts of the sketch to collapse. Left column sketch ©Arturo Tolentino, used with permission.

correction. We use a light gray material to shade proxy meshes computed automatically with *VIPSS* [137], and a dark gray material for the ones created manually. Note how our method recovers the fine details and sharp edges depicted in the sketch even from very smooth, approximate proxy meshes. Our supplementary website [2] displays each result with an interactive 3D viewer.

We detail in Table III.1 the performance of our algorithm on all results shown in the main paper. Timings vary from a few seconds on simple sketches up to a few minutes on complex sketches composed of many strokes. The bottlenecks reside in the initialization of the algorithm and in the final mesh optimization, while the segmentation algorithm takes less than 10 seconds in most cases. Users can thus refine the segmentation multiple times with relatively short wait times and only compute the final mesh once satisfied.

[2] see https://ns.inria.fr/d3/Surface3DSketch/results-page/

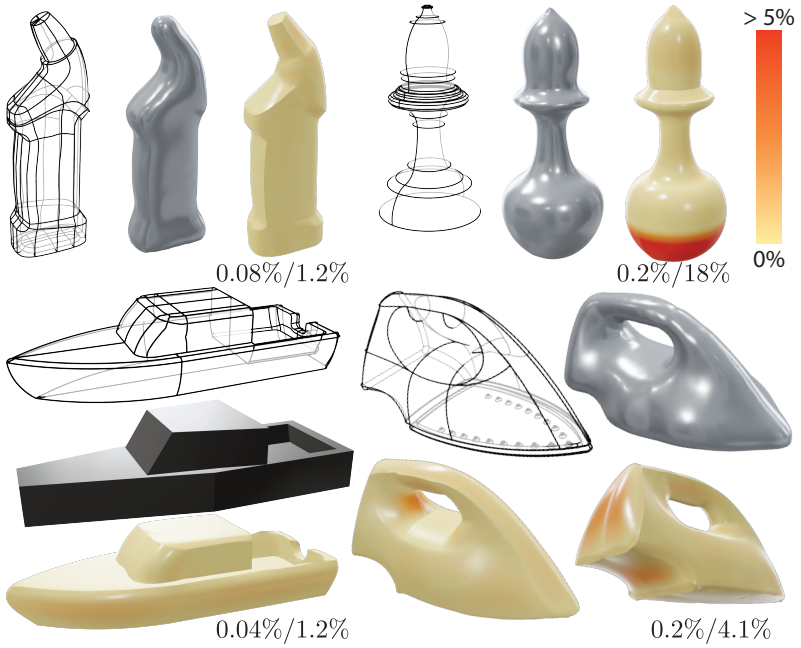


Fig. III.14: We evaluate our method on synthetic sketches extracted from ground truth surfaces using *FlowRep* [119] (bottle, boat) or from the boundary representation of CAD models (iron, bishop). The deviation is measured as a % of the sketch bounding-box diagonal, and is low almost everywhere except in places where the sketch lacks strokes to disambiguate the intended surface (interior of the iron handle, bottom of the bishop). We provide the median and maximum deviation for each sketch.

Comparison against ground truth surfaces We evaluate our method quantitatively by comparing our results to ground truth surfaces for which curve networks are available. We obtain such data from *FlowRep* [119], a method to generate descriptive curve networks from 3D meshes, as well as from CAD models for which we extract the boundary representation (B-rep) [175]. Fig. III.14 visualizes the results of this experiment, where the color map indicates that our results are very close to ground truth except in ambiguous regions devoid of strokes.

Comparison against curve network surfacing Fig. III.15 compares our method to the one by Pan et al. [237], which is the most recent method for surfacing well-connected curve networks. The two methods produce very similar results on the original connected network. However, a unique strength of our method is to also produce similar results from disconnected, noisy sketches, which we synthesized by perturbing the original network (displacing and duplicating strokes, introducing gaps). Our method also accounts for strokes that are not connected to the main network, which Pan et al. [237] had to ignore (Fig. III.15, red strokes at the front of the car).

In Fig. III.16, we compare to Pan et al. [237] on a more complex input. By balancing

Table III.1: For each sketch in the paper, we provide the runtimes for the initialization **(1)** – which corresponds to the strokes-proxy association (Section III.5.1) and step 0 of Algorithm 1 ; for the iterative segmentation **(2)**, and for the final mesh optimization **(3)**. We also give the number of iterations of Algorithm 1 necessary to converge to a stable solution.

Sketch name	(1)	(2)	(3)	It.	In paper
robbins_car1	46.3s	11.4s	68.3s	8	Fig. III.2
author1_car	16.7s	1.9 s	41.4s	5	Fig. III.3
flowrep_boat	17.1s	1.6 s	45.7s	3	Fig. III.14
flowrep_bottle	17.6s	2.2 s	20 s	3	Fig. III.14
onshape_bishop	9.2 s	1.2 s	14.8s	4	Fig. III.14
onshape_iron	20.6s	5.6 s	25.8s	6	Fig. III.14
flowsurf_beetle	14.6s	0.8 s	31.2s	2	Fig. III.15
ils_roadster	9.3 s	1.5 s	35.3s	4	Fig. III.15
t2f_car_97	18.2s	1.6 s	18.7s	3	Fig. III.19
author2_guitar	16.2s	1.6 s	26.8s	3	Fig. III.19
robbins_car2	62.3s	9.5 s	64.5s	6	Fig. III.19
author1_building	32.9s	4.8 s	41.9s	4	Fig. III.19
ils_speaker	16.9s	1.2 s	23.5s	3	Fig. III.19
swh_vr_controller	146.4s	79.1s	35 s	8	Fig. III.19
author1_bulbasaur	11.1s	3.7 s	23.2s	5	Fig. III.20
cassie_hat	17.7s	1.2 s	60.3s	3	Fig. III.20
tolentino_shoe	57 s	8.2 s	38.8s	5	Fig. III.20

fidelity with smoothness and simplicity, our method tends to miss small details, such as the buttons on top of the machine. Our smooth surface models also do not capture well the generalized cylinder that forms the nozzle. In contrast, by assuming that the input strokes form a clean curve network, Pan et al. can trust every curve to create interpolating surfaces and can leverage connectivity information to detect which curves are sharp features.

III.7.3 Limitations

Relying on geometric criteria only. We emphasize that our method relies purely on geometric criteria to place sharp features in the final surface. As a consequence, our method can miss semantically-important features if they do not contribute significantly to the shape, such as the round headlights of the car in Fig. III.2.

Sharp features not depicted by the strokes. Our algorithm assumes that all sharp features of the intended surface occur along some of the input strokes. Fig. III.17 illustrates a limitation of this assumption, where the artist implicitly indicated that the headlight presents a sharp feature by sharp corners in neighboring strokes. In such cases, users

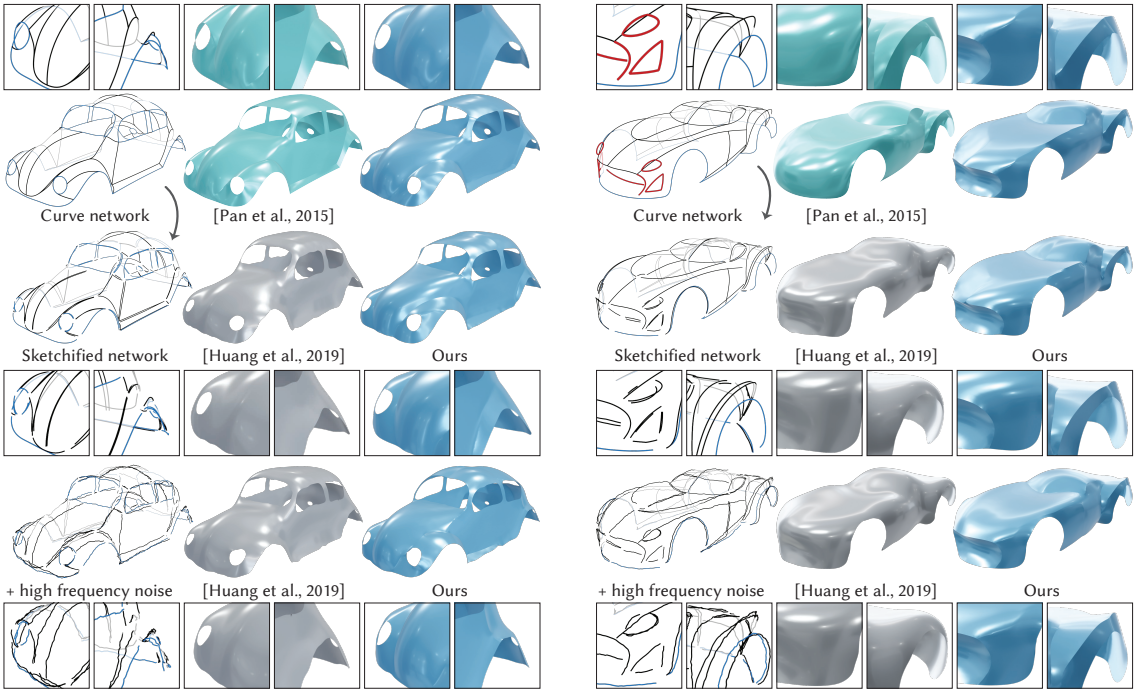


Fig. III.15: Our method can surface well-connected curve networks, similarly to [237]. More importantly, it also achieves similar results from imprecise sketches that contain duplicate strokes, gaps (middle row) or high-frequency noise (bottom row). Our method also accounts for disconnected parts, like the red stroke that depicts a concavity on the car, which Pan et al. [237] ignored.

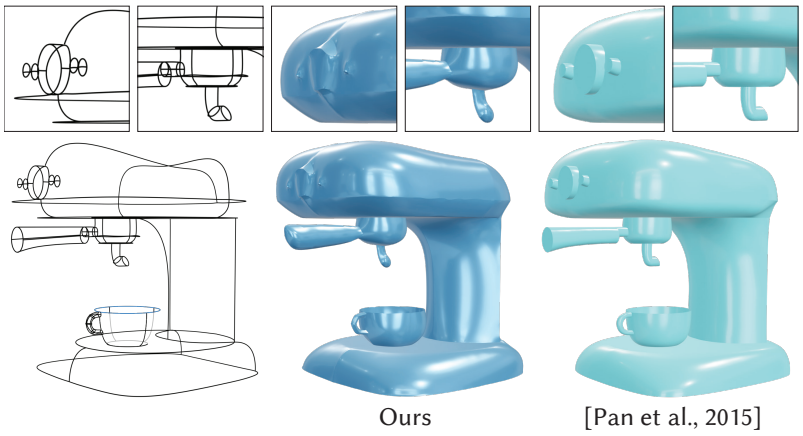
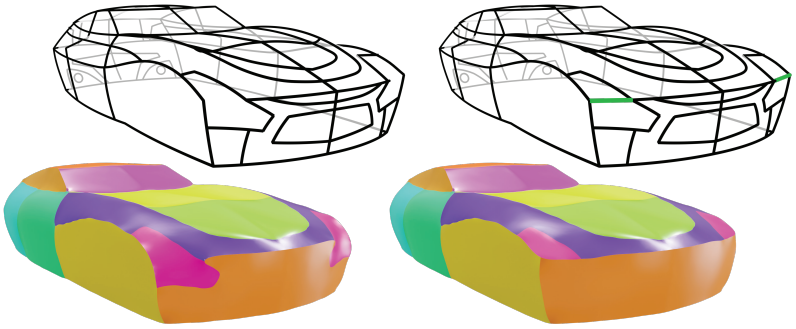


Fig. III.16: Our method tends to miss small yet important details (middle) in an effort to produce smooth, simple outputs robust to imprecise drawings. The method by Pan et al. [237] does not face this challenge as they consider that all input curves are drawn precisely, and as such should be kept in the output. Similar to Pan et al. [237], we surface the coffee cup separately from the coffee machine.



(a) No stroke at sharp feature (b) Adding a stroke

Fig. III.17: Our method cannot take higher-level cues into account, and will thus fail to align with user intent if a critical sharp feature stroke is missing, as on the car headlight (a). Adding a stroke and re-running the method successfully recovers that feature (b).

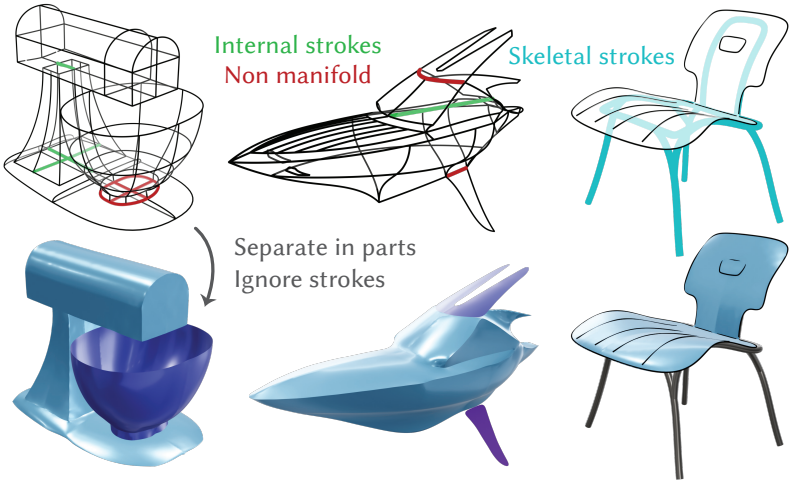


Fig. III.18: Our current implementation does not automatically support non-manifold surfaces, as present on the boat and blender (red strokes). Furthermore, our assumption that strokes lie on the surface they depict is not always true, as there may be some strokes inside the outermost depicted surface (green) or strokes that depict tubular structures (cyan). A workaround is to manually separate the sketch into parts and surface them individually, or mark out some strokes to be ignored by our method (second row).

need to draw a few additional strokes to obtain the intended surface (Fig. III.17b). While our assumption that sharp features are explicitly represented seems to hold in our dataset, further analysis of how people draw in 3D is needed to quantitatively evaluate this hypothesis. In the future, more global cues such as stroke tangent continuity or sketch symmetry could be leveraged to make surfacing better aligned with viewer perception.

Other types of strokes. Since we assume that all strokes lie on the intended surface, our method cannot handle strokes that lie inside the shape, or that depict the skeleton of a tubular structure (Fig. III.18). While we simply discarded such strokes to produce our results (see supplemental webpage [3] for all original input sketches, with deleted strokes highlighted in red), future work could attempt to identify these strokes and surface them with dedicated representations, such as generalized cylinders for skeletal stokes [348].

Non manifold configurations. In theory, our method could recover non-manifold surfaces if provided with a non-manifold proxy mesh. In practice, we only surfaced manifold shapes because we implemented our algorithm using a manifold data-structure to represent the mesh. As a work-around, it is typically possible to manually separate the sketch into multiple manifold pieces that can be surfaced separately (Fig. III.18, bottom row and coffee cup in Fig. III.16). An exciting direction for future work would be to perform topology analysis of the unstructured sketch to detect and process non-manifold parts automatically.

III.8 Conclusion

After decades of research, a number of robust algorithms now exist to surface dense point clouds [33]. In contrast, very few methods have been proposed to surface the sparse *stroke clouds* that designers produce when sketching in VR or with sketch-based modeling systems. Inspired by the unique characteristics of this emerging form of 3D data, we have proposed an approach to locate smooth patches in unstructured 3D sketches, and to optimize their geometry to produce a piecewise-smooth surface aligned with salient strokes. The resulting 3D meshes can benefit numerous tasks. For instance, we used them to occlude hidden strokes in all figures of this chapter, which helps perceive the correct shape from the sketch via relative depth cues [19]. Additionally, our sketch-aligned surfaces are compatible with all downstream 3D processing and modeling tasks. This opens exciting avenues to integrate 3D sketching with other creation modalities such as sculpting [7, 39, 215]. As we have demonstrated, sculpting can be used to create a rough proxy surface, that is then automatically refined by our method to align with a sketch. It is easy to further refine the surface by sculpting, since our method outputs a 3D polygonal mesh compatible with all 3D modeling and sculpting software.

[3] https://ns.inria.fr/d3/Surface3DSketch/results-page/

III.9 Future work

Surfacing partial 3D sketches We cast the problem of surfacing a 3D sketch as an offline task that is run only when the sketch is complete. In previous work we also explored the opposite end of the spectrum by continuously creating surfaces while the artist is sketching [337]. Exploring other options along this spectrum seems promising to afford more flexibility in the sketching and surfacing process, and power new workflows that are not yet possible if we view surfacing as a disparate step to sketching. For example, an artist might want to sketch a partial 3D sketch, then generate a surface to provide visual or physical support for further sketching – drawing curves by projecting them on a surface is an efficient way to sketch 3D curves with 2D input devices, and can alleviate freehand imprecision issues with a 3D input device [18, 19, 172]. Newly sketched strokes could in-turn affect the 3D surface, by updating it wherever new strokes deviate from the surface. Our current work cannot be easily adapted to this setting. While our method can be made more efficient to reduce computation time with more engineering effort, it is not trivial how to take into account new strokes or edits on an existing stroke without globally affecting the surface. Moreover, partial sketches introduce more ambiguity to the surfacing problem since some regions may be altogether undefined at a given stage of the process.

Exploring a 3D shape design space In this project, we cast the problem of obtaining a 3D surface from a 3D sketch as a *reconstruction* problem. This follows the tradition of point-cloud surfacing methods, that posit that there exists a *ground truth surface* sampled partially with sensors that the method strives to reconstruct. However, in the case of 3D sketches there is arguably no such ground truth surface to reconstruct, since a 3D sketch is not created by a sampling process. Instead, an artist creates a 3D sketch as an ambiguous representation of a *space* of possible 3D shapes that they are trying to gradually refine by iterating between externalization through sketching and reflection through reviewing the sketch [52, 148]. Exploring the design space of possible 3D surfaces that can be represented by a given 3D sketch could be a powerful tool for artists, by favoring exploration of multiple possibilities instead of freezing the design to a single 3D surface. Recent work on stochastic surface reconstruction [275] might be a valuable inspiration to reconstruct a distribution of shapes from a given 3D sketch. Such a distribution could be used to sample shapes or help the artist determine which areas of their sketch is most ambiguous.

A unified toolset for surface and curve modeling Previous work has shown that 3D curves can be a powerful surface modeling tool [76, 111, 226, 286, 317]. Our method reconstructs a 3D surface from a set of 3D curves, yet we did not explore how to bind the curves and surface together to enable such edition operations. 3D strokes are placed strategically on the depicted surface to represent it in a sparse way, so we expect that they could also act as powerful handles to control shape deformation. The main challenge

of our setting compared to previous work on surface deformation using curve handles is the unstructured and messy nature of a 3D sketch, as previous methods focused on clean networks of curves or supported only a few curves.

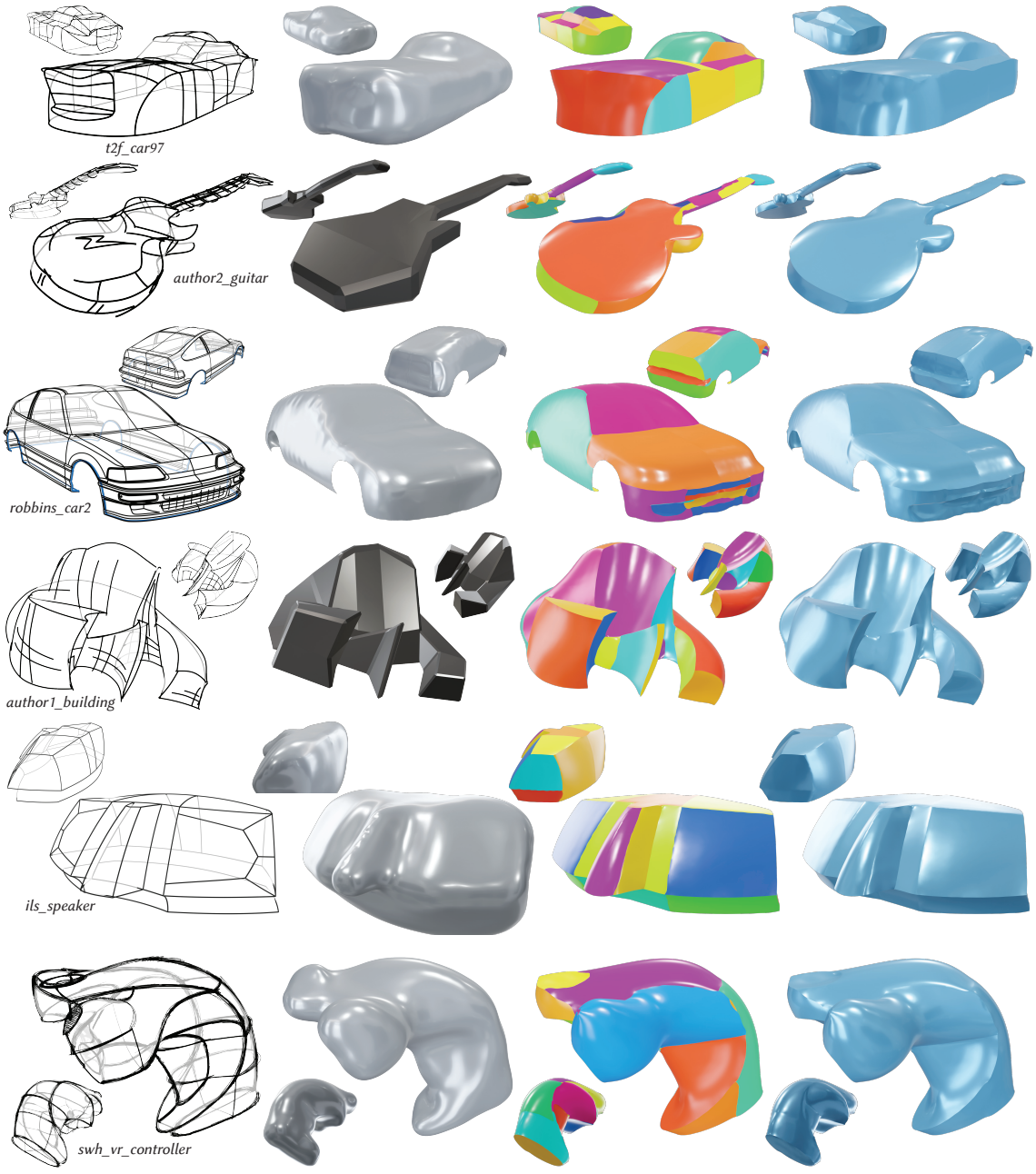


Fig. III.19: Results of our method applied to a variety of 3D sketches (a). We display proxy meshes computed with *VIPSS* in light gray, and the ones created manually in dark gray (b). For each sketch, we show the segmented patches with random colors, and the final surface in blue (c). Sketch *robbins_car2* ©James Robbins, used with permission.

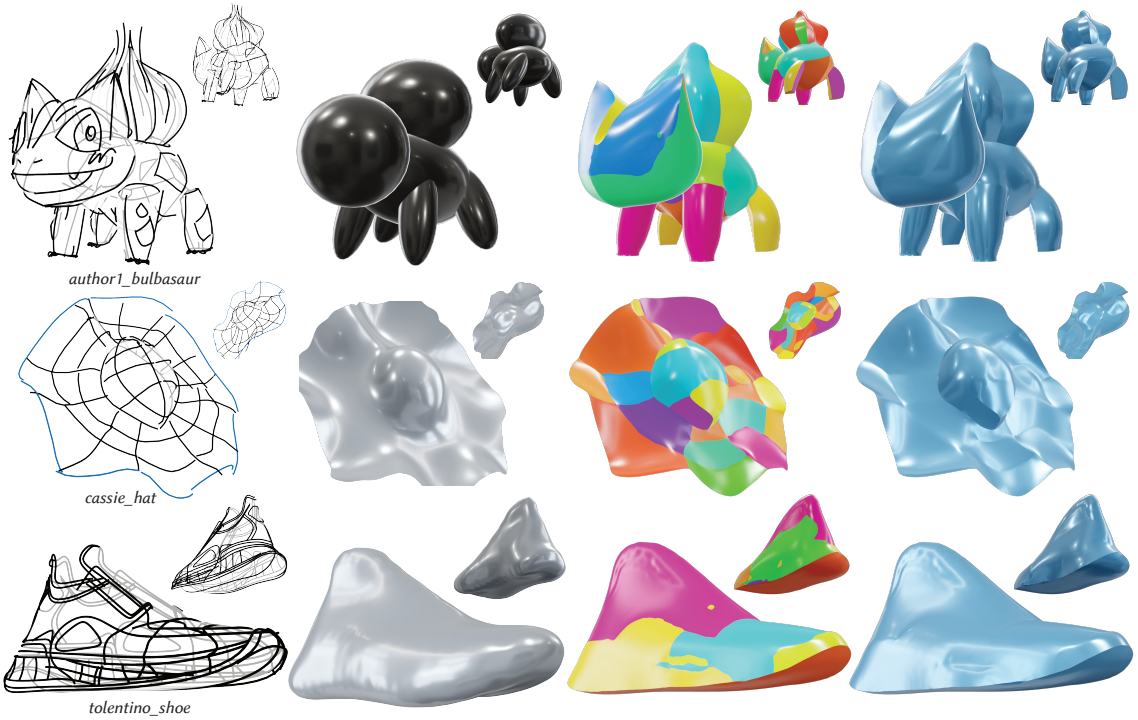


Fig. III.20: Results of our method applied to sketches of organic shapes (a). We display proxy meshes computed with *VIPSS* in light gray, and the ones created manually in dark gray (b). For each sketch, we show the segmented patches with random colors, and the final surface in blue (c). Sketch *tolentino_shoe* ©Arturo Tolentino, used with permission.

Chapter IV

How Do People Paint In VR?

IV.1 Introduction

3D strokes can represent more than just a 3D surface. By assigning colors to strokes, artists can use them to represent appearance of a scene, in the same way that an oil painting can represent any scene through a complex arrangement of many colored brush strokes. *VR painting* is an emerging form of 3D art that lets artists use 3D space as their virtual canvas [140, 267, 289] to create compelling *3D paintings* used as video game assets [184], in short animated movies [109, 213], or as 3D illustrations [259]. Despite the simplicity of its underlying representation – an arrangement of 3D strokes – VR painting allows artists to create complex 3D scenes made of varied shapes and colors (see Fig. IV.1).

Given the growing interest from artists in exploring the potential of VR painting as a medium for 3D content authoring [60, 118], and the success of commercial applications for VR painting such as Quill and OpenBrush, we see an opportunity for academic research to study this emerging workflow. Many studies and systems related to creating 3D content with VR have focused on the problem of geometric accuracy when sketching 3D shapes, for example to measure inaccuracies and explain their cause [19, 211], to investigate the potential for novices to learn to sketch accurately [323] or to design systems that mitigate the lack of accuracy in sketching [18, 87, 157, 210, 337, 338]. Previous work has looked

Fig. IV.1: VR painting examples. Left to right: "Hopewell Rocks, Fundy, New Brunswick" (©Nick Ladd), "Mescaform Hill: The Missing Five" (©Edward Madojemu), "This is fine" (©Tiantian Xu)

into the potential of VR sketching for a variety of applications such as 3D design and visual thinking [148, 185, 235], helping 2D artists create 3D artworks [129] or patient rehabilitation after suffering from a stroke [12]. Yet, none of these studies has considered how practicing VR artists with deep experience of this medium work. By investigating how these artists work, we can see a fuller picture of the potential of this novel artistic medium and of the challenges that software or hardware improvements might mitigate. Studying expert users can be helpful in better understanding difficulties faced by all users, and design better solutions inspired by ad hoc solutions. We draw inspiration from other similar studies in other application domains (Section IV.2).

In this chapter, we report on a series of semi-structured interviews with 4 participants who are hobbyist VR artists or professional artists with VR painting as their main medium. We report on how artists use VR painting (Section IV.3.1), we summarize the motivations for working with VR painting (Section IV.4), and the main challenges encountered by participants (Section IV.5).

IV.2 Related work

VR painting practice is currently understudied in academic research. We take inspiration from other studies of artistic practices to conduct our study on VR painting.

Previous work has studied a variety of visual design practices, such as icon design [346], undo and redo [222], color palette usage [155, 285], the usage of reference imagery [131], visual note taking [347], math formula visual design [127], educational diagrams [209], live-coding audiovisual performances [46], visual choreography documentation [62] and more generally how artists use and develop software [191]. These studies led to the formulation of design recommendations, and some authors also contribute the development of a prototype or technology probe [139] based on their design recommendations. We take inspiration from their methodology in our study design, for example by conducting interviews with expert practitioners that have a better understanding of VR painting than other populations such as novice users. We also follow the methodology of many of these studies who prompt participants to recount their creation process. More precisely, Maudet [214] recommends using artifacts as "focal points" in interviews with designers to help participants recount the story of their workflow, prompt specific memories about challenges encountered, and facilitate understanding through a visual support. We adopt this methodology for our interviews.

Most closely related to our topic of interest, researchers from the fine arts community have explored what VR painting can mean for painting as an artistic practice. Goodyear and Mu [118] report on Goodyear's personal experience working with VR painting tool OpenBrush [140] as part of her abstract painting work, providing insights into how artists might appropriate such tools to serve particular practices. Chittenden [60] analyze 3D painting through the lens of art history, explaining how VR painting relate to previous

painting techniques that blurred the separation between viewer and depicted picture. We expand their insights by taking a more focused and critical look at the interaction design and digital representations used in current VR painting software, drawing from our experience as software developers and HCI researchers.

IV.3 Procedure

We conducted semi-structured interviews and analyzed online tutorials to better understand how artists who practice VR painting create and edit their artworks. We focus on the commercial software Quill [289], because a short survey of online communities revealed that this particular VR application has a very active and open community of users, with a Discord server featuring 300+ active users sharing artworks and tips on the software, a Facebook group with 20K+ members, and a YouTube channel with 62 long-format tutorials where artists share a diverse set of workflows [1]. Despite our focus on Quill during this formative study, we surveyed other popular VR painting applications such as Open Brush and AnimVR, and confirm that the workflows and challenges encountered are similar.

After learning informally from online tutorials and getting a feel for how people work with Quill in general (eg [183, 223, 242]), we invited 4 artists who have used Quill in professional or personal projects extensively to participate in interviews and provide more in-depth insights (Table IV.1). We invited participants that we saw as active members in VR painting communities (Discord channel, Facebook group), or on art sharing platforms (Sketchfab, Artstation, personal blogs). We invited each participant to a 1 hour video conference call, and asked them to be prepared to discuss a recent project that they worked on. We encouraged participants to share a particular piece or artifact by sharing their screen or providing links to a hosting platform, and walk interviewers through their process in creating that piece. Participants often showed multiple artifacts through the course of the interview. We encouraged the participant to give a detailed account of their workflow, and we asked for clarifications or more context when participants would describe breakdowns and how they overcame those [212]. Participants received a 30CAD gift card. This study was approved by the ethics boards of the University of Toronto and Inria.

We audio-recorded and transcribed each interview, then coded the data. We used an inductive thematic analysis approach [50]. We started our analysis without a pre-existing coding frame and were open to emerging themes, letting our research questions evolve as the analysis goes on. Our broad research questions were to understand *why* artists use VR painting as opposed to other forms of 3D authoring, and *how* artists work with this medium. We are interested in finding what challenges participants face in their practice,

[1] Facebook group: https://www.facebook.com/groups/virtual.animation/ ; Youtube channel: https://www.youtube.com/@VirtualAnimation

Table IV.1: Participants information.

Participant ID	Description	Artworks discussed
P1	Illustrator, VR artist	Animated illustration, background and character models for VR animated shorts
P2	Director, VR artist, animator	Game background scenes, animated and print illustrations, self-directed VR short
P3	Illustrator, animator, developer	Self-directed VR short, animated illustrations
P4	Product designer, artist	Animated illustration, artifacts of learning Quill

and what techniques they use to overcome them. In the following analysis, we start by an account of the different ways artists work with VR painting, then we focus on some of the themes that emerged. The key themes we report are chosen based on a joint consideration of theme prevalence among participants and of how our research question evolved, driven by our standpoint as HCI and computer graphics researchers.

IV.3.1 VR painting workflows

In this section we report how participants work with VR painting, striving to provide a broad view of the different workflows and techniques.

IV.3.1.1 Broader creation context

Participants used the VR painting software Quill in their professional or personal projects for a variety of end results (see Table IV.1 last column). In all projects, Quill was used as the main software for 3D modeling, but participants sometimes complement Quill with other software for later stages of the authoring process, such as Blender and Unity for shot planning and rendering (P3), Premiere Pro for compositing and editing shots (P1), Sketchfab and Quest TV for adding visual effects and providing an interactive 3D viewer (P1, P2, P4). In some projects Quill was *only* used as a 3D modeling tool as part of a larger pipeline, for example P2 would model background sets and other artists were in charge of assembling this background set with other elements to make a VR game. Conversely, P1 used Quill end-to-end in the creation of her animated illustration, from the first rough sketch to the final result featuring small animation loops, and hand-painted visual effects that simulate a water surface – *"little touches that just really add a sense of atmosphere"*. She then used Sketchfab as a hosting platform and renderer, to allow viewers to navigate her scene freely. Other ways of sharing VR artworks included releasing content on VR-specific platforms such as Quest TV and the VR Animation Player, and rendering the

scene from a fixed camera path to create a video shareable on social media.

In many cases, participants used images (P2, P3, P4) or 2D sketches done prior (P3) as references – those can be imported in the virtual workspace. The artwork itself is often created in multiple stages: P1 used a rough 3D sketch made of thin lines to *"figure out the sense of scale and position of the different elements"*, and P2 and P3 started with creating a rough block out of their indoor scene that *"helps answer a billion questions, like spatial layout" (P3)*. Such temporary steps of the artwork can be organized in "layers" in Quill, that are stroke containers. Layers in Quill do not affect stroke rendering order or appearance like one might expect layers to do in 2D digital painting. They only allow control over opacity of its content to make a layer semi-transparent or hide it altogether.

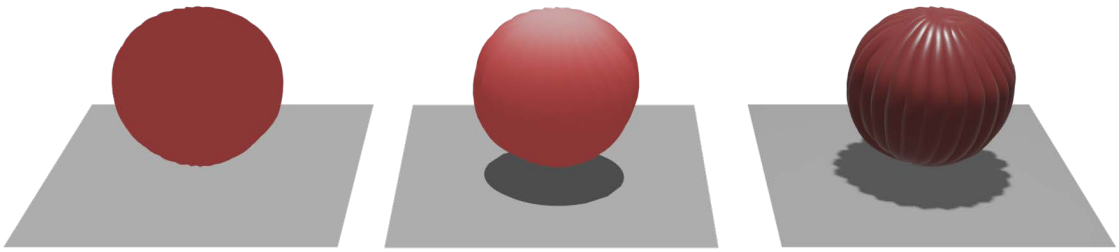
In the following, we focus on the workflow for 3D content authoring, which covers 3D shape creation and the definition of appearance. We leave to future work more in-depth studies of the end-to-end authoring workflow, or of specific parts such as artwork sharing.

IV.3.1.2 Authoring shape and appearance

The traditional 3D pipeline tends to have a clear separation between the step of modeling 3D shape (Section II.1) and the step of defining the appearance of a 3D asset through material and texture authoring (Section II.2). Despite some participants (P2, P3) being familiar with this workflow, they talked about the way they paint in VR *without* mentioning an explicit separation between shape and appearance authoring. The way participants described their workflow resembled the account of a traditional painting experience, and it is summarized in these words from P2: *"you can just kind of draw in the world right?"*

People create shapes in VR painting by sketching colored strokes mid-air. In Quill each type of strokes has a particular base shape which in turn influences the shape of the stroke, for example a spherical brush will yield a tubular stroke. P3 explains choosing the shape of his brush carefully depending on the shape he is creating, using a spherical brush to create *"the core structure of the character"* while using flat strokes to imply the thin fabric of the character's outfit. Once a stroke is created, it can be deformed, rigidly transformed and duplicated. Strokes can be recolored by using a brush that affects all parts of strokes that fall inside. All participants recounted using a process in multiple steps, with a combination of sketching strokes, duplicating and deforming, recoloring strokes.

Quill renders all strokes with *unlit shading*, meaning that the final color displayed is fully determined by the color of the stroke itself (see Fig. IV.2a). How then do artists create the illusion of light illuminating the scene? P4 explained the process in this way: *"For oil painting or traditional painting you have to paint light and that's what you're doing in Quill."* Participants all had techniques to "hand-paint" light and shadows in their artwork (see Fig. IV.2b), for example simulating the sun illuminating a clearing in the forest with



(a) Unlit render

(b) Unlit render with hand-painted lighting

(c) Lit render

Fig. IV.2: Lighting VR paintings. Quill renders strokes with *unlit shading* (a), so artists often paint shading effects and shadows by hand (b), by using gradients (on the red sphere) or drawing new strokes (shadow disk). Rendering a Quill painting with computer generated lighting that takes into account the surface normals to compute shading reveals that the surfaces are built out of strokes (c).

a gradient from dark to bright (P3), blending in bright yellow with objects close to the sun (P1), or drawing simple cast shadows to simulate a light source from above (P4).

Participants were aware that this workflow of painting shading differs from other 3D authoring workflows in which the computer renders final appearance while the artist controls indirect factors such as lighting and surface material parameters. P3 was acutely aware of the difference between authoring in Quill and authoring within the traditional 3D pipeline, and sought to find *"a symbiotic way between the 2 pipelines"*. He explored combining hand-painted ambient lighting with dynamic computer generated lighting in a rendering engine, in order to create a short animated movie with a scene that goes through a day-night cycle. Computer generated lighting allowed him to *"completely reinvent how the scene feels and looks without having to do a lot of manual work"*. He developed custom stylized shaders and tricks to modify normals of 3D models, in order to combat the *"sausage effect"*, caused by 3D geometry being composed of many messy strokes (see Fig. IV.2c). P2 used a rendering engine to previsualize how the 3D scene he modeled would look like with the lighting setup he imagined. He rendered a few still images to be used as references, yet allowed himself to stray from the computer generated reference to better serve storytelling: *"I kind of cheated like for the briefcase it's really important that the character can click on it and see it, so I kind of made it brighter on purpose."*

IV.4 An accessible, direct and controllable 3D authoring tool

In this section we summarize the themes related to *why* people like working with VR painting to create 3D content.

IV.4.1 Accessibility and directness

VR painting is an accessible and approachable tool, that participants described as easy to learn. Participants appreciated how "direct" VR painting felt to them from the very start of their learning journey (P1, P4), they describe VR painting as an embodied experience: *"I like that it's really about to be able to just use your body to kind of manipulate objects" (P4)*.

The participants all had previous experience working with 2D digital illustration before getting into VR painting. Some transfered that knowledge to approach VR painting, describing VR painting as *"basically like a Photoshop" (P2)*. Participants talked about their workflow through analogies between tools available in 2D painting and Quill. For example P1 recounts using a recoloring brush with color blending modes to define lighting just like how she would use multiple layers with blending modes in Procreate [146]. However, she explains that such an analogy has limits since there isn't such a notion of non-destructive layering of colors in Quill, instead colors have to be *"applied"* to strokes.

Multiple participants (P1, P2, P3) described a positive experience in transferring from an illustration background to learning VR painting, as it *"allows [them] to use their skills directly in 3D" (P3)*.

In particular, some participants mentioned a sharp contrast between their experience with learning Quill and learning to author 3D with the more traditional 3D pipeline with desktop applications. P4 noted it took her only a few days to start creating her *"own art"* in Quill, as opposed to several months in Blender, and P3 explained that traditional 3D had *"technical aspects to it that get into the way of actually executing what you want"*. Similarly, P2 explained that the rapidity of working with VR painting – critical to him in time sensitive contexts like during a game jam – was something he felt that he could not achieve within the traditional 3D pipeline: *"in regular 3D you have to unwrap, you've got to texture the thing, bring it back. In here I can just... if I want this window to be red, I can just, you know, draw a red window."* This fast feedback loop between idea and realization can also favor exploration and serendipitous creation: *"sometimes you'll draw a stroke by accident but it will work out, or I'll start with a sketch and then I'll build off of that sketch" (P2)*.

These accounts of the "directness" of painting in VR are consistent with our overarching observations from Chapter I. Participants appreciate the possibilities for using their own bodies in the interaction [151], and they liked transferring interaction knowledge from their previous digital experiences [254]. Overall we can interpret their experience using 3D strokes in VR painting as requiring them to bridge less of a *semantic distance* to get to "what they want" than in other experiences using other 3D data representations [138].

Fig. IV.3: Stylized shadows. In VR paintings, artists sometimes opt to simplify how shadows look like as a stylistic choice. Left: the character's drop shadow is painted as a simple circle on the ground, same for the little statues (©Nick Ladd). Right: the window frame casts a very distinctive shadow on the floor, but this shadow is not cast on the painting canvases, and the canvases, easel and table do not cast a shadow (©Zoe Roellin).

IV.4.2 Artistic control

When recounting their workflow, especially when describing how everything in an artwork was drawn or placed by hand which amounts to a tremendous amount of work, participants did not mention feeling like this was a tedious task. While they recognized the limitations and challenges of this workflow – which we cover in more details in the following section – they explained that having the ability to paint everything afforded them a lot of artistic control over the final result. In particular, when discussing how shading is hand-painted in VR painting, as opposed to automatically calculated by a computer, P3 who has deep experience of working with both explained that ultimately if it weren't for time budget limitations he would prefer to hand-paint shading, or at least have a mix of both approaches: *"I like to have as much control up front rather than relying on fidelity of what the computer can render. I'd rather have something that works because it's well art directed rather than because the computer renders it."* P2 explained how for his indoor game environment scene, painting shadows manually allowed him to emphasize *"the style that we're going for in the game"*, by choosing deliberately to simplify the shape of shadows, for example drawing just an ellipse for the shadow of a stool, despite the physically accurate shape of the projected shadow being much more intricate.

Participants talked about the way they achieved certain effects with a sense of pride in having found certain *"tricks" (P2)* or *"illusions" (P1)*, overcoming the software limitations by coming up with new techniques. For example P1 explained faking colored glass windows by recoloring all outside shapes to look a bit more grey, and P2 created an effect of a reflective water plane by simply duplicating the painted shapes above the water and mirroring them. P3 talks about how he overcomes the lack of precise shape modeling tools in Quill by using additional *"hand-drawn lines to imply that there's more detail or more surface that there actually is"*.

Sometimes, participants described embracing the limitations of VR painting by making them part of what defines the artwork's unique style. P4, who set herself the challenge to learn Quill within only 100 days of practice explained that she chose for her artwork to *"make it as simple as possible"* and that she ended up being very happy about these *"minimum viable product"* results. P2 described a 3D scene he created that had a visual style reminiscent of 2D vector art by emphasizing its simplicity: *"I intentionally left it very... almost abstract just because I think it's something that you can't easily do in regular 3D, so I like to embrace that sort of style instead of trying to be too realistic or too detailed."* Similarly, P1 mentioned using deliberately freeform brush strokes that are set a bit apart from surfaces without aiming for precise alignment as that was what made her artwork have a recognizable style in the community. The lack of geometric accuracy was also seen as an interesting quality of the medium by P2: *"because I'm doing it with my own hands, it's... more likely that mistakes will come up and the mistakes I think are part of what make it more human and more alive"*.

IV.5 Challenges

We end our analysis of the interviews by reporting prevalent themes related to challenges that participants face in their workflow.

Coupling of strokes and appearance Strokes in a VR painting are represented digitally through a *discretization*, and colors are encoded as per-element data on this discretization, with linear interpolation in-between elements. Therefore this discretization defines how finely variations of color can be achieved (see Fig. IV.4). For example, P1 explained that this means the direction in which flat strokes are assembled to form a box will affect strongly the look of a gradient along the strokes, meaning that for a desired gradient direction, strokes *"need to be pointed in the right direction"*. This technical detail has strong implications for the hand-painted shading technique: *"all the painted lighting has to be bound to the strokes, which means the quality of your lighting is directly proportional to the resolution of the geometry you're using to paint the lighting." (P3)* In video tutorials, we have seen artists plan out how they lay their strokes carefully in order to create hard edges between colors, and achieve a certain direction of gradient. The resolution of the discretization also matters in order to obtain smooth gradients. The fact that stroke authoring has implications for how finely *appearance* can be edited is a challenge that experienced artists attempt to overcome through careful planning of stroke directions [242], or artificially increasing the resolution of strokes [183, 223].

Memory budget and performance management Quill artworks and VR paintings in general can be shared with the public in different ways (Section IV.3.1.1), but all participants related that having the possibility to share their artwork on platforms for VR viewing was important to them, sometimes stating that the immersive viewing experience

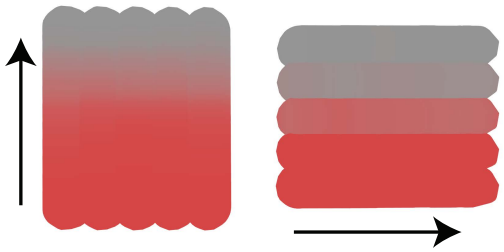


Fig. IV.4: Stroke layout affects coloring possibilities. Strokes are discretized as polylines with per-vertex colors in Quill. Thus coloring a patch of strokes depends on stroke orientation (black arrow): a smooth vertical gradient is easy to achieve with a patch of vertical strokes, but hard to achieve with the same patch of strokes rotated 90 degrees.

is a defining part of their piece (eg immersive animated short where the spectator has to navigate the set during the story). However, viewing artworks on VR headsets comes with technical restrictions on the artworks, due to hosting platform limitations or headset hardware limitations: *"you need to be under a certain amount of triangles, you need to have a certain amount of draw calls" (P2)*. For example, P3 mentioned a *"1.5GB limit"* for a 15 minutes animated short that he and his team published on Meta Quest TV, which *"was kind of something [they] had to be very careful with"*. The resolution of stroke discretization directly impacts this memory budget, which amplifies the problem of having appearance editing relying on degrees of freedom provided by stroke discretization: *"in order to be able to color and shade very smoothly that shape needs to be [...] like super super high detailed as far as polygons go which is something I try to avoid in Quill because otherwise Quill drawings get really really heavy" (P1)*. Both P1 and P2 reported that as their experience working with VR painting increased, they changed their drawing habits to adopt techniques that yielded paintings with lower polygon count: *"the stuff I was making when I first started was way too heavy. So I have started to be a bit more conservative with the strokes, to just kind of you know make sure I don't waste too many strokes." (P2)* Similarly, P3 describes some *"best practices"* that he and his team adopted on their animated short to respect the memory budget, such as placing strokes strategically to support specific shading change effects without increasing polygon count.

Editability Sometimes a VR painting needs to undergo changes, for example a character model has to be adapted to match the environment light in multiple scenes of an animated short movie, or a background scene must be changed to better reflect a director's vision for the mood of the scene. Since Quill in particular provides many features for animation, artists also might want to have dynamic elements in their artworks, such as a character moving through the scene. In all of those situations, hand-painted shading may need to be edited. While in particular in Quill there is a feature to "recolor" strokes, participants reported that the recoloring workflow was *destructive*, meaning that once a new color was applied to a stroke – even with partial opacity or a color blending mode – it becomes

impossible to separate again the constituent colors at a later stage. This means that once a particular shading is applied to the scene – eg applying an orange hue gradient to simulate a rising sun glow – it becomes impossible to recover the initial look of the scene in a neutral light, except if one were to recolor strokes to their original color one-by-one. P3 explains that *"trying to recolor a scene to match a different tone is a lot more... not only is it more time consuming, it's that you end up cutting corners really quickly because there's so much to do"*. As a strategy to protect her work from this destructive step of the workflow, P1 recounts saving duplicates of her artworks, for example keeping a duplicate of the environment in an invisible layer before applying the recolor brush, or for characters starting *"with just a very basic color character"* and only painting the light and color *"as the very last step"*, so that in a different lighting environment she can reuse the basic color character. Some dynamic effects such as drawing the cast shadow of a moving object can also be difficult to achieve by hand-painting that shadow, so P2 describes a work-around where he set up the strokes making up the shadow of a turning ceiling fan to turn in sync with the strokes making up the fan.

Depicting 3D shape effectively All participants touched on the topic of how they learned to depict 3D shape effectively with Quill, or on how their style and workflow evolved as they got more experienced with the tool. As noted in quantitative studies [19], controlling 6 degrees-of-freedom with the hand is challenging: *"it's hard to do something precisely in Quill, because I'm using my hand, I'm using my controller" (P4)*. P1 explained she found it difficult to represent solid surfaces with many thin strokes without things looking *"messy"*: *"when you start rotating things around, suddenly you could see all kind of gaps between your strokes and so on" (P1)*. Finally, as reported in previous work [211, 323], it takes time to learn to sketch in 3D: *"It felt like going back to basics because you learn figure drawing, you get really good at it, and then you jump into VR and none of it works because you draw something and then your head moves the camera a little bit and then it doesn't look like anything anymore" (P3)*. Participants also explained that some techniques can alleviate these issues. For example P2 explained that using Quill's "line" tool that enables drawing axis-aligned lines precisely helps him create large flat patches that are well-suited for architectural or interior design. He also uses a particular surface rendering mode when painting in Quill that helps perceive where existing strokes lie in 3D space (see Fig. IV.5), in order to facilitate alignments. P1 and P3 reported that their painting technique evolved into a mix of 3D modeling – done in Quill by using large volumetric strokes – and freeform stroke painting: *"nowadays I have really started modeling the base of my characters with very simple shapes [...] and then just drawing in the actual lines and details at the end" (P1)*.

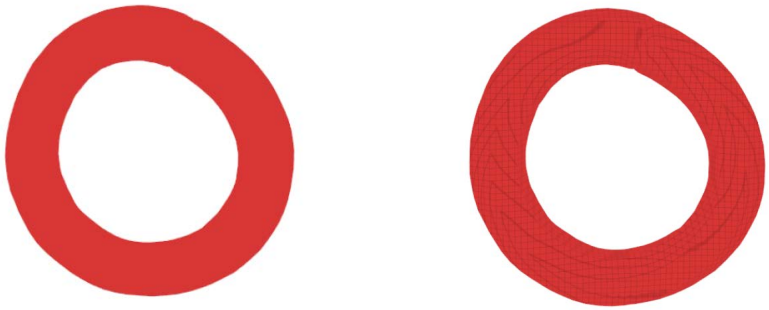


Fig. IV.5: Stroke rendering can help increase accuracy. In Quill strokes can be rendered with a subtle 3D grid texturing that helps perceive 3D shape (right), compared to the default unlit shading of a stroke (left).

IV.6 Conclusion

We contribute the first study of VR painting practices among experienced artists. Our study reveals novel insights on the strengths and weaknesses of VR painting as a new artistic medium. We are excited about future possibilities to design VR painting tools that address the challenges that artists face today. While the majority of recent work has focused on helping artists achieve a better geometric accuracy while sketching – a goal motivated by previous experimental studies [19, 211] – we see a much wider range of problems to tackle and hope that our study can motivate others to start addressing those. The next chapter of this thesis explores how to design a painting system where appearance is decoupled from strokes shape and resolution while favoring editability. This project stands as a concrete example of an interesting HCI and computer graphics research problem that arises when engaging with one of the lesser-known challenges in VR painting.

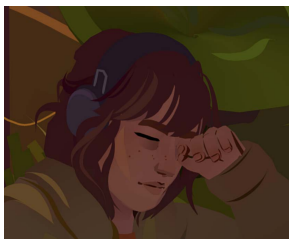
Limitations Our study was done at a small scale. Extending this study to include more participants – in particular participants that use a different VR painting software than Quill – would surely expand the range of findings, and allow for more nuance in our observations. Furthermore, we cast a wide net in terms of topics that we consider, from the overall authoring workflow down to the details of how people paint 3D shape. We hope that further studies on specific topics of interest – for example by narrowing down on some of the challenges we have uncovered – can help better characterize aspects of VR painting, and propose precise design guidelines for future VR painting tools.

Chapter V

3D Layer Compositing For VR Painting

V.1 Introduction

VR painting is a novel 3D authoring workflow that feels direct, highly controllable and easy to learn (see Chapter IV). This simplicity and directness contrasts with the traditional 3D authoring workflow, where artists need to express their idea as an assemblage of disparate representations: a complete scene is composed of 3D geometry, of a model for how light interacts with the geometry, and of a representation of lights that affect visual appearance. Many artists embrace the VR painting workflow and take full advantage of it by creating visuals that would be very hard to achieve with the traditional 3D modeling workflow, such as using many distinct brush strokes to give the artwork a unique painterly look.



While VR paintings are inherently 3D assets, their creation workflow is similar to that of traditional 2D painting (Section IV.3.1.2). Just like in traditional oil painting, VR artists simply lay down colored strokes on their 3D canvas, and these colors altogether precisely define the final painting – eg VR artist Zoe Roellin uses a single light brown brush stroke to convey the impression of her character's hair strands shining in the morning sun light (see inset). Albeit artists appreciate the level of control afforded by this workflow, the fact that a 3D stroke represents simultaneously information about the hair strand's 3D shape, about the color of the hair itself and about how light influences that color, makes paintings hard to edit after the fact [155, 285].

To overcome a similar hurdle, artists working with 2D digital painting tools have developed workflows based on *layer compositing* [204, 258] which is a feature widely available in digital painting software [3, 4, 146]. By painting on multiple layers, 2D artists can disentangle the final color of a pixel into individually editable constituents, favoring post-hoc **editability** of a painting (Fig. V.1). Bitmap layers with an alpha channel can be used as clipping masks, in order to **reuse** a carefully painted shape as a mask for less precise, broader shading brush strokes, or as a target to clip and blend in a texture image. Beyond those specific benefits, layers in digital painting are **flexible interaction primitives** that artists use for ad-hoc needs that emerge and lack explicit support in existing software, such as artwork organization and history management [222].

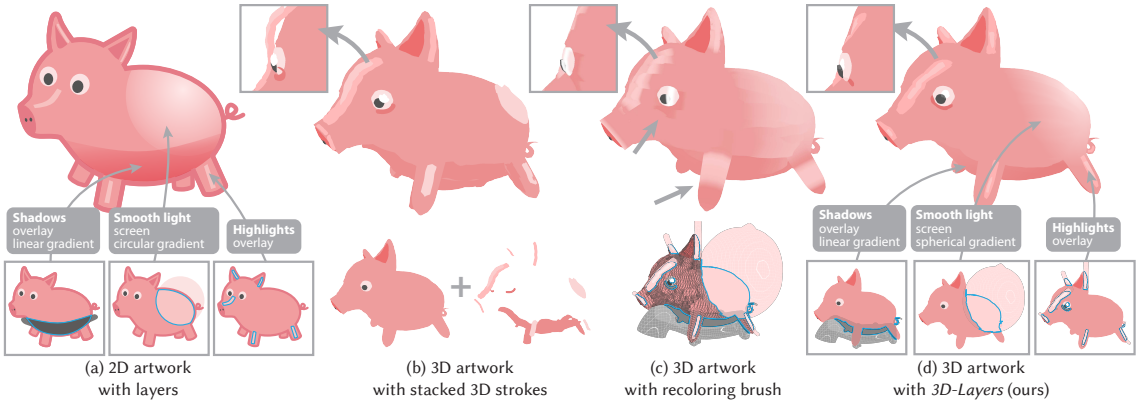


Fig. V.1: Conceptual Approaches to Layering. In 2D graphics design, artists commonly use layers to modify object colors, for instance to add shading, shadows and highlights without overriding the base colors of the artwork (a). In VR painting, artists can achieve similar visual effects by stacking strokes close to object surfaces (b). However, since it is difficult to paint at a precise depth, strokes often inter-penetrate or leave visible gaps (b, inset). Alternatively, artists can use the recoloring brush (c), a tool that recolors any stroke vertex it intersects. For visualization purposes, we display the volumetric extent of the brush and we delineate its intersection with the strokes as blue curves (c - bottom). Recoloring vertices reveals the low resolution of the stroke mesh (c - arrows). In our approach, users paint color edits in separate layers (d), offering the same flexibility as in 2D graphics design. Our rendering algorithm computes a precise intersection between the layers and the strokes to be recolored, avoiding the visual artifacts of existing 3D solutions.

Despite their ubiquity in 2D digital painting, layers are currently not supported by VR painting software [140, 289]. In the 2D case, layers are semi-transparent frames that are composited in a user-defined *z-ordering*, with each pixel from the final frame accumulating color from the corresponding pixel of all layers in the stack (see Fig. V.1)). In the 3D case, a direct parallel to this 2D case cannot be established. 3D paint strokes are 3D geometry embedded in 3D space, so to obtain a coherent 3D scene they must be rendered by taking into account *depth-ordering* – a stroke closer to the camera must occlude a further stroke. This ordering is completely determined by the position and shape of 3D strokes in space, and by the position of the camera, making it incompatible with the concept of a user-defined *z-ordering*. While it is possible to manually place strokes in 3D space such that they have the correct z-ordering from a given viewpoint (see Fig. V.1 b), it is challenging to achieve that for multiple viewpoints without creating gaps or inter-penetrating strokes (Fig. V.1 b, inset).

Existing methods to paint on 3D geometry leverage the idea of working in the 2D texture space defined by the surface of a 3D shape, in order to propose an authoring experience closer to that of 2D digital painting. But painting in texture space is not trivial both from a technical perspective and from an authoring perspective.

First, defining how the 3D surface maps to 2D texture space without introducing awkward seams and deformations is a challenging research problem [115, 279], that is made even harder in the case of VR paintings where the surface is composed of many overlapping strokes (see inset). Secondly, painting in texture space is also hard from the user's perspective. Directly painting on the 3D surface and using the inverse mapping to apply paint onto the texture is an effective solution in many cases [125]. However, it can be tedious when using a 2D input device that paints on the surface via a raycast projection due to the need to hunt for a disoccluded view [108, 234, 304], and those difficulties are heightened when using a 3D input device such as a VR controller due to the difficulty of aligning a stroke to a curved surface while freehand sketching [19, 22].

We propose a design for *3D-Layers* to provide similar benefits to artists as the use of 2D digital layers, and help alleviate some of the challenges encountered when painting in VR (Section V.3). We approach the problem of designing 3D-Layers from two perspectives: on one hand we describe 3D-Layers from an interaction perspective by defining how a user can interact with them and use them as part of their creation workflow ; on the other hand we describe 3D-Layers from a technical perspective by defining how to efficiently render 3D-Layers within a 3D painting software.

Our main contribution is the design of 3D-Layers as a novel representation of shape and appearance in 3D paintings, that helps make color edits in VR paintings more convenient, favor reuse of content and editability (Fig. V.1). We build a proof-of-concept application as a test bed for 3D-Layers (Section V.4). We demonstrate the flexibility of 3D-Layers in supporting a range of appearance editing operations by demonstrating 2 different workflows (Section V.6).

The project is still in progress at the time of writing of this thesis, and we plan to conclude it by inviting 6 experienced VR artists to use our prototype application during a user study (Section V.7).

V.2 Related work

We provide an overview of VR painting research and commercial tools in Section II.1.2, and of existing methods for appearance editing via painting in Section II.2. In this section, we first mention specific research and commercial tools for VR painting to explain how they relate to our work, then we give an overview of research related to layers in 2D digital painting.

V.2.1 VR painting tools

Layers Many commercial tools for VR painting [140, 228, 287, 289] have a "layer" panel. Layers in all existing VR painting tools to our knowledge are not layers in the way we think about them in this chapter. In existing tools, layers are merely stroke containers. Their visibility can be turned on or off in order to temporarily hide a group of strokes without deleting them. In some software (eg [287, 289]), layer opacity can be adjusted on a range from transparent to fully opaque. Adjusting opacity can be useful to make strokes from a rough sketch partly faded, in order to conserve a visual reference while focusing on a new version of the artwork. Toggling a layer opacity on or off is used by artists to preserve "checkpoints" of parts of their artwork, to explore different possibilities or to ensure easy backtracking. In Quill, layers are also used by artists to separate their artwork into semantically meaningful parts (eg a character's head, torso, arms and legs can be in separate layers), in order to facilitate further selection, batch modification or animation of strokes. In this chapter, we consider layers to be stroke containers, but beyond that, we enable layers to control how the strokes they contain are rendered – eg whether a stroke acts as a 3D shape or only as a coloring effect on other strokes.

Recoloring brush Another closely related feature in commercial VR painting software is the ability to recolor existing strokes using a "recoloring brush" [140, 289]. This feature allows artists to edit stroke colors, and the recoloring brush can control which color mix mode is used when mixing the new color with previous stroke colors. This feature gives some level of color editability and helps artists reason in terms of color mixing like in 2D digital painting (see Section IV.4). However, recoloring operates at the level of per-vertex colors, meaning that its success is dependent on the resolution of stroke discretization (see Fig. V.1) ; and it is a destructive operation, meaning that once a color is applied, it becomes impossible to recover again the base colors. Our design of 3D-Layers takes inspiration from the recoloring brush and overcomes its limitations by "reifying" recoloring commands as brush strokes [32]. Those brush strokes have a resolution that is independent of the stroke they color, and the user can edit them further at any point.

3D painting systems Previous VR or MR (mixed reality) painting systems consider strokes as tubes or ribbons of a single opaque color [18, 87, 157, 182, 260, 267]. In contrast, our prototype for 3D-Layers allows artists to precisely recolor strokes. The work of Kim et al. [170] is closely related to ours. Motivated by the observation that current VR painting software does not provide sufficient amount of control to artists over stroke coloring and color mixing, they implement a VR painting app that uses volumetric strokes to fill in 3D voxels with colors. Voxels can be recolored at any point, making it easier to recolor precisely – as long as the voxel grid is fine enough. We share the same goal, but differ from their work in representation choice: we represent strokes as 3D surface meshes, making it possible to easily integrate our technique with existing VR painting tools. Furthermore, we explore the use of stacks of layers.

While VR systems consider brush strokes as solid 3D shapes to be rendered as opaque surfaces, another line of work proposes to instead preserve the painterly aesthetic of 2D digital art by merging 2D and 3D rendering techniques: *Overcoat* [268] enables artists to paint with a 2D tablet on and near a 3D surface, by embedding their strokes in a volumetric canvas that surrounds this surface. The brush strokes can lie off-surface, and they are rendered as screen-space splats, which allows for a rough stylized look. Subsequent work has studied how to render the brush strokes with coherent ordering, since *depth-order* and *stroke-order* – the order in which strokes are drawn – are two relevant but potentially conflicting ways to order stroke fragments for compositing [28]. We take inspiration from the idea of 3D painting systems such as *Overcoat*, to achieve a painterly look of strokes by replacing traditional 3D rendering rules with variants inspired by 2D digital painting rendering. However, our solution considers the case of VR painting, where using 3D shapes as brush strokes enables a variety of workflows such as quickly modeling rough volumes with large volumetric strokes (see Section IV.3.1.2). We bring 2D rendering concepts such as layering and stroke-order rendering to the VR painting context.

V.2.2 Layers in 2D digital painting

Layers are a staple feature of 2D digital painting software (eg [4, 146]), and artists have adopted them in their workflow for a variety of manual rendering tasks such as using blending modes to paint shadows and highlights, using masks to paint on top of an existing layer while preserving its outer shape, applying gradients and applying image textures [258]. Organizing an artwork in multiple layers is also useful for non-destructive color editing operations, and past work has focused on how to perform this decomposition automatically [11, 92, 204, 305, 306]. Layers can be used for other ad-hoc operations that are not well supported in digital painting software, such as selective undo and history preservation [222]. We show that this breadth of applications of layers in 2D digital art can transfer to VR painting, as demonstrated by our example applications (Section V.6).

V.3 Challenges of depicting shape and appearance in VR painting

In this section we give a focused summary of the challenges that participants reported concerning depicting shape and appearance in existing VR painting systems. Chapter IV details the methodology of our formative study and provides the full analysis.

Spatial resolution of colors In existing VR painting tools such as Quill [289], a stroke is a tubular 3D surface with color encoded on a discretization of the surface – ie the surface is a polygonal mesh with per-vertex colors. The finite and typically low resolution of this discretization limits how finely appearance can be defined, and makes it dependent

on how strokes are oriented or arranged with respect to each other (see Fig. IV.4). Artists need to place strokes to define a 3D surface, and at the same time make sure this placement gives sufficient degrees of freedom to achieve a desired color variation along the surface.

Lack of editability Artists often need to create assets that adapt to different lighting conditions, or wish to explore multiple color keys for their piece. However current VR painting tools "bake" color edition operations to strokes, for example Quill's recolor brush changes stroke vertex colors to be a blend of the previous color with the new one. This makes global color editing operations overly complex since they need to be translated into many local operations, discourages exploration of different possibilities and prevents achieving complex color or light animation effects.

Accuracy Precise alignment of a stroke along a curved surface is hard to achieve when sketching freehand. This makes it difficult to create strokes on an existing surface, which often arises when one wants to edit the surface color – eg decal writing, patterns, drawing a sharp highlight.

Design goals Based on these observations from our formative study, we define the following design goals for 3D-Layers:

- **G1. Decouple color editing from underlying shapes**: make color edition independent of shape resolution and of how strokes are arranged.
- **G2. Non-destructive color mixing**: enable easy backtracking or global edits of the scene colors.
- **G3. Permissive edition**: relax requirement for exact 3D positioning when painting coloring strokes.

Our key insight is to differentiate strokes that affect shape from strokes that affect color in the final painting. In practice, this entails the following design choices:

- We let the user organize their strokes into these two categories by putting strokes either in **substrate** layers or in **appearance** layers. Substrate layers define the geometry of the scene. Appearance layers are organized in **stacks**, with always a substrate layer at the bottom of the stack. Any appearance layer above a given substrate acts on the color of the substrate. Each appearance layer can be edited independently of other layers in the stack (G2).
- Appearance strokes act on the substrate by coloring the intersection of the stroke with the substrate (see Fig. V.2 "Colors" layer). This operation is fully independent

of the substrate resolution (G1). We introduce a tolerance threshold (see Fig. V.2 "Decals" layer) to allow permissive intersections and reduce the need for accurate stroke placement (G3).

- Despite our separation of strokes into those that define shape and those that merely act as color modifiers, in-fine they are all strokes and can serve either purpose interchangeably. This makes it possible to reuse strokes created as shapes as color modifiers, alleviating the need to carefully redraw complex shapes when a duplicate can suffice (G3).

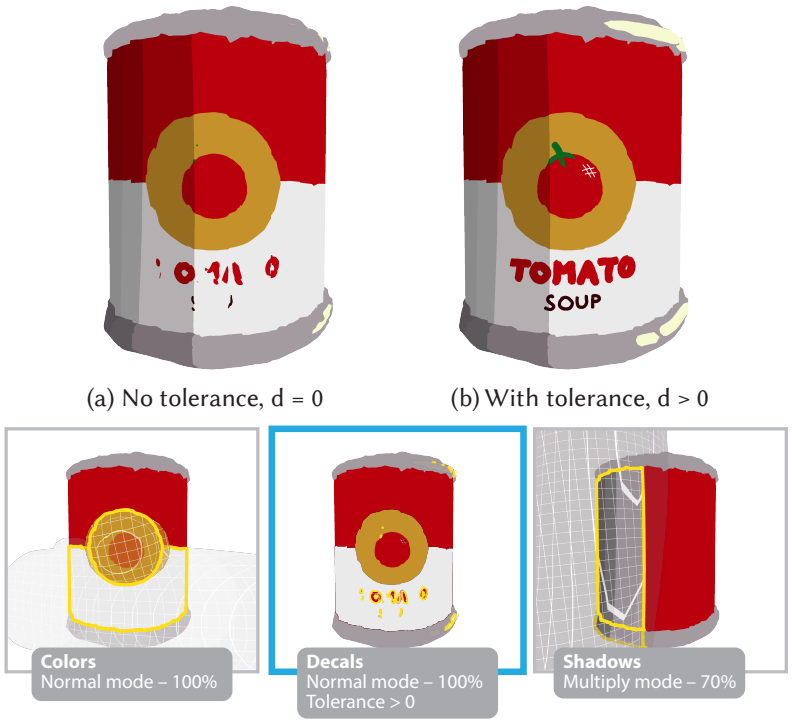
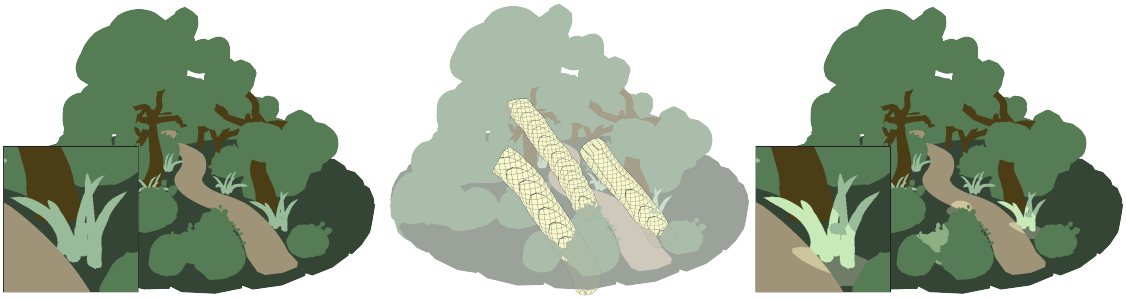


Fig. V.2: Appearance strokes only recolor the substrate strokes they intersect. But thin strokes need to be placed very close to the substrate to intersect it (a). We allow users to increase the intersection tolerance d of an appearance layer to alleviate the need for precise stroke placement (b).

V.4 Painting with 3D-Layers

We propose *3D-Layers*, a layering system for 3D paintings that exposes *layer* objects that artists can use to organize strokes, in order to help decouple shape and appearance editing, favor post-hoc editability of a VR painting, and alleviate the need for accuracy in painting. We implement 3D-Layers in a prototype VR painting application. In the



(a) Substrate layer (b) New strokes in appearance layer (c) Final result

Fig. V.3: Appearance layer. Starting from a 3D painting (a) defined as a substrate layer, an artist can paint brush strokes in an appearance layer (b), so that those brush strokes act on the color of the substrate by recoloring it inside the new strokes (c).

following section, we present the novel features that 3D-Layers enable by describing an interaction scenario. We delay discussion of implementation details to Section V.5.

Ana is a VR artist working on a 3D illustration of a path going through a forest. Ana starts by defining the 3D geometry of her scene. She uses many colored brush strokes and alternates between painting, modifying her strokes, and inspecting the scene. In this phase she focuses mainly on how the 3D shapes look, once she is happy with that, she starts using 3D-Layers to refine the colors of the scene and explore different lighting scenarios, as she is not sure yet what the mood of the scene should be like.

V.4.1 Authoring shape and color with layers

Ana decides to paint sun light sifting through the forest canopy. She creates a new layer that is applied on top of the whole scene, picks a light pale yellow color and starts painting large brush strokes to represent beams of light coming through the trees and intersecting parts of the path and forest ground.

Editing the color of brush strokes in the scene is an essential ability to quickly explore and define lighting effects. 3D-Layers enable artists to do this by creating new colored strokes on an **appearance** layer that will act on the color of strokes located on the **substrate** layer of the layer stack (see Fig. V.3). Ana can recolor large volumetric regions of the scene with a few large brush strokes. When she paints in the appearance layer, the substrate layer is colored at its intersection with strokes from the appearance layer (Fig. V.3 c).

She wants to add some hand-drawn texture details on the trees. In order to make sure her strokes only affect the tree, she creates an appearance layer on top of a new stack with the tree trunk strokes as a substrate.

To allow more localized color edits, layers can be organized in multiple stacks that apply to different subsets of strokes in the scene. This prevents undesirable leaking of a broad

(a) Stack for the tree trunks (b) Final result

Fig. V.4: Appearance layer with Threshold > 0**.** An appearance layer can be applied selectively to a subset of strokes in the scene by defining stacks. For example here the artist puts only strokes forming the tree trunk (dark brown) in the substrate layer of this stack (a). By increasing the tolerance threshold and drawing a few light brown strokes (a), the artist can quickly create a bark effect. Note that the final result (b) always renders the light brown strokes on top of the substrate strokes, despite small depth imprecisions (see insets).

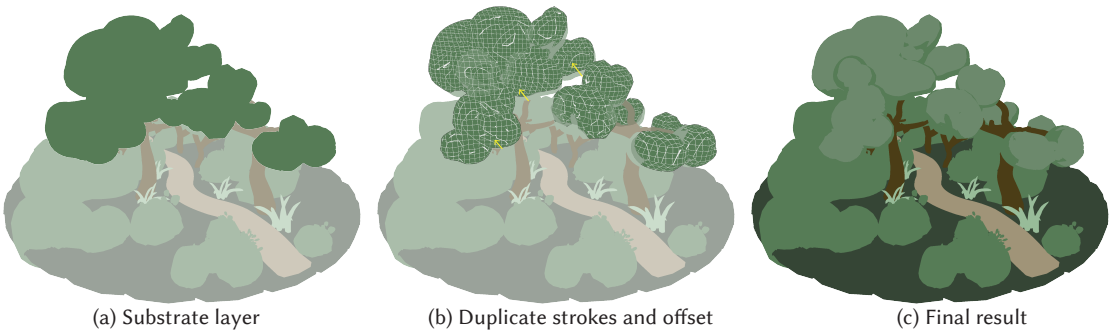
coloring operation onto nearby strokes that the artist does not want to color in the same way. This is convenient for per-object shading painting, which should not affect nearby objects.

She draws loose freehand strokes near the tree trunk to create a stylized bark texture.

Sometimes the user needs to paint thin strokes "on top" of the existing surface. For that scenario, the user can increase the tolerance threshold for an appearance layer, which will apply the color of strokes from the appearance layer to the substrate as long as those strokes lie close to the substrate within a threshold (see Fig. V.4).

Ana wants to create some color variations on the tree tops. She selects the strokes forming the tree top shape, copies them and pastes them onto an appearance layer, then applies a small position offset. This creates the impression of light hitting the tree tops, and makes their shape easier to read.

Strokes are the basic primitive used in layers for both shape and color definition, regardless of layer type – substrate or appearance – all layers are in-fine stroke containers. In practice, this means that it is possible to transfer strokes from one type of layer to another, so artists can *reuse* strokes created earlier to define shape when they later want to define color (see Fig. V.5). This helps artists get more out of thoughtfully painted shapes. A rim light effect or simple toon-like shading on a complex shape can be achieved by copying and pasting the shape to a top layer, removing the need to carefully retrace the existing shape.



(a) Substrate layer (b) Duplicate strokes and offset (c) Final result

Fig. V.5: Reusing substrate strokes for appearance editing. 3D strokes of a uniform color can be hard to parse visually (a). By copying the tree top strokes onto an appearance layer and offseting them slightly (yellow arrows, (b)), we can quickly achieve a toon-like shading (c).

V.4.2 Non-destructive coloring and effects

Ana explores the mood of the scene by coloring the whole scene with a large purple stroke in an appearance layer. She sets the color blend mode to "Overlay", in order to let the appearance layer affect the substrate colors without fully replacing them.

There can be multiple appearance layers stacked onto a substrate layer, in which case all appearance layers are blended in a user-defined *z-order* to obtain the final color. Artists can choose a *color blend mode* for each appearance layer, to achieve complex color modification effects (see Fig. V.6 b and c). For example setting a layer to screen mode and painting with a light color can simulate the effect of painting the zones affected with a lighter shade of paint than the original shade. This way of creating a highlight or shadow effect is non-destructive: it allows the artist to change the color of strokes in any other layer below and preserve the shading effect, and conversely to change the color of the effect at a later time (Fig. V.6 e).

She applies a spherical gradient on the appearance layer with the purple stroke that makes the effect fade out closer to the viewer.

Layers can also be modified with *layer masks* that affect the opacity of the layer, by setting opacity at each point in 3D space to a value sampled from a spatial gradient or sampled on a pre-defined volumetric function (see Fig. V.6 d). Layer masks enable creating both smooth gradients and high-frequency variations of color independently of stroke orientation, or the resolution of stroke geometry.

(a) Substrate layer (b) Normal blend (c) Overlay blend (d) Gradient (e) Exploring variants

Fig. V.6: Color blend modes and gradients. Starting from the substrate layer (a), painting a large purple stroke in an appearance layer has the effect to color all substrate strokes when using the "normal" color blend mode (b). Other color blend modes such as "overlay" allow quick modifications of all colors in the scene via blending (c). The artist can control the appearance layer's opacity with a 3D gradient (d). It is easy to explore variations of the scene by changing the color of the stroke to yellow and tweaking the 3D gradient parameters (e).

V.5 Implementation

We implement the proposed design for 3D-Layers as part of a prototype VR painting application. Strokes in appearance layers affect the scene by acting on the appearance of their substrate. To make this possible in practice, our main technical challenge is to convert 3D strokes from appearance layers into a color modification of substrate strokes. As showcased in the interaction scenario, this conversion must happen in real-time upon stroke creation and modification.

Strokes are represented as discretized 3D surfaces, for which many geometric approaches have been developed. Computing mesh booleans [58] can be useful to find the intersection of a large stroke with the substrate ; or polyline to mesh projection can help ensure that a thin stroke lies just in front of the substrate [22]. Instead of implementing two disparate solutions based on geometric operations, we propose to avoid altogether processing the discretized surfaces by working at the rendering stage, an approach that is also used for rendering and slicing CSG models [339].

Concretely, we implement coloring operations as custom rendering passes in a programmable rasterization pipeline. This choice enables us to achieve the real-time painting and rendering performance required of an interactive VR application for paintings of up to many hundreds of strokes (eg the desk diorama scene in Fig. V.10 has 432 individual strokes). It also allows 3D-Layers to support any kind of 3D geometry that can be rasterized, not limiting ourselves to strokes defined as polylines nor to triangles meshes. Working on a 2D buffer also simplifies supporting stroke-wise and layer-wise color blending operations.

In an upcoming publication of this work, we plan to include a more systematic benchmark of our rendering algorithm performance depending on the number of strokes and the number of layers in a scene. Furthermore, we will explore the potential of our rendering method to rasterize different kinds of primitives, such as point clouds.

In the following section we explain how strokes in appearance layers are rendered, then we show that we can deal with color blending and gradients by implementing custom rendering passes too. Finally, we give a brief overview of the authoring interface for our prototype application.

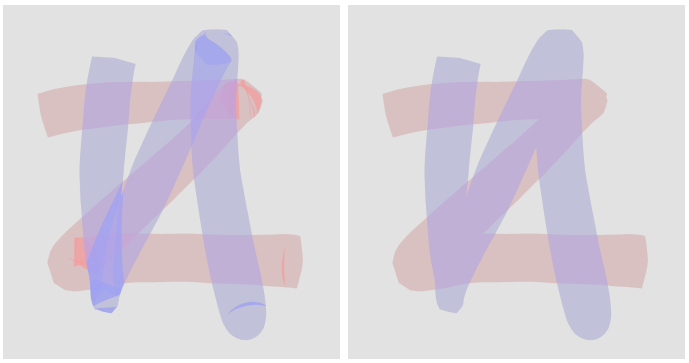
Rendering a layer stack For each stack, we first render the substrate layer. The substrate layer is rendered in a standard way, by writing directly color and depth from the strokes to the main framebuffer, and performing standard depth testing. Strokes from substrate layer are ordered based on depth. Additionally we write in the stencil buffer an identifier of the current stack, in order to use it later as a pixel-wise mask that ensures that only appearance layers from the stack can act on the color of the substrate. Then we loop through all appearance layers and render them in layer order. Since appearance layers only act on the appearance of the substrate and do not generate geometry in the 3D scene, ordering of appearance layers is independent of depth ordering and is purely defined by layer order. We render all strokes in a single appearance layer to a blank layer-wise framebuffer ; strokes are alpha-composited in stroke drawing order.

Rendering strokes in an appearance layer Each stroke from an appearance layer is rendered based on stencil and depth tests. First, a stencil test checks whether the value in the stencil buffer matches the stack identifier of the layer. Secondly, we prevent intra-stroke fragment overlap by writing and reading a 1-bit mask in the stencil buffer. This enables achieving inter-stroke alpha blending with semi-transparent strokes without encountering intra-stroke blending (see Fig. V.7). This 1-bit mask in the stencil buffer is cleared in-between strokes rendering. Other tests vary depending on the layer's coloring mode and are described in the following two paragraphs.

Coloring the intersection with the substrate We simulate coloring the substrate at the intersection between a coloring stroke and a substrate stroke. We achieve this effect in a spirit similar to shadow volumes [68] by taking the screen-space intersection of fragments that pass the depth test and fragments that fail the depth test with the substrate. This can be formulated as stencil write and test operations in two render passes, and uses only 1 bit of the stencil buffer, leaving the other bits that carry stack information untouched.

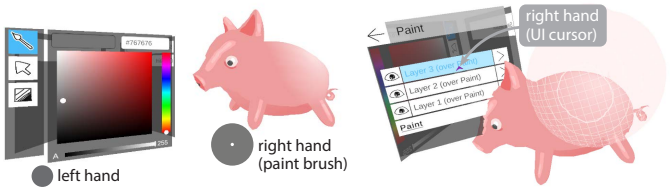
Coloring a permissive intersection Since it is difficult to paint strokes precisely near a surface, we achieve tolerance to depth imprecision by implementing a custom depth test in the fragment shader, which tests whether a layer fragment is within a depth range d of the substrate fragment.

Color blending and layer modifiers Once all strokes of an appearance layer have been rendered to the layer-wise framebuffer, we apply layer modifiers that will affect



(a) Rendering all fragments (b) Rendering maximum one fragment per stroke per pixel

Fig. V.7: We support semi-transparent strokes in appearance layers, that are composited in stroke-order within a layer. Doing so naively can cause intra-stroke fragment blending (a) when a stroke winds over itself and creates self-overlaps. We prevent this by marking pixels already covered by a stroke fragment in the stencil buffer to obtain correct inter-stroke blending with no intra-stroke blending (b).



(a) VR UI with hand-held 3D menu (b) Layer panel and visualisation

Fig. V.8: We implement 3D-Layers in a VR painting system (a) that exposes the layers' structure via menu panels (b). To help users navigate in layers, we implement a transient highlighting visualisation upon hovering the layer's name (b). Here we show the right-handed user interface.

opacity of the framebuffer content. We compute the opacity value by sampling a 3D function at the 3D position corresponding to each fragment. This 3D function can be procedural, such as a linear or spherical gradient, or it can be a predefined 3D texture. Finally, the shader copies the layer-wise framebuffer to the main framebuffer. The final color is computed by applying a color blending operation – multiply, screen, overlay.

Overall VR painting system Our prototype implements features similar to commercial VR painting software, such as stroke creation via mid-air controller gesture, selections of strokes, rigid transformation, duplication, recoloring or deletion of a selection, navigation in the canvas, undo/redo. We strive to follow design conventions from Quill – eg button mappings – since we aim to evaluate our prototype among expert Quill users. By leveraging participants' "interaction knowledge" we hope to facilitate discovery and learning of features in our prototype interface [254]. We implement basic layer organization and navigation features, such as 2D panels listing all layers in a stack with the

possibility to reorder layers (Fig. V.8 a) ; visualization of the content of a layer in-situ with a transient highlighting upon hovering a layer button (Fig. V.8 b) ; the possibility to move selected content to another layer via a drag and drop interaction. We support importing Quill paintings into our system, which enables users to re-work with 3D-Layers some of their past Quill pieces. Finally, we instrument the prototype to log all actions performed by the user during a painting session, and allow users to save and reload their session.

V.6 Workflows

We demonstrate the potential of 3D-Layers by creating 2 scenes in which we show a variety of complex effects that would be difficult to create with current VR painting tools. We focus on the creation of a particular effect in each example so we do not explain the complete creation workflow from scratch. Note that all of the examples we have made are started off in Quill, to benefit from the larger set of 3D stroke edition features, and then imported to our 3D-Layers prototype. In all examples we often refer to a stack of layers by designating it via the object present in the substrate layer of the stack. We assume that those substrate layers are pre-created for the sake of simplicity.

V.6.1 Flying over the clouds

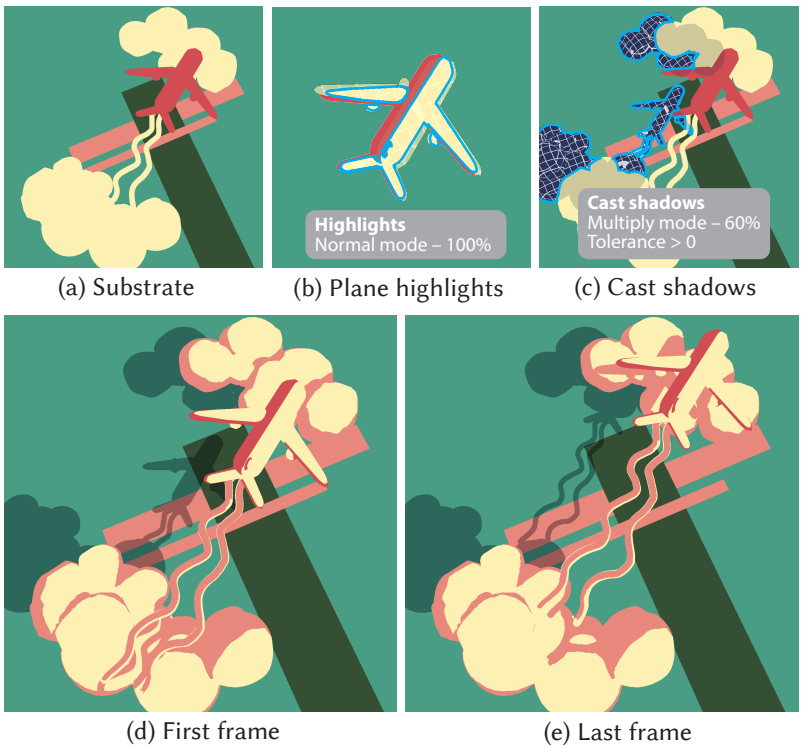
We create a VR painting of a plane flying over the clouds and the fields below, inspired by a 2D marker illustration by Priya Mistry.

Stylized shading We reproduce the stylized shading from the illustration by using appearance layers with fully opaque strokes. For the plane we quickly create a basic shading effect by copying strokes from the plane substrate layer to an appearance layer, and slightly offseting them to achieve a stylized shading effect (see Fig. V.9 b).

Shadows We create the cast shadows of the plane and of the trail on the ground by copying strokes from those objects and pasting them in an appearance layer on the ground stack. We set a non-zero tolerance in order to simulate a cast shadow (Fig. V.9 c). In this setting, the cast shadows are 3D objects too, so we can easily animate the plane moving forward and apply the same translation to the shadows (see Fig. V.9 d, e).

V.6.2 Day and night desk diorama

We create a stylized indoor scene of an office room with a desk, inspired by the dioramas from artist Viktoriia Nikitina. We propose two variations of the same scene in two lighting conditions: in the day time version the office is lit by directional sun light coming through the main window, while in the night time version the office is lit by a small desk lamp. We start by creating the day time version, then explain how we create the variation.



(a) Substrate (b) Plane highlights (c) Cast shadows

(d) First frame (e) Last frame

Fig. V.9: In this example, we duplicated the substrate strokes of the airplane and shifted them toward the sun to create a large highlight (a,b). We also used duplication to create shadows of the airplane on the clouds and ground (c). Finally, we created a simple animation of the airplane and its shadow by translating the airplane strokes along with their duplicates (d,e).

Hand-authored shadows We create stylized shading and in the same way as in the previous painting. For cast shadows of the office chair and desk, we purposefully simplify them to look like geometric shapes. Hand-authoring of shadows gives us direct control over how each shadow should look. To get the characteristic shape of the window frame grid on the light beam that hits the ground, we use a duplication of the actual window frame to an appearance layer above the layer responsible for the light beam (Fig. V.10 b).

Creating a variation Since all color changes are isolated on layers, creating the night time variation (Fig. V.10 c) can be done by hiding or deleting layers that are not useful anymore (eg, the beam of sun light on the floor). Because coloring operations are encoded as strokes, changing the desk lamp appearance from off to on is simply a matter of changing the color of the stroke inside the desk lamp from dark grey to light yellow. We also grab and move around the cast shadows from objects on the desk to better match the new light source.

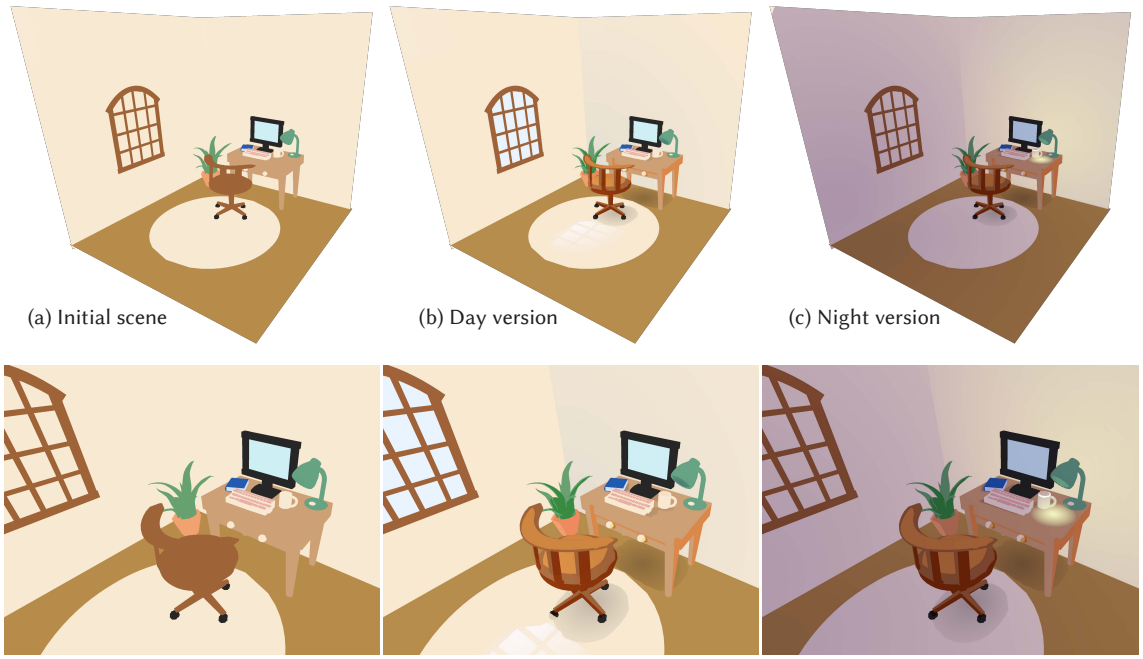


Fig. V.10: Creating variations of lighting on the same scene is facilitated by layers. We first paint the scene as if it were lit by direct sun light (b), then decide to see how the scene looks like in a night time setting (c). Both variations feature stylized shading and cast shadows.

V.7 User evaluation

At the time of writing of this thesis, we have not yet performed the user evaluation part of the project. In this section, we describe the goals of our user study, the protocol we plan to follow as well as the analysis we plan to run following the study.

V.7.1 Goals

We propose a design for 3D-Layers, a new primitive in the VR painting workflow that enables artists to use *3D strokes* as representations for both 3D shape and 3D appearance. In Section V.6, we demonstrate how as expert users we are able to reproduce a variety of effects from stylized illustrations and 3D artworks using our prototype implementation of 3D-Layers. The main goal of our user study is to evaluate our proposed design among professional and hobbyist VR artists, to test the following hypotheses:

- **H1: Usability.** People familiar with VR painting can transition easily to using 3D-Layers as part of their painting workflow.
- **H2: Simplifying shape and color authoring.** 3D-Layers make the painting workflow simpler by decoupling shape authoring and color authoring, while supporting both with the same 3D stroke painting interaction.

- **H3: Supporting exploration.** 3D-Layers support exploration of multiple outcomes by facilitating non-destructive coloring operations.

The secondary goal of our user study is to use our prototype as a technology probe [139]. We designed our prototype for 3D-Layers to be open-ended and ambiguous – a layer is merely a container for 3D geometry that applies pre-defined rendering rules to its content. Furthermore, we note that layers in 2D digital painting software and grouping primitives in 3D modeling are often used for ad-hoc purposes such as versioning and managing visual clutter during focused tasks. We hope that 3D-Layers can also be used for unforeseen tasks beyond shape and appearance creation, inspire users to try new workflows or trigger discussions about their needs for use cases not covered with our prototype.

V.7.2 Protocol

Participants We recruit 6 participants that have a minimum of 1 year experience working with VR painting, and at least a beginner level of working with Quill painting tools. Participants are invited for a two-hour study, and compensated for their time.

Overview We run the study remotely by sending an executable of our prototype to participants – each participant is expected to have access to a VR headset by their own means for the duration of the study. The participant connects to an experimenter via a video call. They are invited to answer a short demographics questionnaire, then they watch a video tutorial explaining the main functionalities of our prototype – both the basic painting functionalities and the specific features we implemented around 3D-Layers. The participant completes a first task for an approximate duration of 30 minutes, then they are allowed a 5 minutes break and are asked to answer a questionnaire about the usability of our prototype. The participant is then prompted to start the second task, which can last up to 1 hour. Since this task is open-ended, the participant is free to take breaks whenever they wish, and we encourage them to do so every 15 minutes. During both tasks, the participant is encouraged to think-aloud and specifically voice anything that they find confusing. An experimenter is available to answer questions or repeat tutorial instructions upon prompting. Our prototype application logs all user actions during painting sessions, and saves final paintings. An experimenter also notes down any observed confusion or frustration, in order to cater the final interview questions to specific events from the study. After the second task, we conduct a semi-structured interview with each participant.

Starter paintings Our challenge in the user study is to have participants interact with the prototype in tasks that are representative of real artist workflows, while keeping the study relatively short and focused on the novel features we provide. Since our prototype

Fig. V.11: Guided task scene. In the guided task, we ask participants to work on the forest scene (left), in order to approximately match the look of a target (middle). Once this is done they have to make changes to make the scene appear to be happening at dawn, for example (right).

lacks some of the painting and modeling features that a mature software like Quill might have, we alleviate the need for advanced modeling features by bootstrapping all tasks by providing a "starter painting" that each task is based on. A starter painting is a simple blockout of the final painting, that is neutral in terms of lighting conditions. We ask participants in advance whether they would like to use one of their own Quill painting as a starter painting for the open-ended creation task. If they wish to do so, we coordinate in advance with them in order to import their artwork in our prototype.

Task 1: guided task Participants are given the starter painting for the forest scene (Fig. V.11 left). We also give a target prompt that they must reproduce approximately (Fig. V.11 middle). This target is given in the form of a picture frame that we import in our VR prototype, so that participants can see it while they paint. When participants are happy with their result, they are asked to edit the painting to match a new prompt that is given in the form of a text prompt from an hypothetic client: "I would like the scene to look more like it's happening at dawn." We encourage participants to finish both tasks within 30 minutes.

Task 2: open-ended creation task Participants start this task with a starter painting that is either their own painting or by choosing a painting among a few provided default starter painting. Participants are encouraged to paint using our prototype to improve or edit the starter scene however they prefer within a maximum of 1 hour.

V.7.3 Analysis

We plan to include both a quantitative analysis of usage logs, as well as a qualitative analysis of participant thoughts and feedback.

In the user study, we do not explicitly compare our system against a baseline. However, since our prototype consists of an augmentation of a standard VR painting user interface by the adding the possibility to work with 3D-Layers, we note that it is in fact possible

to use our prototype in a way that completely ignores the features related to 3D-Layers. In practice, this can be done simply by painting all strokes in a single substrate layer. Therefore, our analysis focuses on looking at how each participant uses the prototype itself. We expect to find patterns in usage that enable us to test our initial hypotheses.

For example, we plan to analyze how much participants painted in appearance layers as opposed to in substrate layers. This can show whether participants are able to use both, which provides evidence towards testing hypothesis **H1**. Furthermore, we argue that if participants choose to paint strokes in appearance layers, it can be an indicator that layers are useful in simplifying the authoring workflow – hypothesis **H2**. We also log stroke duplication actions, which we can quantify across participants and tasks. We can further isolate stroke duplications with a layer switch – eg reusing a substrate stroke in an appearance layer. A significant amount of duplicated strokes means that the duplication feature – combined with 3D-Layers capabilities – contributes to diminishing the effort required in repainting strokes (**H2**). Finally, the second part of task 1 – editing the painting to match a prompt – aims specifically to probe how well our prototype supports edits and exploration of multiple options (**H3**).

We plan to analyze transcripts of the post-study interviews to collect more fine-grained feedback on how participants felt about using 3D-Layers, and to report any limitations that participants encountered.

V.8 Conclusion

VR painting is a novel medium for 3D content authoring, that plots an alternative path to current industry-standard authoring pipelines. Investigating novel digital representations for 3D shape and appearance that better match this authoring workflow has the potential to bridge the feature and capability gap between this emerging medium and well-established authoring pipelines. We hope this can help push VR painting to a new level of adoption by artists and foster the emergence of artistic styles not previously achievable with industry-standard pipelines based on surface modeling, texturing and material editing. In this chapter, we demonstrate that taking inspiration from established interaction paradigms in another authoring workflow – in our case 2D bitmap layers in digital painting – is a way to design new digital representations of content that artists easily relate to based on past experience. We augment the VR painting workflow by simplifying the shape and appearance authoring process, allowing reuse of content, and supporting exploration of possibilities. We will deploy our prototype among 6 experienced VR artists to get insights into how 3D-Layers impacts their workflow. In the meantime, preliminary pilot studies and extensive use of our prototype by myself to create the paintings from this chapter yielded promising results in the potential and usability of our design.

V.9 Future work

Exploring the lighting authoring spectrum With 3D-Layers, we embraced a design where lighting effects – shading, cast shadows, atmospheric ambience – are all hand-authored. This is in line with one of the workflows we have observed among experienced VR artists (Section IV.3.1.2). Such hand-painted lighting is appreciated by the artists we interviewed because it affords a high level of control over the final result, enabling artists to infuse art direction choices and their personal style. On the other end of the lighting authoring spectrum, shading and shadows are defined programmatically by specifying environment and material parameters for each 3D shape. A rendering engine interprets the whole scene and generates images based on programs that define rules on how material and shape react to light from the environment. Such automatically computed lighting is very powerful to quickly explore a variety of visual results by tuning a small number of semantically meaningful parameters. Going forward, we would like to explore more lighting authoring workflows that sit somewhere in the middle of these two extremes, to design authoring tools that combine the expressiveness of hand-painted lighting with the flexibility and power of programmatic lighting. In particular, such an authoring tool could be based on converting appearance editing operations performed by hand-painting into a programmatic representation that can be modified by adjusting meaningful parameters [230], animated by interpolation [244], or applied to a different 3D shape [103]. We plan to investigate how such ideas fit in the 3D-Layers framework, for example finding how hand-painted shading effects can be mapped from one substrate layer to another, how to "link" edits of appearance layers via common properties [328], or how to programmatically modify strokes of a hand-drawn appearance layer to match a desired change in lighting setup.

Layer navigation user interface in 3D Our prototype VR application features a basic 2D panel user interface to navigate layers, select a layer or adjust layer parameters. However, previous work on 2D selection has shown that the task of selecting a given layer in a large stack of layers can be hard [281]. Preliminary tests with our prototype confirm their observations and moreso, we think selecting a target layer – eg finding which layer acts on the color of a given 3D point – might be even harder than in the 2D case. Some challenging factors include our multi-stack organization, the large potential amount of overlapping strokes at a single point in 3D space, and the fact that VR editing is done in a 3D UI. In a 3D UI, performing a selection task while staying aware of the larger visual context of the piece – described as a "divided attention problem" [126] – is even more challenging than in the 2D case because the visual context occupies the full viewport at all time. Therefore a selection modal window or panel in 3D space has to be designed and placed thoughtfully so as to be informative without occluding content. We hope to take inspiration from existing research in 2D layer navigation [251, 281, 321] and VR user interface layout optimization [56, 97] to improve our user interface design for 3D-Layers. More broadly, we hope such work can help design interactions for 3D

selection tasks in VR that support complex yet realistic scenarios in 3D authoring tasks, such as the presence of many shapes that occlude each other.

Digital representations for VR painting We decided to base our design of 3D-Layers on strokes that are represented digitally as polygonal meshes with per-vertex colors, and we define strokes in substrate layers as being fully opaque surfaces. In the future, we wish to explore how novel representations of 3D shape and appearance based on volumetric fields or on point clouds can be used in VR painting. Previous work has shown the potential of casting VR painting as editing voxels in a volumetric canvas [170] or as a series of edits of a signed distance field [98]. Using point cloud splatting is key in the rendering engine of *Dreams* to achieve a unique painterly look [98]. State-of-the-art real-time rendering techniques now make it possible to render large clouds of 3D colored gaussians, that can represent well semi-transparent surfaces [168]. Adopting a representation such as 3D Gaussians for VR painting can open new possibilities in terms of editing operations – 3D points can be freely grabbed and moved or deleted without needing to manage mesh topology ; and new possibilities in terms of aesthetic appearance – semi-transparency and color mixing *across* objects can enable new visual styles such as the equivalent of a painterly watercolor look in 3D. We are excited about possibilities to adapt this rendering technique originally designed to render realistic 3D scenes captured in the real world and consider it instead as the backbone for a VR painting tool. We see potential research questions in pursuing that, such as how to design creation and edition operations inspired from 3D painting and direct manipulation ; and how to create painterly splats in 3D, for example using 2D or volumetric textures, or applying procedural 3D effects.

Chapter VI

3D Scene-Aware Hand-Drawn Animation on Videos

VI.1 Introduction

We have shown in previous chapters that by letting artists work with 3D strokes as a digital representation, we can explore new authoring workflows for 3D shape and appearance. In this chapter, we show how 3D strokes can enable new workflows for hand-drawn animation. In particular, we study the creation of *video doodles*.

Video doodles are an emerging mixed media art that combines video content with hand-drawn animations to produce unique and memorable video clips. Adding animated drawings and annotations to videos is an effective way to emphasize actions and motions of characters in the scene, to create visual explanatory material for educational purposes, or simply to make mundane videos more fun, personalized and attractive.

Artists typically create video doodles by drawing 2D animations frame-by-frame, using the video as an underlay. The main difficulty faced by artists following this manual workflow is to make the drawn content interact convincingly with the video content. Let us consider the example of adding a few animated "doodles" to a tramway video (see Fig. VI.1). When drawing a face over the tram, the artist must make sure that the eyes and mouth *follow* the tram as it moves across the frame. However, simply translating the corresponding doodles may not be enough, as their 2D scale, skew and orientation need to change to reflect the deformations caused by *perspective projection* as the tram comes closer to the camera and makes a turn. Even when doodling a static background element, such as the rainbow bridge across the rails, camera motion can yield non-trivial trajectories in the image plane, which the doodles must follow to appear fixed to the scene. Other difficulties arise in the presence of *occlusions*, which greatly contribute to the perception of 3D layout. For example, the tram should occlude part of the bridge to give the impression that it passes below it. To summarize, effective video doodles require the hand-drawn content to be *scene-aware*, meaning that they appear as if they were embedded in and synchronized with the content of the 3D scene captured in the video.

While computer vision methods can assist in tackling this challenge, existing solutions fall short in terms of accuracy, control, and accessibility to novice users. Professional video editing software offers a plethora of tracking algorithms to insert content that

Fig. VI.1: *Video doodles* combine hand-drawn animations with video footage. Our interactive system eases the creation of this mixed media art by letting users place planar canvases in the scene which are then tracked in 3D. In this example, the inserted rainbow bridge exhibits correct perspective and occlusions, and the character's face and arms follow the tram as it runs towards the camera.

follows video motion [6, 43, 106]. But these algorithms come with numerous parameters to tweak, and are prone to errors and drift in the presence of occlusion or in areas that lack reliable image features. Furthermore, tracking algorithms offer limited support for 3D motion estimation and are not sufficient to simulate occlusions between the inserted drawings and the video content. Artists reproduce such effects by decomposing the video into several layers using keyframed binary masks and rotoscoping.

In this chapter, we start yet again with the observation that current representations of content that artists work with are not well suited to the task at end. Despite the final artifact being a succession of 2D images, working in the 2D image plane to create animated doodles that move in the 3D scene space introduces a high *semantic distance* [138] since artists must in fact envision how the intended 3D motion projects to the image plane.

An alternative and perhaps more useful representation of the video is to see it as the capture of a 3D scene with an additional time dimension. Computer vision methods can interpret the video as such by estimating the camera poses and the – partial – geometry of this scene [37, 178, 274, 344]. Such a reconstruction can be used as a starting point for a 3D modeling and animation workflow, to insert virtual objects into the scene and re-render the video. Defining 3D motions of doodles can be done in 3D space, and the re-rendered frames will have accurate perspective and occlusions. Unfortunately, introducing 3D animation into the video editing workflow is not much of an improvement in *directness* for most users, due to the added *articulatory distance* [138] introduced when performing 3D manipulation in 2D user interfaces.

Instead, we design an interface where users can sketch and manipulate doodles in 2D as if in a traditional 2D animation tool, yet behind the scenes our algorithm for positioning and animating doodles leverages knowledge of the captured 3D scene. Animated doodles are composed of strokes, and we place those strokes in the 3D scene by embedding them onto *planar 3D canvases* that can be positioned and animated in 3D. Our system takes as input casually-captured videos, which we preprocess with recent computer

Fig. VI.2: At the core of our interactive system is a novel tracking algorithm that deduces the 3D position and orientation of a planar canvas over an RGBD video given a few keyframes (green denotes a position keyframe, red denotes a position and orientation keyframe). Note how the canvas rotates to align with the direction of the trajectory and gets occluded by the body and the poles. Users create scene-aware doodles by drawing over the canvas in a simple 2D interface.

vision methods to obtain per-frame cameras [274], optical flow [309], and dense depth maps [178]. We leverage this geometric information to anchor these canvases to 3D points in the scene. Our system automatically renders the doodles with convincing *perspective* and *occlusion* effects (Fig. VI.2). We further augment the canvases with a dedicated tracking algorithm, such that the 3D position *and* orientation of the canvases follow the arbitrary moving objects they are anchored to. Importantly, we let users control this algorithm by keyframing the canvas position and orientation in image-space, effectively hiding most of the complexity of the underlying 3D representation. Our optimization then solves for the canvas 3D trajectory and orientation that best follows the scene motion while being constrained by the keyframes. Combined together, our technical and user interface contributions enable even novices to turn their videos into convincing *video doodles* for a variety of applications, ranging from fun posts on social media to engaging illustrative tutorials.

We published supplementary materials online, the reader can refer to them at the following URL: `https://ns.inria.fr/d3/VideoDoodles/video_doodles_supplemental_webpage`.

VI.2 Related work

We first discuss professional software for video editing as well as research work that aims at augmenting video editing by leveraging 3D geometry and motion estimation. We then discuss methods for point tracking, along with the user control they offer, as such tracking is at the core of our *VideoDoodles* system.

Professional video editing software. VFX artists and professional motion designers can choose from a rich array of powerful tools to insert animated virtual objects in captured scenes [6, 37, 43, 106, 262]. In particular, 3D camera estimation [6, 37] and object tracking [106, 167] are now mature technologies that accommodate many special effects, although these professional tools have a steep learning curve and sometimes require planning the shot at the time of capture – e.g. by placing markers for better tracking.

Planar tracking [43] is closest to our goal but is limited to tracking planes present in the scene, while our scene-aware canvases can move and orient differently from the surface of objects they are anchored to.

Leveraging 3D geometry. Automatic 3D computer vision methods hold the potential of bringing advanced editing tools to casual users, including re-rendering photos and videos with different camera properties [174, 202], stabilizing complex camera trajectories [176], creating 2.5D parallax effects [177]. Closer to our target application are systems that leverage 3D pose and geometry estimation for stylizing moving objects [290] or for drawing over their surface [163, 253]. But these systems do not allow drawing *away* from existing surfaces, which is important to produce a variety of effects, as in Fig. VI.1 where the bridge is drawn above the ground, and the arms are drawn around the tram.

A large body of multi-view 3D reconstruction algorithms, including Structure-From-Motion [274] and SLAM [221], can be used to find reliable sparse 3D scene points and camera poses, but these methods are designed only to reconstruct static components of videos. Furthermore, sparse reconstruction methods could support some features of our approach, but occlusion requires per-pixel geometry, which these methods do not produce. Recent work has focused on estimating a consistent *dense* depth representation from videos with dynamic elements [178, 207, 344]. These methods were applied to virtual object insertion, but their demos were created by loading the 3D reconstruction in professional 3D rendering and compositing software such as *Blender* or *Nuke*. Our approach uses the video depth reconstructed by these methods to track, orient and render drawing canvases in the scene, allowing users to insert scene-aware doodles with a simple 2D user interface.

Inserting virtual objects into captured scenes is also key to augmented reality applications [16, 88, 314]. In such applications, users typically place virtual objects in the scene through in-situ direct interactions [188], by placing tracking targets on objects [198], or by selecting pre-defined targets – e.g. detected planar surfaces [16]. Our system is designed to be used as a post-processing video editing tool rather than as an in-situ augmented reality tool. This positioning allows us to offer more precise and expressive, keyframe-controlled trajectories for the inserted doodles.

Leveraging scene motion. Our approach draws inspiration from prior work that leverages motion tracking to augment videos with hand-drawn annotations and with direct navigation along motion trajectories [86, 116, 117, 227]. Subsequent work by Suzuki et al. [302] allows the insertion of responsive sketches that follow or react to tracked motion. We extend this family of work by enabling more direct user control over the tracking results through keyframing of both position and orientation of drawing canvases, and by accounting for the 3D geometry of the scene over which the doodles are drawn.

In human-computer interaction, many approaches leverage human pose estimation and tracking [55] to offer dedicated visual effects and visualizations [217, 266, 342]. *PoseTween* [201] is a system for animating drawings over human action videos which relies on human pose for 2D interpolation of user-provided keyframes. In contrast to these domain-specific solutions, our approach tracks 3D trajectories of arbitrary objects and supports 3D effects, including perspective transformations and occlusions of the inserted doodles.

Tracking arbitrary points in videos. While tracking bounding boxes [128], segmentation masks [229], planar regions [197], or semantic keypoints for objects of a known class [330] has been studied thoroughly, the more general problem of tracking arbitrary surface points across a video has received surprisingly little attention, as stressed by the recent *TAP-Vid* benchmark for long-range point tracking [83]. Aiming at helping novices to create cartoon animations, *Live Sketch* [295] guides a point tracker using keyframing to extract motion from a video clip and transfer that motion to a drawing. Their algorithm follows prior work that casts user-guided point tracking as an optimization where the point trajectory corresponds to the shortest path in a directed graph formed by the video pixels [13, 51]. This formulation was also used by Doersch et al. [83] to annotate ground-truth trajectories for the *TAP-Vid* benchmark. We extend this formulation to the case of *3D* point tracking by accounting for the depth and camera pose at each frame. Furthermore, we leverage the resulting trajectory to also orient the 3D canvas according to the direction of the moving object. Finally, rotoscoping algorithms track curves along contours in a video, guided by user-provided keyframes [8, 194]. These algorithms rely heavily on the smoothness and contrast of image contours, while we focus on tracking points that may lie in feature-less regions.

VI.3 Challenges in video doodles authoring

We investigated current video doodling practice by surveying 20 online tutorials (T1-20, see complete list in supplemental materials) and by discussing the most common techniques with two professional motion designers (P1 and P2) having 17 and 15 years of experience with 2D motion graphics tools (*Adobe After Effects*, *Adobe Character Animator*), and animating with code (*Processing*, *CSS*).

2D animation workflow. The majority of tutorials we found [T1-13] describe a process akin to traditional 2D animation. The artist imports the video in an animation software and proceeds to draw on one or multiple overlaid layers, for every frame of the video. To streamline this workflow, artists can copy and transform the drawings from one frame to the next, or can rely on smooth interpolation of the drawn strokes between sparse keyframes [T11,12]. This is a difficult process that requires significant manual tweaking: *" [It'd be] probably a lot of changing stuff on a per frame basis to make sure it catches up*

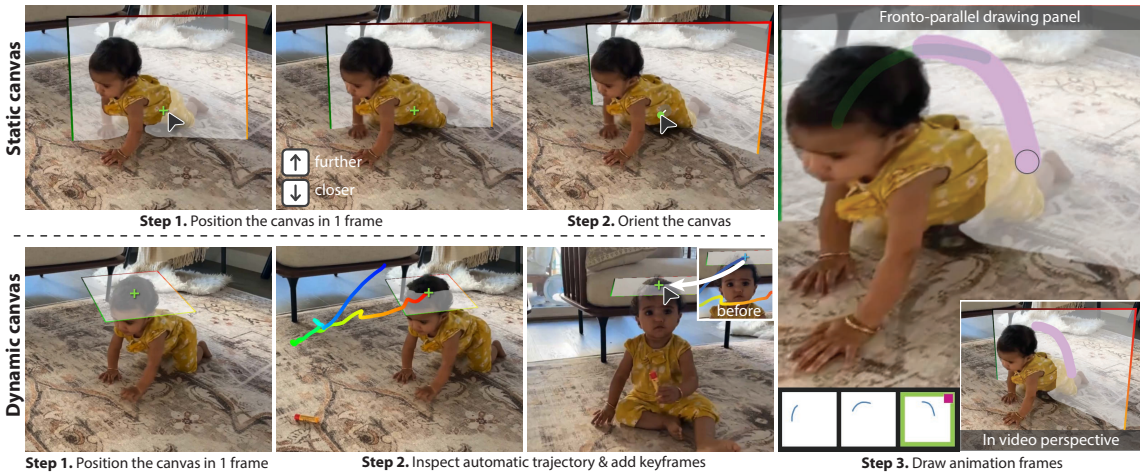


Fig. VI.3: Main features of our user interface. The user can place a static (top) and dynamic (bottom) canvases in the scene by dragging their centers (green cross). Canvases can be moved closer or further away from the camera, and can be oriented using a gauge figure. Dynamic canvases are placed in one or more keyframes, from which our system deduces a 3D trajectory (time is color-coded with a jet ramp). The user can then draw doodles in a dedicated panel where the canvas is rectified to be fronto-parallel, alleviating the need to draw in perspective.

properly, you'd need to just be like, on this frame, you're here, and change the drawing. " (P2)

Dealing with occlusions requires either erasing occluded parts of the doodles [T2], or creating binary masks that follow the occluding shape across the video [T12,13]. Another major challenge resides in synchronizing the drawings with events in the video. Artists achieve such synchronization by keeping the video visible as an underlay to provide visual context, and by taking notes on the precise timing of key events to plan the animation [T1].

Motion and camera tracking. Artists rely on experience and intuition to draw doodles such that they appear to have the correct motion and perspective in the scene, which can be challenging even for simple camera motions: *" Honestly that's a trial and error process. It's me saying OK, let me try this position keyframe and see if [this] looks interesting like this and if not, I'll just keep on doing it until I get something that I feel looks right. " (P1)*

To ease this task, artists are faced with diverse tracking tools to pick from.One-point tracking [T14,20] is easy to set up but does not take orientation or perspective changes into account, whereas 3D camera tracking [T16] is well suited to place static elements in the scene but cannot track moving objects. When using these algorithms, artists often correct the inferred trajectories by deleting erroneous parts and replacing them with interpolated keyframes [T15], by providing corrective keyframes to the algorithm [T17-20], or by tuning tracking parameters [T17-19]. *" If at a certain point I move too fast*

or something happens, it's blurry and the tracking goes off and then it's all over the place, this little tracker. So then I have to do some manual edits and put it back and just make sure it works OK. And yeah, it's just a little bit cumbersome. " (P1)

Due to these challenges, both professional designers we interviewed judged that it would take them several hours to create a video doodle of a few seconds using the tools they are familiar with.

Design goals. Based on the above observations, we define the following design goals for our interactive *VideoDoodles* system:

G1. Scene-aware doodling: Maintain a 2D animation workflow, yet streamline the creation of perspective and occlusion effects.

G2. Flexible motion tracking: Support the tracking of both the moving camera and moving objects in the scene.

G3. User control: Offer control on the motion and timing of the doodles using the established interaction paradigm of keyframing.

VI.4 User workflow

Our system follows the principles of *mixed-initiative* user interfaces [134] as it leverages 3D computer vision to offer significant value-added automation, combined with simple 2D interaction mechanisms to enable amateurs to efficiently guide and refine the end result. We first describe a typical interactive session with this system, illustrated in Fig. VI.3. We provide a recording of this session in the accompanying video, along with additional animated results. We detail the algorithms behind our system in Section VI.5.

Input. Our system takes as input a video clip representing a single camera shot. In a preprocess, we augment this input with per-frame camera pose, depth map and optical flow (Section VI.5.1).

Planar canvases. We fulfill our first design goal (**G1**) by embedding the doodles into *planar 3D canvases* that are placed in the scene via 3D rigid transformations. On the one hand, users can easily draw strokes on planar canvases using a 2D interface. On the other hand, we can render these canvases with correct perspective and occlusions in all video frames thanks to the estimated camera poses and depth maps. While this simple mental model cannot represent non-planar curves, advanced animation effects can be achieved by animating the strokes within the canvas, as in traditional 2D animation. Many sketch-based modeling systems rely on similar canvases to lift 2D strokes to 3D [23, 36, 54, 85, 188, 196].

Placing a canvas. Our interface allows users to place *static* or *dynamic* canvases in the video. Static canvases have a fixed position and orientation in the scene, and as such only react to camera motion (**G2**). Users place static canvases by dragging and rotating them over one of the video frames, and by optionally adjusting their scale relative to the scene. Our system automatically sets the center of the canvas such that it lies at the same depth as the underlying surface. Users can over-write this default depth using a slider.

Dynamic canvases follow moving objects in the scene (**G2**). To obtain such tracking, users position and orient the canvas in at least one keyframe. Our system then infers a 3D trajectory that follows the scene point under the center of the canvas across the video, and it orients the canvas relatively to the direction of that trajectory. If needed, users can refine the result by adjusting the position and orientation of the canvas in additional keyframes (**G3**). By default, users only need to specify the 2D position of the canvas in each keyframe, and our algorithm takes care of deducing the depth that yields the best tracking in 3D. As with static canvases, users can over-write the inferred depth using a depth slider. Since our system embeds the canvas in 3D, the canvas automatically appears bigger or smaller as it moves in depth, without requiring an explicit keyframing of scale by the user.

Drawing on a canvas. Canvases can undergo significant foreshortening depending on their 3D orientation, and prior studies have shown that even experienced artists struggle to draw accurately over slanted surfaces [270]. Our interface avoids drawing in perspective by providing a secondary drawing panel, where the canvas is displayed under an orthographic, fronto-parallel view (**G1**). However, drawing on a blank canvas would make it difficult for users to align and synchronize their doodles with the video content. Our interface provides the necessary contextual cues by rectifying the portion of the video covered by the canvas and by displaying it as an underlay in the drawing panel. We help users find a suitable frame to draw onto by reporting the amount of foreshortening of the canvas for each frame of the trajectory, less foreshortened canvases yielding less distortion of the video after rectification. Furthermore, our interface directly shows occlusion effects as strokes are drawn to help users assess the 3D insertion of their doodles. Finally, we provide a basic frame-by-frame 2D animation tool within the drawing panel, which allows users to create simple loops that are repeated along the video. This tool also includes a simple *onion skinning* feature – displaying previous and next frames as semi-transparent overlays – to facilitate drawing animated sequences.

VI.5 Algorithmic Components

At the core of our system is a tracking algorithm that leverages camera and scene motion, depth estimation, and user-provided keyframes to find the 3D trajectory of a scene-aware canvas.

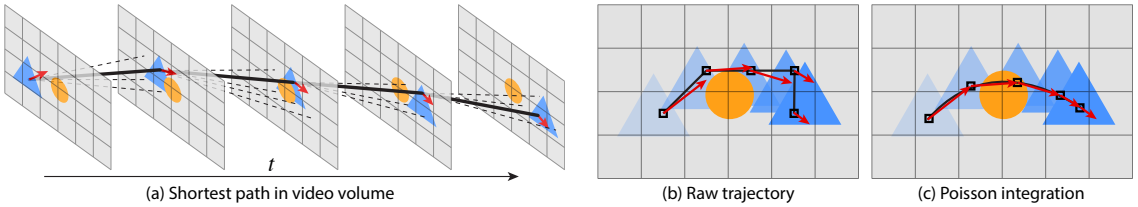


Fig. VI.4: Schematic illustration of our tracking algorithm. We extract an initial trajectory as the least-cost path in the directed graph connecting each keyframed pixel to similar pixels in consecutive frames (a). This trajectory often jitters over the object to track due to occlusions, lack of visual features, and our use of a low-resolution graph (b). We recover a stable trajectory by integrating the scene flow sampled along the initial trajectory at full resolution (c, red arrows).

VI.5.1 Pre-computing depth and motion

Our method is built upon 3D video reconstruction; given a set of input frames, per-frame camera poses and world-aligned depth maps are computed. We use our own re-implementation of *Robust Consistent Video Depth Estimation* [178]. This approach first estimates camera pose and a projection matrix for every frame using COLMAP [274]. It then uses a deep single-image depth predictor to compute scale and shift invariant depth maps, and solves for a geometric optimization that aligns these depth maps into consistent world coordinates, yielding a dense, temporally consistent geometric reconstruction. This approach internally uses optical flow [309] computed between consecutive frames, which we also save for later use.

Equipped with the camera matrix and depth map for each frame t, we compute for every pixel p_i^t its 3D position P_i^t by unprojection. Similarly, we lift the optical flow vectors v_i^t to 3D to obtain scene flow vectors V_i^t.

VI.5.2 Keyframe-based tracking

Given one or more user-specified canvas keyframes (consisting of a position and orientation), our goal is to recover a trajectory that

- Makes the canvas follow the scene point it is attached to, such that users do not have to reproduce that 3D motion by hand.
- Interpolates between keyframes, such that users have full control and can (optionally) deviate from motion tracking if desired.

We cast this problem as a series of optimizations, where the variables are the 3D positions and orientations of the canvas in each frame, the keyframes are expressed as hard constraints, and the motion tracking is expressed as soft objectives to allow for correction by the user. We first detail how our approach tracks 3D positions, and then explain how we extend it to additionally track 3D orientation.

Tracking 3D positions. Our method builds upon related keyframe-based 2D tracking algorithms that search for trajectories with coherent appearance and motion within the space-time video volume (Fig. VI.4a) [13, 51, 83, 295]. These methods build a directed graph that connects each pixel in frame t to every pixel in frame $t+1$, and assigns each edge a weight that is proportional to the difference in appearance and position between the two pixels [13, 295], or to the agreement with pre-computed optical flow motion vectors [83]. Frames with a keyframe have only one node in this graph, corresponding to the pixel specified by the user to be the center of the canvas, which forces the trajectory to adhere exactly to the constraints. All nodes at the first frame and last frame are connected to a super-source and sink node respectively, and the trajectory is computed as the shortest path from source to sink.

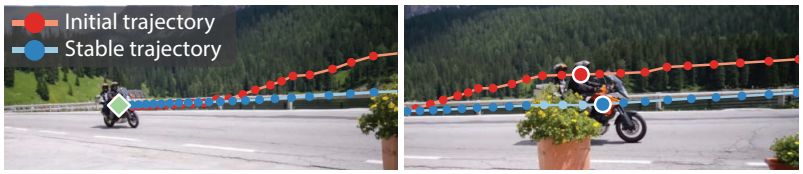
We extend this family of algorithms to leverage the 3D information we extracted during preprocessing, and to generate a 3D trajectory. Specifically, we encourage the *scene-space* trajectory to align with the scene flow by expressing the edge weight between pixels in consecutive frames as

$$w\left(p_i^t \rightarrow p_j^{t+1}\right) = \left\| \left(P_j^{t+1} - P_i^t\right) - V_i^t \right\|^2 . \tag{VI.1}$$

Computing this weight as a 3D distance prevents the trajectory to jump between objects that lie close together in image space yet are far apart in depth. We improve the robustness of this formulation by removing nodes (pixels) that are too close to motion or depth discontinuities, as detected by computing the agreement between forwards and backwards optical flow for the former, and by computing the gradient of the depth map for the latter. This safeguard further reduces the risk of crossing object boundaries.

Building the graph over the entire video volume would be prohibitive. We drastically reduce complexity in two ways. First, we trim a large part of the graph by augmenting each pixel with an appearance vector a_i^t – which we compute as a set of deep visual features [149] – and by only retaining, for every frame, the 10% pixels most similar to the keyframes in appearance. This trimming also helps the tracking algorithm to focus on the region of interest. Second, we reduce the size of the graph by working at the resolution of the deep visual feature maps, where one pixel corresponds to a patch of 10×10 pixels in the original video. With these settings, building the graph and finding a shortest path with Dijkstra's algorithm for 80 frames of a 1200×674 video takes around 3 seconds on a 2.4GHz Intel i5 MacBook Pro with 16GB of memory.

Recovering stable, high-resolution trajectories. Prior methods directly output the nodes of the shortest path as the tracking trajectory [13, 51, 83, 295]. However, this initial *discrete* trajectory often suffers from jitter and drift over featureless surfaces or in the presence of occlusions, which are exacerbated when working with a low-resolution graph (see Fig. VI.5). We correct for such instability by observing that large portions of objects often move coherently, such that neighboring pixels have similar motion vectors.



(a) 1 keyframe at clip start (b) Trajectory after occluder

Fig. VI.5: Effect of Poisson integration. Given a single keyframe as input (a), the shortest-path algorithm yields an initial trajectory that drifts over the bicycle as its gets occluded by the foreground plant (b, red). Since the entire bicycle undergoes the same translation, integrating the scene flow vectors sampled along the initial trajectory removes the drift, resulting in a stable trajectory that runs behind the occluder (b, light blue).

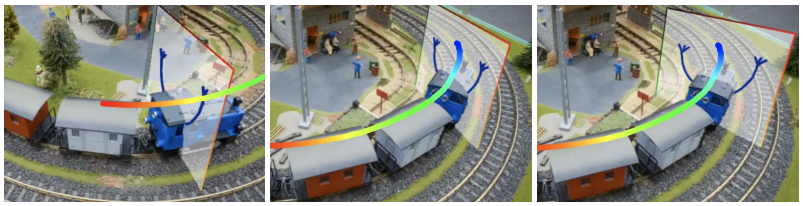
Following this intuition, we sample the scene flow along the initial trajectory to get an estimate of the trajectory's *derivatives* $\{V^1, \ldots, V^T\}$. We then reconstruct a stable, high resolution trajectory by solving for the *continuous* 3D positions $\mathrm{P} = \{P^1, \ldots, P^T\} \in \mathbb{R}^{3\times T}$ of the canvas that satisfy these derivatives, which corresponds to a Poisson problem where the user-provided keyframed pixels $\{\tilde{p}^k\}$ act as boundary constraints (Fig. VI.4b,c). We formulate these constraints via the camera projection operator Π^k, such that the trajectory passes through 3D points P^k that reproject exactly on keyframed pixels. We further regularize the problem by encouraging the trajectory to run close to the unprojected keyframed pixels $\{\tilde{P}^k\}$, yielding:

$$\min_{\mathrm{P}} \sum_t \left\| \left(P^{t+1} - P^t\right) - V^t \right\|^2 + \lambda_{\text{depth}} \sum_k \left\| \tilde{P}^k - P^k \right\|^2, \quad \text{such that } \tilde{p}^k = \Pi^k(P^k). \tag{VI.2}$$

In cases where the user also specifies the depth of a keyframe via the depth slider, we directly enforce that the trajectory passes through the resulting 3D point by setting the constraint to $\tilde{P}^k = P^k$. Such constraints are particularly useful in scenarios where the user wants to anchor a canvas to the hidden side of an object, yet wants the canvas to follow the overall motion of that object (e.g. the left arm of the flamingo in Fig. VI.12). Our Poisson formulation reconciles the initial trajectory, which by definition of the graph nodes only passes through visible points of the object, with the user-specified depth values.

Tracking 3D orientations. Users can also control the orientation of a dynamic canvas along its trajectory by specifying a rotation matrix in one or more keyframes $\{\tilde{R}^k\} \in SO(3)^K$, from which our system deduces a sequence of rotation matrices $\{R^1, \ldots, R^T\} \in SO(3)^T$, one for each frame of the trajectory.

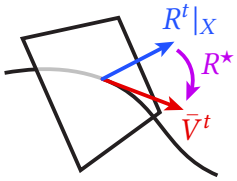
Following our second design goal (**G2**), we want the orientation of the canvas to follow the orientation of the moving object it is tracking, as illustrated in Fig. VI.6. Let us first



(a) 1 keyframe (b) Constant (c) Optimized

Fig. VI.6: In this example, the user orients the canvas to be perpendicular to the motion trajectory in one keyframe (a). Keeping this orientation constant in scene space produces an implausible result as the trajectory turns to follow the tracks (b). Our optimization rotates the canvas to preserve its orientation *relative* to the trajectory (c).

consider the restricted case where the user orients the canvas perpendicularly to the trajectory of the object. Let us further assume that the canvas normal is given by the first axis of the orthogonal frame encoded by the canvas rotation matrix, denoted as $R^t|_X$. Given the normalized scene flow vector $\bar{V}^t$ at each frame t, we can encourage the canvas' normal to align with the trajectory by minimizing $||R^t|_X - \bar{V}^t||^2$.



However, we also want to give users the freedom to choose the *relative* orientation of the canvas with respect to the moving object, for instance to make the doodle parallel rather than perpendicular to the trajectory. We achieve this behavior by introducing an (unknown) rotation matrix $R^\star$ that transforms the canvas relatively to the motion trajectory, yielding:

$$\min_{\{R^t, R^\star\}} \sum_t ||(R^t R^\star)|_X - \bar{V}^t||^2 + \lambda_{\text{smooth}} \sum_t ||R^{t+1} - R^t||^2 , \quad \text{such that } R^k = \tilde{R}^k. \tag{VI.3}$$

The first term encourages the canvas to align with the tangent of the trajectory, up to the relative orientation $R^\star$. The second term prevents the canvas to twist unnecessarily as it travels along the trajectory [294]. Finally, the constraint ensures that the canvas perfectly respects the keyframed orientations.

While the smoothness term brings robustness to noise in the scene flow, it tends to damp the rotation of the canvas around sharp turns of the trajectory. We address this issue by splitting the trajectory in the presence of abrupt turns (which we detect as local minima in velocity, i.e., when $|V^t| < 0.2 \times \max(|V^t|)$) and by assigning a different matrix $R^\star$ to each segment. A sudden change of direction in the trajectory is then captured by an instantaneous change of $R^\star$ rather than by progressive changes of R^t.

Our formulation bears resemblance with algorithms for interpolating orthogonal frames along 3D curves [45, 294], although such methods do not include the additional rotation matrix $R^\star$, which is key to offer users control on canvas orientation in our context.

Similarly to these prior methods, we solve our optimization over the manifold $SO(3)^T$ of rotation matrices using the Riemannian Trust Region algorithm implemented in the Pymanopt library [311], with parameters described by Boumal [45]. Following their recommendations, we initialize orientations with a spherical linear interpolation of the keyframe quaternions. While solving the small linear system in Equation VI.2 is very fast, finding a local minimum of Equation VI.3 can take up to a dozen of seconds, depending on the number of frames and segments (e.g., 1.5" for 1 segment of 80 frames in Fig. VI.6, vs. 9" for 3 segments and 143 frames in Fig. VI.3).

Parameter setting. We used a fixed set of parameters for all our results. A weak regularization on canvas depth $\lambda_{\text{depth}} = 0.01$ mainly serves when the user does not provide depth keyframes. In contrast, a high regularization on orientation smoothness $\lambda_{\text{smooth}} = 10$ prevents the canvas to align with every little turn in the trajectory.

VI.6 Results and evaluation

Fig. VI.12 showcases a variety of video doodles we created with our system based on clips from the *DAVIS* datasets [243, 247], or captured by us. These results cover various application scenarios, ranging from humorous augmentations of casual videos (Flamingo cocktail party, BMX), to informative or instructive (Climbing, Tennis), and other forms of annotations and highlights (Travel vlog, Swinging). Importantly, these results exhibit numerous occlusions between real and drawn content (text in the Travel vlog, legs of the flamingo, frame of the swing), as well as complex trajectories, both in terms of position (Climbing, Comics parkour) and orientation (Swinging). We strongly encourage readers to look at the corresponding videos in supplemental materials.

We first evaluate our tracking algorithm quantitatively on a recent benchmark. We then evaluate our interface qualitatively by asking novice and professional users to create video doodles.

VI.6.1 Tracking accuracy

In the absence of a benchmark for 3D point tracking, we evaluate the accuracy of our algorithm on the recent *TAP-Vid* benchmark [83], which provides a set of ground truth 2D trajectories (called *tracks*) and occlusion flags for the task of tracking arbitrary points in videos. Since our primary contribution is the design of an interactive system for video doodling, the goal of this evaluation is not to claim improvement over fundamental tracking algorithms, but rather to demonstrate that the tailored algorithm we designed to support our user interface is on par with recent algorithms developed for a similar – albeit not identical – task.

Our approach requires a 3D reconstruction, but current algorithms for camera calibration from a single monocular video can fail when there is not enough camera motion, or

Fig. VI.7: Visual comparison of our tracking trajectories against *TAP-Net* [83]. For visualization purposes, we compensate for camera motion by rendering 3D trajectories projected onto the first frame of the video clip. Since the ground truth and *TAP-Net* trajectories are provided as 2D image-space points, we unproject them to 3D using the same cameras and depth maps as used by our method. Our method produces 3D trajectories that are more precise (higher Average Jaccard, AJ) and smoother than *TAP-Net* (top row). While our method can lose track of a point due to occlusion (bottom left), adding a keyframe often suffices to resolve such issues (inset). Both our method and *TAP-Net* struggle to recover subtle composed motions, such as the rotation of the car's wheel (bottom, right).

too many dynamic objects. We therefore restrict our evaluation to a subset of the *TAP-Vid-DAVIS* dataset consisting only of those videos whose 3D reconstruction succeeded, yielding 24 (out of 30) annotated videos for a total of 575 tracks of 62 frames on average.

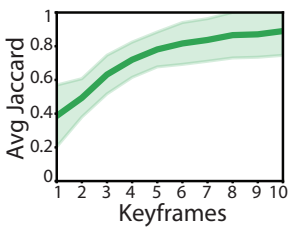
Single keyframe First, we evaluate the performance of our algorithm in the case where only one keyframe is specified. Following the *strided* setting of *TAP-Vid*, we sampled each track every 5 frames to obtain a set of query keyframes, then ran our algorithm for each of these queries to obtain multiple predictions. The final metric is computed as an average over all queries for all tracks and all videos. Since our algorithm outputs a 3D trajectory, we project each point to 2D using the cameras and test for occlusion by checking if the point lies outside the image frame, or if the 3D point lies behind the depth map. Table VI.1 summarizes the outcome of this evaluation using the metrics recommended by Doersch et al. [83] – average Jaccard, average position accuracy of visible points, and binary occlusion accuracy (higher is better). Our method outperforms the state-of-the-art *TAP-Net* method proposed by Doersch et al. [83] on the two first metrics, and it achieves comparable accuracy on occlusion. We also include the scores of the initial trajectory produced by the shortest-path algorithm (Ours w/o Poisson), highlighting the benefit of including the Poisson integration step in our method. Finally, we provide the scores obtained by our method when fixing the keyframe to be the first

Table VI.1: Evaluation of our tracking algorithm on 24 videos of the *TAP-Vid DAVIS* benchmark with a single keyframe (1kf). Adding an extra keyframe (2kf) yields a significant improvement in all metrics (higher is better). We report the results of *TAP-Net* [83] computed on the 24-videos subset for which we obtained a successful 3D reconstruction.

Method	Avg Jaccard (↑)	$< \delta^x_{avg}$ (↑)	Occlusion (↑)
TAP-Net (1kf, strided)	37.2%	52.7%	80.2%
Ours (1kf, strided)	40.8%	59.2%	80.6%
Ours (1kf, strided, w/o Poisson)	18.3%	31.6%	75.1%
Ours (1kf, first)	31.6%	50.9%	75.2%
Ours (2kf)	45.5%	67.3%	78.3%

visible ground-truth 2D position of each track (Ours 1kf, first), as this setting is closer to the way users would interact with our system. As noted by Doersch et al. [83], this keyframe selection strategy yields lower performance.

Multiple keyframes Since we designed our algorithm with interactive control in mind (**G3**), we now evaluate how adding keyframes improves the resulting trajectory. The last row of Table VI.1 (Ours 2kf) reveals that positioning one keyframe at the start of the visible trajectory, and another keyframe at the end, suffices to improve the quality of the track significantly. We additionally ran an experiment where we progressively increased the number of keyframes for each of the 25 tracks of one representative video (50 frames total), adding each new keyframe as the midpoint of the two most distant existing keyframes. The inset shows that the tracking accuracy quickly increases with additional keyframes, and eventually saturates as keyframes get close together.



Qualitative comparison Fig. VI.7 provides a visual comparison between a few of our tracks and the ones predicted by *TAP-Net*, along with the corresponding ground truth. We include additional video comparisons as supplemental materials. Overall, the tracks produced by our method exhibit less jitter than the ones predicted by *TAP-Net* – a property that is essential to achieve convincing video doodles. We also stress that *TAP-Net* only supports a single query point per track, while our method allows users to correct tracking failures due to occlusion or drift by adding keyframes. Moreover, while *TAP-Net* predicts occlusion at a single 2D pixel, our 3D trajectories enable depth testing against a predicted dense depth map to render occlusions over the entire planar canvas.

VI.6.2 User study

We conducted a study with 7 participants, 5 being novices while the other 2 being the professional motion graphics designers we interviewed during our formative study (Section VI.3). The five novices participated in-person by drawing on a Wacom Cintiq 16 tablet, whereas the two professionals participated remotely using only a mouse or trackpad. Participants were first introduced to our system via a short video tutorial, and were then given two tasks:

Task 1 - Goal-directed (10 minutes). Participants are shown an example video doodle that they have to reproduce, shown in inset. This task was designed so that users could experience all the main features of our system, and was inspired by the goals of online tutorials (see T3 in supplemental materials).

Task 2 - Open-ended (30 minutes). Participants are asked to create two novel video doodles by choosing among 15 short video clips (3" duration on average).

Finally, participants answered a questionnaire about the different features of our system. While we conducted this study with a small group of participants, the collected material represents 4.5 hours of interaction with our system, corresponding to a total of 49 canvases. In the following section, we detail important insights we gained from this data, including typical usage of the system's features, unexpected user behavior, and suggestions for improvement. We provide all 21 video doodles as supplemental materials to illustrate the type of effects that users of our system could create after only a few minutes of practice.

Everyone can create video doodles with our system – fast. Fig. VI.11 provides a gallery of results created by the participants during the open-ended task, along with their completion time. Creating the moderately complex video doodle of Task 1 only took 10'30" on average (sd = 03'07"). Creating a video doodle from scratch – including time to brainstorm and experiment ideas – took an average of 14'00" (sd = 07'05'). Both of the professional participants emphasized the speed with which they could create video doodles with our system: *"I feel like we just raced through all those different things because it was so easy and quick to use. If I'm working in After Effects, I have to spend a lot more time worrying about the tracking, or about whether the canvas is going behind this thing or this other thing. Here it just happened automatically and it worked how I thought it should." (P1)*

Furthermore, all participants were highly satisfied with the video doodles they created (score of 4.7 on a 5-point Likert scale). Many participants were enthusiastic at the idea of using our prototype with videos that they would capture themselves.

Automating tracking and 3D rendering frees up time for doodling. By analyzing usage logs, we find that participants spend around half of their time doodling and creating

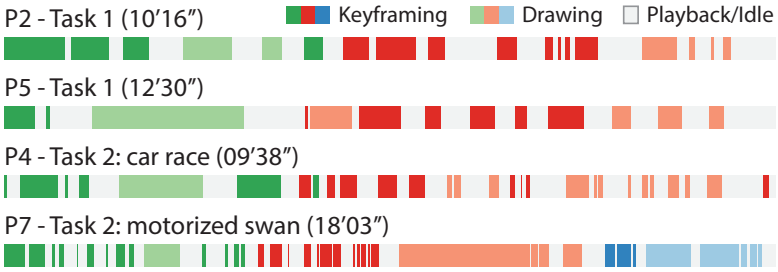


Fig. VI.8: Timelines of performed operations. In this visualization, different hues (green, red, blue) depict different canvases, and brightness differentiates keyframing a canvas (dark colors) from drawing on a canvas (light colors). Participants roughly spent as much time keyframing as drawing, and sometimes alternate between the two operations (P4).

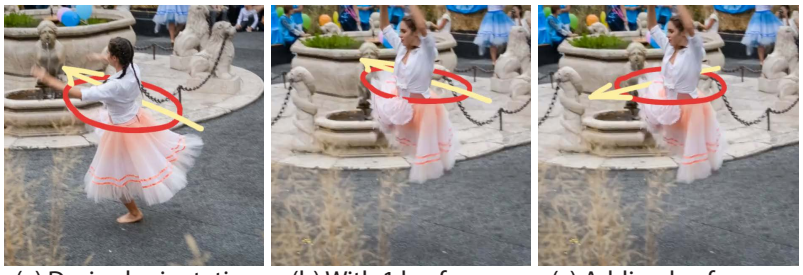
compelling frame-by-frame animations (see Fig. VI.8 for a temporal breakdown of typical sessions). On average, 49% of a session was dedicated to drawing (sd = 17%, min/max = 24/89%, measured as the time spent with the drawing panel open). Even novices picked up quickly the concept of frame-by-frame animation, and all succeeded in creating such animation clips. On average, participants drew 3.3 (sd = 1.8) frames per canvas, with 6.0 (sd = 9.2) strokes per frame. We also observe that participants sometimes alternate phases of drawing with keyframing to adjust the trajectory of a canvas after drawing on it, which happened in 19 out of the 49 canvases created over all video doodles.

These observations suggest that we achieved our design goals **G1** and **G2** – by decoupling 2D drawing from 3D in-scene embedding and tracking, users are able to focus on the creative task of doodling: *" What I found myself spending more time on was redoing the drawings to make them laser or like add more stuff. Since the "hard part" – the tracking – is being taken care of, I could focus on drawing the shapes that I want. " (P2)*

Nevertheless, some participants would have liked to be able to draw directly over the video in complement to the rectified drawing panel, or have the option to vary the brush shape and texture, two features that we did not implement in our prototype.

Combining tracking with keyframing keeps users in control All participants quickly grasped the concept of keyframing to control the position and orientation of the canvases. They used an average of 3.3 (sd = 3.0) position keyframes and 2.5 (sd = 1.8) orientation keyframes per dynamic canvas. Making the canvas track a particular object required 1 or 2 position keyframes in the majority of cases. Out of the 26 dynamic canvases, only 6 canvases required 5 keyframes or more. This extra work was needed when the tracked object gets occluded along its trajectory (P1 - swing, P2 - dancer's hand), or when the desired trajectory does not strictly follows a single point of a moving object (e.g., P5 - lama where the backpack first follows the lama's flank, then its rear).

The effort that participants put into perfecting their results with more keyframes shows



(a) Desired orientation (b) With 1 keyframe (c) Adding keyframes

Fig. VI.9: Limitation. The dancer rotates on herself as she jumps. Given a single keyframe, our tracking algorithm follows the upward motion of the body, but not its rotation (a-b). The user can achieve the desired effect by keyframing the canvas rotation (c).

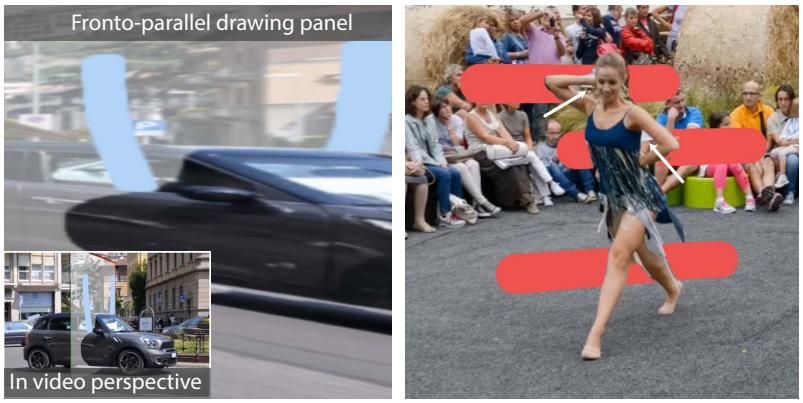
that the interface gave them a sense of agency in the tracking task, in accordance with our design goal **G3**: *" Overall I felt like I was in control, and that it was making smart automatic decisions for me. " (P1) " Adjusting the plane with the keyframes was super simple, and it's great that you can adjust things to make them perfect. You get the [automatic] trajectory and that does most of the work for you, but then you can tweak it. " (P2)*

Still, expert users value the more advanced features available in professional software: *"With After Effects tracking tool, there are so many more options that you have, so I feel like I have more control over the tracking at a pixel level, since I can zoom in and make it perfect." (P1)* One of the experts also reported having difficulty positioning the canvas in depth at first, which suggests possible improvement in visualizing or controlling depth.

Working around limitations. Our choice of embedding strokes in planar canvases prevents the creation of non-planar doodles. Several participants found creative solutions to work around this limitation, such as placing two orthogonal canvases to depict sections of the cloud of steam (Fig. VI.11, P6 - train), or drawing different doodles to depict the side and back views of a backpack on the lama, taking care of switching between these doodles to match the viewpoint in the video (Fig. VI.11, P5 - lama).

VI.6.3 Limitations.

Our tracking algorithm assumes that the object to track is oriented towards its dominant motion trajectory. Fig VI.9(a,b) illustrates a case where this assumption does not hold, as the dancer rotates on herself as she moves across the scene. While our algorithm does not capture this in-plane rotation, the user can reproduce it by adding keyframes (Fig VI.9c). Orientation tracking might be improved by extending our user-guided point tracking approach to track multiple nearby points simultaneously. Our algorithm also tends to lose track of objects when they are too thin, or when they become too occluded. Prior methods proposed to use *skip edges* to deal with occlusions as part of the shortest path optimization [295], but this mechanism increases the complexity of the graph significantly. In our



(a) Drawing context foreshortening (b) Erroneous occlusions

Fig. VI.10: Limitations. When the canvas is highly foreshortened, the video context appears with strong distortions in the fronto-parallel drawing panel (a). We rely on existing video depth estimation methods to handle occlusions. While most of the silhouette of this dancer is well captured, erroneous depth yield wrong occlusions in small regions around her arms (b).

experience, most issues caused by occlusion can be resolved by adding a few keyframes. Better tracking of thin objects could be achieved by increasing the resolution of the graph, potentially relying on a multi-scale strategy to keep the problem tractable, as has been suggested by Bian et al. [35]. While we focused on general point tracking for maximum flexibility, integrating domain-specific algorithms (e.g. body pose tracking [123], or hand trajectory prediction [203]) within our keyframe-based interface could improve robustness for specific use cases.

While we provide context in the drawing panel by rectifying the underlying video with a homography, strong distortion appears when the canvas is too foreshortened (Fig. VI.10a). Future work could explore the use of more advanced novel-view-synthesis techniques, possibly by leveraging the depth maps and multiple views of the scene we have access to.

Finally, our system relies on the depth map provided by *Robust Consistent Video Depth Estimation* to render occlusions automatically. While this solution often suffices for doodles made of sparse strokes, errors around the silhouette of the occluder can appear on densely painted doodles (Fig. VI.10b). Users can fall-back to existing masking tools to handle such cases, although a more integrated solution would consist in providing sparse user corrections to the depth estimation algorithm to refine its result.

Fig. VI.11: Video doodles created with our system by participants of our user study. Participants created static doodles (text in P1 - hike, finishing line in P7 - motorbike race) as well as dynamic doodles (P3 - angel & devil child, P5 - squirrel friend). Several of these doodles get occluded by real objects (poles in P4 - car race, face on the tree in P5 - squirrel friend), and are synchronized with specific video events (yellow sparkles when P2's dancer touches the ground, water splash when P6's horse ends its jump.)

Fig. VI.12: Additional video doodles created with our system.

VI.7 Conclusion

Recent progress in computer vision and recording devices makes 3D reconstruction readily available from casually-captured videos, a trend that is likely to grow with the democratization of depth sensors and on-camera SLAM systems. In this chapter, we showed how depth and motion information can be leveraged to ease the creation of *video doodles*, a popular media that mixes videos with hand-drawn animations. Depth and estimated cameras allow us to embed hand-drawn doodles in 3D with correct perspective and occlusion effects, while our novel controllable tracking method allows artists to easily control how the doodles move along with objects in the scene. A user study and resulting artifacts demonstrate the effectiveness of our interface and our keyframe-based tracking algorithm to create a wide-range of 3D effects.

VI.8 Future Work

3D doodles We focused on the creation of video doodles composed of planar doodles and as such do not support 3D freeform strokes, for instance to draw a swirling ribbon around an object. Future work could explore the definition of non-planar canvases based on parametric shapes [144], height fields [18], extruded contours [231] or planar canvases with user-defined depth [171]. Another promising idea consists of tackling the problem of "lifting" strokes drawn in the 2D image plane to 3D space, following some priors on how strokes are drawn on and around existing 3D surfaces in the captured scene [179, 196]. Our particular setting offers an additional challenge caused by the partial and noisy nature of the information we have about scene geometry. However, we could leverage the interactive setting to let users give more information by drawing 3D strokes at different frames – corresponding to different views. Another avenue for future work is to explore animation effects inspired by work in 3D animation, such as enriching the simple 3D rigid transformation animations we currently have with generated secondary motion [324, 340].

Synchronization and interaction between real content and doodles In the tutorials we surveyed, we noted that a key challenge was about timing a hand-drawn animation to coincide with temporal events in the captured video. Our current system helps users sketch doodles that are spatially aligned to a single frame but does not explicitly provide support to time the doodle animation with respect to the video. Analyzing the video's visual rhythm [71] or key events [296] can help provide the user more insight about the frequency they need to match in an animation loop, similar to how we saw experienced editors analyze manually a video to note down the duration of a specific motion [T1]. Another possibility is to provide a system to create *procedural animations* of doodles, to enable interactions and synchronization between the doodle animation and the video content to be programmatically defined. This can help support triggering animations following video events [343], or to imbue a doodle with physical properties such as

colliding with 3D scene objects [329].

Content authoring applications and computer vision In this chapter we showcase one example of leveraging computer vision and deep learning techniques as part of a content authoring tool designed to be accessible to novices. We are at a time of fast-paced progress in all areas of computer vision – eg in the year following the completion of this project, many new methods have already been published to tackle the problems of arbitrary point tracking in videos [250, 320], and of high-quality video matting [199]. The progress in quality of results in all computer vision tasks can be harnessed to improve image and video editing applications and in fine enable people to create visual effects that they could not have achieved before without extensive training and time resources. However, we believe that designing applications to make computer vision outputs useful to people in a particular authoring scenario is an interesting challenge in itself – and we hope that this chapter is enough evidence to convince the reader. Computer vision excels at automating tasks and giving answers to well-defined questions – eg, where is that point in every single frame of the video? – but an authoring process is more than giving commands to a black box that executes them. Software for creative applications needs to support humans in exploring ideas, and provide a sense of joy, playfulness and pride in the creation process [283]. Designing such software for other video and motion editing tasks such as multi-viewpoint broadcasting, or sports and scientific visualizations, constitutes a promising area of future research.

Least-effort computer vision Video doodles are an example of an application that can work with relatively low quality 3D reconstruction of 3D scenes – an approximate 2.5D reconstruction of the scene via cameras and depth maps is enough. We rely on the original video frames for rendering the new frames, based on the premise that the person capturing the video has already chosen an appropriate camera path – forgoing the need to reconstruct a 3D scene in which free viewpoint navigation is possible. Even though the depth maps we use are sometimes erroneous, our choice to focus on an application with an appealing non-photorealistic style – colorful doodles – make small errors less noticeable to viewers. Despite the ever-increasing capabilities of computer vision methods, we believe that reflecting on how to purposefully leverage lower quality methods can constitute an interesting avenue for future work. Given the scientific consensus on the urgent need to move away from endless resource consumption, questioning whether one *really needs* the highest quality results or most advanced methods for a particular task is an essential step forward [224, 278]. While we arrive at this observation serendipitously in this research project, we hope that future work can consider "least-effort computer vision" as a first-class design principle. Rethinking applications such as VideoDoodles in a "post-growth" perspective [278] raises interesting technical challenges, such as avoiding reliance on the availability of new hardware – in our case depth sensors – and

designing systems that can run locally on consumer devices, as opposed to requiring enterprise-scale compute clusters [169].

Chapter VII

Conclusion

We started off this thesis by proposing 3D sketching as a way for people to express their goals within a 3D authoring system. We have seen three particular examples of how choosing 3D strokes as a representation can help bridge the gap between a person's goals and a 3D authoring system, for 3D shape, appearance and animation authoring. To conclude, I summarize our contributions and discuss future research perspectives inspired by the successes, struggles and limitations of the research projects presented in this thesis.

VII.1 Contributions

We showed that 3D strokes can be used for 3D surface modeling by proposing an algorithm to obtain a 3D surface from a sparse 3D sketch. We take into account the fact that 3D strokes encode not only a sampling of the depicted surface, but also indicate where sharp features are located on this surface. This enables us to obtain a piecewise-smooth surface that better conveys the shape depicted by a sketch than globally smooth surfaces produced by previous methods. We show that it is possible to process 3D sketches that are messy, imprecise and lack connectivity – properties that often arise in the context of sketching freehand in VR.

VR artists also use 3D strokes to directly depict appearance, in a process similar to traditional oil painting where many colored strokes are arranged to render a picture of a scene. Through interviews with 4 VR artists, we give novel insights into this workflow that combines elements of manual rendering with 3D modeling aspects. We report that artists value using VR painting because it is an approachable 3D authoring tool that feels direct and gives a high level of control over the final result. We also highlight 4 challenges faced by artists in their current workflows. We hope this analysis serves as grounds for future inquiries about this emerging artistic practice.

Building on these findings, we set out to use 3D strokes as a way to author *both* 3D shape and appearance, while allowing to edit one independently from the other. While in current VR painting software, 3D shape and appearance are both encoded on a single stroke, we enable artists to assign different roles to strokes: strokes either act as 3D shapes or act as color modifiers on shape strokes. We achieve this by introducing "3D-Layers", an abstraction inspired by bitmap layers in 2D digital painting. We show that this design

enables us to reproduce compelling hand-painted shading effects. This project is still in progress at the current time, we hope to further test our design in a user study.

Finally, we show how 3D strokes can serve to do hand-drawn animation on top of videos – what we call video doodles. Hand-drawn animation is usually done by drawing 2D strokes, yet for video doodles the animation must appear to be embedded in and synchronized with the content of the dynamic 3D scene captured in video. We show that considering how to animate 3D strokes in a partial reconstruction of the 3D scene helps us simplify animation control. In our system, users can create complex motions for doodles with few 2D keyframing interactions. Our 3D-aware algorithms take care of achieving consistent motions and occlusions, by leveraging state-of-the-art computer vision techniques.

VII.2 Research perspectives

VII.2.1 Designing better tools for artists

In this thesis, we show how to use 3D strokes as a representation that artists interact with to create 3D shape (Chapter III), appearance (Chapter V) and animations (Chapter VI). We have demonstrated in each instance that 3D strokes are a valid choice of representation to achieve specific goals that we posit artists have, such as creating a plausible 3D surface from a 3D sketch, or painting a 3D scene. While we demonstrate that our systems work for these purposes and enable artists to generate appealing results, our evaluation is often content with verifying success of the system for one such specific goal, without considering how our system inevitably frames and constrains the authoring process. This approach to designing and evaluating creativity support tools might limit our ability to propose designs that truly "empower" artists [192], as it does not account for the value of letting artists find and develop idiosyncratic workflows that can support unique and personal outcomes [191].

As computer graphics researchers, our work puts a strong emphasis on *artifacts* – the final results of the authoring process, as their visual quality and appeal constitute one of the pillars of evaluation of our scientific contribution. This focus on the final artifact of creation leads us to design authoring tools that support a "hylomorphic model of creation" [145], in which we consider that artists have a "preconceived form" in mind to be realized, and that the creation process is merely about expressing this intent. With that assumption, we disregard how much the very *process* of creation impacts artistic intent and the final result [214]. For example, automatically creating a 3D surface from a 3D sketch of a car generates appealing visual artifacts. Yet a skilled 3D industrial designer going through the process of modeling each surface patch manually by using the 3D sketch as reference might discover important opportunities for improvement of the shape, and in fine arrive at a wholly different final artifact.

However, computer graphics research can provide the technical solutions necessary to

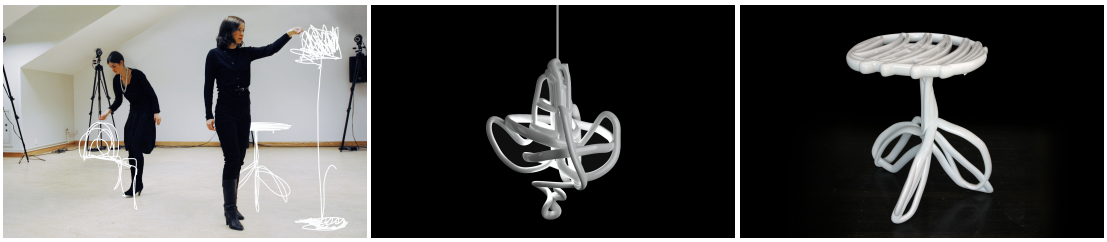


Fig. VII.1: Sketch Furniture Performance Design. Designers of FRONT studio record 3D sketches of furniture drawn mid-air (left), then materialize the sketches as functional furniture (middle and right). ©FRONT [107]

design tools for artists that support the creation process as a whole. Contributions in real-time shape edition and visualization can favor quick exploration of possibilities [218, 239] ; novel digital representations for color can make experimenting with color palettes more playful [285] ; building bridges between programmatic control [152] or large learnt latent spaces [284] and direct manipulation can help artists develop their own unique set of tools. In the future, I am excited to continue my exploration of how to design, develop and evaluate authoring tools that support personal and particular workflows, and that "users appropriate for ends unforeseen" [192]. This might be achieved through close collaboration with artists throughout the research process [191], which is something I hope to pursue in future research.

VII.2.2 3D sketching for computational fabrication

We have shown in this thesis that 3D strokes are a valuable representation of digital 3D shape and appearance because they allow interactions that feel direct and expressive. 3D strokes can help humans manipulate digital material ; can 3D strokes also become part of the interface between humans and fabrication machines?

In 2005, designers of FRONT studio exhibited a performance in which they record 3D sketches of chairs and tables that they draw with a tracked hand-held device mid-air. The resulting strokes were then "materialized" as life-sized functional furniture using additive manufacturing (Fig. VII.1). In the future, I am interested in exploring this question, raised by FRONT almost 20 years ago in their performance: "is it possible to let a first sketch become an object, to design directly onto space?" [107]

Computer controlled tools and machines are often considered as black box systems that take a 3D shape as input and are tasked with producing the shape as closely as possible. However, hiding away the fabrication process prevents artists from engaging with how a machine works and how material interacts with it, which can hinder exploration or development of new workflows and in turn, hamper creation of novel designs and aesthetics [191]. A recent line of work in computational fabrication looks instead at the control of machines at the level of the *toolpath*, for example allowing a 3D printed clay

vase to be defined by the path that the clay extruder takes through 3D space [47]. With programmatic control over how the machine moves through space while depositing or carving material, makers can control how material used for fabrication reacts to the process, for example how gravity and the deposition motion affects clay coils [47], how drawing speed affects ink bleeding in paper [105], or how the deposition orientation of plastic filament affects surface reflectivity [59] or material properties [158].

Previous work shows tremendous potential in building tools that let engineers and craftspeople control machine toolpaths. In the future, I would like to explore how to provide control over machine toolpaths through direct manipulation. Researchers have explored this topic in the past, by integrating machine command capabilities into design software [105], linking live input sources with machine control [325], and by combining manual toolpath control with computational assistance [249, 256, 350]. I believe that here again, 3D sketching could be part of the design for a toolpath control tool that feels direct and expressive.

VII.2.3 Digital content authoring within a material context

Authoring 3D content *in-context* is a topic we have touched on in Chapter VI, where the video footage serves as the context. In this scenario, we consider the *geometric context* in which the animated doodles are added and help people make their animation match this context. The geometric context can be very useful for 3D design, for example to create hand-held products that fit well with particular hand poses [173], to prototype furniture that fits in a given space [196], or to design augmented reality experiences that adapt to the surroundings [188]. Geometric context is also critical to consider in the scenario mentioned above of computational fabrication, as the physical object must fit its context, if we want it to be useful [297].

In future research, I want to explore how *material context* can impact and constrain the digital 3D authoring workflow, when designing for physical fabrication. By material context, I mean to encompass material aspects of fabrication that are specific to the local context in which fabrication happens: what machines are available to a person? what manual skills do they have? what material resources do they have access to? Taking into account material context means considering not only the final artifact and its 3D shape as an objective for design, but also the wider ecosystem in which fabrication takes place, which introduces an array of exciting new challenges. Supporting 3D object design while taking into account material context can mean exploring a design space formed by multiple conflicting objectives other than shape [345] ; enabling and encouraging people to design with material usually considered waste [14, 81, 186, 319] ; or incorporating considerations about resources scarcity as part of the design process [276, 327].

Bibliography

[1] Fatemeh Abbasinejad, Pushkar Joshi, and Nina Amenta, 2011. "Surface patches from unorganized space curves". In *Computer Graphics Forum*, vol. 30. Cited pages 27 and 28.

[2] Rinat Abdrashitov, Alec Jacobson, and Karan Singh, 2019. "A system for efficient 3D printed stop-motion face animation". *ACM Transactions on Graphics (TOG)*, vol. 39, no. 1, pages 1–11. Cited page 21.

[3] Adobe, 1987. "Illustrator". https://www.adobe.com/products/illustrator.html. Cited page 71.

[4] Adobe, 1990. "Photoshop". https://www.adobe.com/products/photoshop.html. Cited pages 71 and 75.

[5] Adobe, 2007. "Substance 3d painter". https://www.adobe.com/products/substance3d-painter.html. Cited page 20.

[6] Adobe, 2022. "After effects". https://www.adobe.com/products/aftereffects.html. Cited pages 94 and 95.

[7] Adobe, 2023. "Medium". https://www.adobe.com/products/medium.html. Cited pages 12, 42, and 53.

[8] Aseem Agarwala, Aaron Hertzmann, David H Salesin, and Steven M Seitz, 2004. "Keyframe-based tracking for rotoscoping and animation". *ACM Transactions on Graphics (Proc. SIGGRAPH)*, vol. 23, no. 3. Cited pages 23 and 97.

[9] Maneesh Agrawala, Andrew C Beers, and Marc Levoy, 1995. "3D painting on scanned surfaces". In *Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 145–ff. Cited page 20.

[10] Sung Joon Ahn, W. Rauh, Hyung Suck Cho, and H.-J. Warnecke, 2002. "Orthogonal distance fitting of implicit curves and surfaces". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5. Cited page 37.

[11] Yağız Aksoy, Tunç Ozan Aydın, Aljoša Smolić, and Marc Pollefeys, 2017. "Unmixing-based soft color segmentation for image manipulation". *ACM Trans. Graph.*, vol. 36, no. 2, pages 19:1–19:19. Cited page 75.

[12] Marylyn Alex, Burkhard C Wünsche, and Danielle Lottridge, 2021. "Virtual reality art-making for stroke rehabilitation: Field study and technology probe". *International Journal of Human-Computer Studies*, vol. 145, page 102,481. Cited page 60.

[13] Brian Amberg and Thomas Vetter, 2011. "GraphTrack: Fast and globally optimal tracking in videos". In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Cited pages 97 and 102.

[14] Felix Amtsberg, Yijiang Huang, DJ Marshall, Kevin Moreno Gata, and Caitlin Mueller, 2020. "Structural up-cycling: matching digital and natural geometry". *Advances in Architectural Geometry*. Cited page 120.

[15] Baptiste Angles, Marco Tarini, Brian Wyvill, Loïc Barthe, and Andrea Tagliasacchi, 2017. "Sketch-based implicit blending". *ACM Transactions on Graphics (TOG)*, vol. 36, no. 6, pages 1–13. Cited page 13.

[16] Apple, 2022. "Arkit". https://developer.apple.com/augmented-reality/arkit/. Cited page 96.

[17] Apple, 2023. "Vision pro". https://www.apple.com/apple-vision-pro/. Cited page 1.

[18] Rahul Arora, Rubaiat Habib Kazi, Tovi Grossman, George Fitzmaurice, and Karan Singh, 2018. "SymbiosisSketch: Combining 2D & 3D sketching for designing detailed 3D objects in situ". In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–15. Cited pages 17, 54, 59, 74, and 113.

[19] Rahul Arora, Rubaiat Habib Kazi, Fraser Anderson, Tovi Grossman, Karan Singh, and George W Fitzmaurice, 2017. "Experimental evaluation of sketching on surfaces in VR". In *CHI*, vol. 17, pages 5643–5654. Cited pages 4, 16, 28, 35, 53, 54, 59, 69, 70, and 73.

[20] Rahul Arora, Rubaiat Habib Kazi, Danny M Kaufman, Wilmot Li, and Karan Singh, 2019. "MagicalHands: Mid-air hand gestures for animating in VR". In *Proceedings of the 32nd annual ACM symposium on user interface software and technology*, pages 463–477. Cited pages 22 and 23.

[21] Rahul Arora, Mayra Donaji Barrera Machuca, Philipp Wacker, Daniel Keefe, and Johann Habakuk Israel, 2023. "Introduction to 3D sketching". In *Interactive Sketch-based Interfaces and Modelling for Design*, pages 151–177. River Publishers. Cited page 16.

[22] Rahul Arora and Karan Singh, 2021. "Mid-air drawing of curves on 3D surfaces in virtual reality". *ACM Transactions on Graphics (TOG)*, vol. 40, no. 3, pages 1–17. Cited pages 20, 21, 73, and 81.

[23] Seok-Hyung Bae, Ravin Balakrishnan, and Karan Singh, 2008. "ILoveSketch: as-natural-as-possible sketching system for creating 3D curve models". In *Proceedings of the 21st annual ACM symposium on User interface software and technology*, pages 151–160. Cited pages 14, 27, 28, 46, and 99.

[24] Seok-Hyung Bae, Ravin Balakrishnan, and Karan Singh, 2009. "EverybodyLovesSketch: 3D sketching for a broader audience". In *Proceedings of the 22nd annual ACM symposium on User interface software and technology*, pages 59–68. Cited page 14.

[25] Andreas Bærentzen, Jeppe Revall Frisvad, and Karan Singh, 2019. "Signifier-based immersive and interactive 3D modeling". In *Proceedings of the 25th ACM Symposium on Virtual Reality Software and Technology*, pages 1–5. Cited pages 11 and 12.

[26] Ravin Balakrishnan, George Fitzmaurice, Gordon Kurtenbach, and William Buxton, 1999. "Digital tape drawing". In *Proceedings of the 12th annual ACM symposium on User interface software and technology*, pages 161–169. Cited page 17.

[27] Seungbae Bang and Sung-Hee Lee, 2018. "Spline interface for intuitive skinning weight editing". *ACM Transactions on Graphics (TOG)*, vol. 37, no. 5, pages 1–14. Cited page 21.

[28] Ilya Baran, Johannes Schmid, Thomas Siegrist, Markus Gross, and Robert W Sumner, 2011. "Mixed-order compositing for 3D paintings". In *Proceedings of the 2011 SIGGRAPH Asia Conference*, pages 1–6. Cited pages 21 and 75.

[29] Connelly Barnes, David E Jacobs, Jason Sanders, Dan B Goldman, Szymon Rusinkiewicz, Adam Finkelstein, and Maneesh Agrawala, 2008. "Video puppetry: a performative interface for cutout animation". In *ACM SIGGRAPH Asia 2008 papers*, pages 1–9. Cited page 23.

[30] Erhan Batuhan Arisoy, Gunay Orbay, and Levent Burak Kara, 2012. "Free form surface skinning of 3d curve clouds for conceptual shape design". *Journal of computing and information science in engineering*, vol. 12, no. 3. Cited page 30.

[31] Jean-Philippe Bauchet and Florent Lafarge, 2020. "Kinetic shape reconstruction". *ACM Transactions on Graphics*, vol. 39, no. 5. Cited page 39.

[32] Michel Beaudouin-Lafon and Wendy E Mackay, 2000. "Reification, polymorphism and reuse: three principles for designing visual interfaces". In *Proceedings of the working conference on Advanced visual interfaces*, pages 102–109. Cited page 74.

[33] Matthew Berger, Andrea Tagliasacchi, Lee M Seversky, Pierre Alliez, Gael Guennebaud, Joshua A Levine, Andrei Sharf, and Claudio T Silva, 2017. "A survey of surface reconstruction from point clouds". In *Computer Graphics Forum*, vol. 36. Cited pages 29 and 53.

[34] Mikhail Bessmeltsev, Caoyu Wang, Alla Sheffer, and Karan Singh, 2012. "Design-driven quadrangulation of closed 3d curves". *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*, vol. 31, no. 6. Cited pages 27 and 28.

[35] Zhangxing Bian, Allan Jabri, Alexei A. Efros, and Andrew Owens, June 2022. "Learning pixel trajectories with multiscale contrastive random walks". In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Cited page 111.

[36] Blender, 2022. "Blender grease pencil". https://www.blender.org/features/story-artist/. Cited pages 21 and 99.

[37] Blender, 2022. "Blender motion tracking". https://docs.blender.org/manual/en/latest/movie_clip/tracking/index.html. Cited pages 94 and 95.

[38] Blender, 2023. "Blender modeling". https://www.blender.org/features/modeling/#tools. Cited pages 11, 12, and 42.

[39] Blender, 2023. "Blender sculpting". https://www.blender.org/features/sculpting/. Cited pages 12 and 53.

[40] Blender, 2023. "Blender uv tools". https://docs.blender.org/manual/en/latest/modeling/meshes/editing/uv.html. Cited pages 19 and 20.

[41] Jules Bloomenthal and Chandrajit Bajaj, 1997. *Introduction to implicit surfaces*. Morgan Kaufmann. Cited page 13.

[42] Alexandra Bonnici, Alican Akman, Gabriel Calleja, Kenneth P Camilleri, Patrick Fehling, Alfredo Ferreira, Florian Hermuth, Johann Habakuk Israel, Tom Landwehr, Juncheng Liu, et al., 2019. "Sketch-based interaction and modeling: where do we stand?" *AI EDAM*, vol. 33, no. 4, pages 370–388. Cited page 14.

[43] BorisFX, 2022. "Mocha pro". https://borisfx.com/products/mocha-pro/. Cited pages 94, 95, and 96.

[44] Mario Botsch and Olga Sorkine, 2007. "On linear variational surface deformation methods". *IEEE transactions on visualization and computer graphics*, vol. 14, no. 1, pages 213–230. Cited page 12.

[45] Nicolas Boumal, 2013. "Interpolation and regression of rotation matrices". In *International Conference on Geometric Science of Information*, pages 345–352. Springer. Cited pages 104 and 105.

[46] Samuelle Bourgault and Jennifer Jacobs, 2023. "Preserving hand-drawn qualities in audio-visual performance through sketch-based interaction". *Journal of Computer Languages*, vol. 74, page 101,186. Cited page 60.

[47] Samuelle Bourgault, Pilar Wiley, Avi Farber, and Jennifer Jacobs, 2023. "CoilCAM: Enabling parametric design for clay 3D printing through an action-oriented toolpath programming system". In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–16. Cited page 120.

[48] Yuri Boykov, Olga Veksler, and Ramin Zabih, 2001. "Fast approximate energy minimization via graph cuts". *IEEE Transactions on pattern analysis and machine intelligence*, vol. 23, no. 11. Cited pages 35, 36, and 43.

[49] Yuri Y. Boykov and Marie-Pierre Jolly, 2001. "Interactive graph cuts for optimal boundary and region segmentation of objects in n-d images". In *IEEE International Conference on Computer Vision*. Cited page 34.

[50] Virginia Braun and Victoria Clarke, 2006. "Using thematic analysis in psychology". *Qualitative research in psychology*, vol. 3, no. 2, pages 77–101. Cited page 61.

[51] Aeron Buchanan and Andrew Fitzgibbon, 2006. "Interactive feature tracking using kd trees and dynamic programming". In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Cited pages 97 and 102.

[52] Bill Buxton, 2010. *Sketching user experiences: getting the design right and the right design*. Morgan kaufmann. Cited pages 11 and 54.

[53] W Buxton, 1995. "Touch, gesture & marking. chapter 7 in rm baecker, j. grudin, w. buxton and s. greenberg, s.(eds.). readings in human computer interaction: Toward the year 2000". Cited page 3.

[54] Mental Canvas, 2022. "Mental canvas application". https://mentalcanvas.com/. Cited page 99.

[55] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh, 2019. "OpenPose: Realtime multi-person 2d pose estimation using part affinity fields". *IEEE Transactions on Pattern Analysis*

and Machine Intelligence. Cited page 97.

[56] Yifei Cheng, Yukang Yan, Xin Yi, Yuanchun Shi, and David Lindlbauer, 2021. "Semanticadapt: Optimization-based adaptation of mixed reality layouts leveraging virtual-physical semantic connections". In *The 34th Annual ACM Symposium on User Interface Software and Technology*, pages 282–297. Cited page 90.

[57] Gianmarco Cherchi, Marco Livesu, Riccardo Scateni, and Marco Attene, 2020. "Fast and robust mesh arrangements using floating-point arithmetic". *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*, vol. 39, no. 6. Cited page 39.

[58] Gianmarco Cherchi, Fabio Pellacini, Marco Attene, and Marco Livesu, 2022. "Interactive and robust mesh booleans". *ACM Transactions on Graphics (TOG)*, vol. 41, no. 6, pages 1–14. Cited page 81.

[59] Xavier Chermain, Cédric Zanni, Jonàs Martínez, Pierre-Alexandre Hugron, and Sylvain Lefebvre, Jul. 2023. "Orientable Dense Cyclic Infill for Anisotropic Appearance Fabrication". *ACM Transactions on Graphics*, vol. 42, no. 4, page 13. URL: https://hal.science/hal-04129173, doi:10.1145/3592412. Cited page 120.

[60] Tara Chittenden, 2018. "Tilt brush painting: Chronotopic adventures in a physical-virtual threshold". *Journal of contemporary painting*, vol. 4, no. 2, pages 381–403. Cited pages 59 and 60.

[61] Byungkuk Choi, Haekwang Eom, Benjamin Mouscadet, Stephen Cullingford, Kurt Ma, Stefanie Gassel, Suzi Kim, Andrew Moffat, Millicent Maier, Marco Revelant, et al., 2022. "Animatomy: An animator-centric, anatomically inspired system for 3D facial modeling, animation and transfer". In *SIGGRAPH Asia 2022 Conference Papers*, pages 1–9. Cited page 21.

[62] Marianela Ciolfi Felice, Sarah Fdili Alaoui, and Wendy E Mackay, 2018. "Knotation: exploring and documenting choreographic processes". In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–12. Cited page 60.

[63] David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun, 2004. "Variational shape approximation". In *ACM Transactions on Graphics (Proc. SIGGRAPH)*, pages 905–914. Cited page 30.

[64] Adèle Colas, Wouter van Toll, Katja Zibrek, Ludovic Hoyet, A-H Olivier, and Julien Pettré, 2022. "Interaction fields: Intuitive sketch-based steering behaviors for crowd simulation". In *Computer Graphics Forum*, vol. 41, pages 521–534. Wiley Online Library. Cited page 21.

[65] Forrester Cole, Aleksey Golovinskiy, Alex Limpaecher, Heather Stoddart Barros, Adam Finkelstein, Thomas Funkhouser, and Szymon Rusinkiewicz, aug 2008. "Where do people draw lines?" *ACM Trans. Graph.*, vol. 27, no. 3. doi:10.1145/1360612.1360687. Cited pages 14 and 26.

[66] Jacopo Colò, 2021. "Penzil". https://www.penzil.app/. Cited page 46.

[67] Sabine Coquillart, 1990. "Extended free-form deformation: A sculpturing tool for 3D geometric modeling". In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 187–196. Cited page 12.

[68] Franklin C Crow, 1977. "Shadow algorithms for computer graphics". *ACM Siggraph*, vol. 11, no. 2, pages 242–248. Cited page 82.

[69] Carolina Cruz-Neira, Daniel J Sandin, Thomas A DeFanti, Robert V Kenyon, and John C Hart, 1992. "The CAVE: audio visual experience automatic virtual environment". *Communications of the ACM*, vol. 35, no. 6, pages 64–73. Cited page 16.

[70] Eric Daniels, 1999. "Deep canvas in Disney's Tarzan". In *ACM SIGGRAPH 99 Conference Abstracts and Applications*, SIGGRAPH '99, page 200. Association for Computing Machinery, New York, NY, USA. doi:10.1145/311625.312010. Cited page 20.

[71] Abe Davis and Maneesh Agrawala, 2018. "Visual rhythm and beat". *ACM Transactions on Graphics (TOG)*, vol. 37, no. 4, pages 1–11. Cited page 113.

[72] Richard C Davis, Brien Colwell, and James A Landay, 2008. "K-sketch: a'kinetic'sketch pad for novice animators". In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 413–422. Cited page 22.

[73] Bruno De Araujo and Joaquim Jorge, 2003. "Blobmaker: Free-form modelling with variational implicit surfaces". In *Proceedings of*, vol. 12, pages 17–26. Cited page 15.

[74] Bruno R De Araújo, Géry Casiez, Joaquim A Jorge, and Martin Hachet, 2013. "Mockup builder: 3d modeling on and above the surface". *Computers & Graphics*, vol. 37, no. 3, pages 165–178. Cited page 17.

[75] Fernando De Goes and Doug L James, 2017. "Regularized kelvinlets: sculpting brushes based on fundamental solutions of elasticity". *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, pages 1–11. Cited pages 12 and 13.

[76] Fernando De Goes, William Sheffler, and Kurt Fleischer, 2022. "Character articulation through profile curves". *ACM Transactions on Graphics (TOG)*, vol. 41, no. 4, pages 1–14. Cited pages 13, 21, and 54.

[77] Chris De Paoli and Karan Singh, 2015. "SecondSkin: sketch-based construction of layered 3D models". *ACM Transactions on Graphics (TOG)*, vol. 34, no. 4, pages 1–10. Cited page 14.

[78] David DeBry, Jonathan Gibbs, Devorah DeLeon Petty, and Nate Robins, 2002. "Painting and rendering textures on unparameterized models". In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 763–768. Cited page 19.

[79] Michael F Deering, 1995. "HoloSketch: a virtual reality sketching/animation tool". *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 2, no. 3, pages 220–238. Cited page 16.

[80] Andrew Delong, Anton Osokin, Hossam N Isack, and Yuri Boykov, 2012. "Fast approximate energy minimization with label costs". *International journal of computer vision*, vol. 96, no. 1. Cited page 35.

[81] Kristin N Dew, Samantha Shorey, and Daniela Rosner, 2018. "Making within limits: Towards salvage fabrication". In *Proceedings of the 2018 Workshop on Computing within Limits*, pages 1–11. Cited page 120.

[82] Manfredo P Do Carmo, 2016. *Differential geometry of curves and surfaces: revised and*

updated second edition. Courier Dover Publications. Cited page 18.

[83] Carl Doersch, Ankush Gupta, Larisa Markeeva, Adria Recasens Continente, Kucas Smaira, Yusuf Aytar, Joao Carreira, Andrew Zisserman, and Yi Yang, 2022. "TAP-Vid: A benchmark for tracking any point in a video". In *NeurIPS Datasets Track.* Cited pages 97, 102, 105, 106, and 107.

[84] Julie Dorsey, Holly Rushmeier, and François Sillion, 2010. *Digital modeling of material appearance.* Elsevier. Cited pages 9 and 18.

[85] Julie Dorsey, Songhua Xu, Gabe Smedresman, Holly Rushmeier, and Leonard McMillan, 2007. "The mental canvas: A tool for conceptual architectural design and analysis". In *15th Pacific Conference on Computer Graphics and Applications (PG'07)*, pages 201–210. IEEE. Cited pages 14, 21, and 99.

[86] Pierre Dragicevic, Gonzalo Ramos, Jacobo Bibliowitcz, Derek Nowrouzezahrai, Ravin Balakrishnan, and Karan Singh, 2008. "Video browsing by direct manipulation". In *Proc. ACM SIGCHI Conference on Human Factors in Computing Systems.* Cited pages 22 and 96.

[87] Tobias Drey, Jan Gugenheimer, Julian Karlbauer, Maximilian Milo, and Enrico Rukzio, 2020. "VRSketchIn: Exploring the design space of pen and tablet interaction for 3D sketching in virtual reality". In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–14. ACM, New York, NY, USA. Cited pages 17, 59, and 74.

[88] Ruofei Du, Eric Turner, Maksym Dzitsiuk, Luca Prasso, Ivo Duarte, Jason Dourgarian, Joao Afonso, Jose Pascoal, Josh Gladstone, Nuno Cruces, et al., 2020. "DepthLab: Real-time 3D interaction with depth maps for mobile augmented reality". In *Proc. ACM Symposium on User Interface Software and Technology (UIST).* Cited page 96.

[89] Tao Du, Jeevana Priya Inala, Yewen Pu, Andrew Spielberg, Adriana Schulz, Daniela Rus, Armando Solar-Lezama, and Wojciech Matusik, 2018. "Inversecsg: Automatic conversion of 3d models to csg trees". *ACM Transactions on Graphics (TOG)*, vol. 37, no. 6, pages 1–16. Cited page 13.

[90] Xingyi Du, Qingnan Zhou, Nathan Carr, and Tao Ju, 2021. "Boundary-sampled halfspaces: a new representation for constructive solid modeling". *ACM Transactions on Graphics (TOG)*, vol. 40, no. 4, pages 1–15. Cited page 13.

[91] Xingyi Du, Qingnan Zhou, Nathan Carr, and Tao Ju, 2021. "Boundary-sampled halfspaces: A new representation for constructive solid modeling". *ACM Transactions on Graphics (Proc. SIGGRAPH)*, vol. 40, no. 4. Cited page 39.

[92] Zheng-Jun Du, Liang-Fu Kang, Jianchao Tan, Yotam Gingold, and Kun Xu, 2023. "Image vectorization and editing via linear gradient layer decomposition". *ACM Transactions on Graphics (TOG)*, vol. 42, no. 4, pages 1–13. Cited page 75.

[93] Marek Dvorožňák, Saman Sepehri Nejad, Ondřej Jamriška, Alec Jacobson, Ladislav Kavan, and Daniel Sýkora, 2018. "Seamless reconstruction of part-based high-relief models from hand-drawn images". In *Proceedings of International Symposium on Sketch-Based Interfaces and Modeling.* Cited page 29.

[94] Marek Dvorožňák, Daniel Sýkora, Cassidy Curtis, Brian Curless, Olga Sorkine-Hornung,

and David Salesin, 2020. “Monster Mash: A single-view approach to casual 3D modeling and animation”. *ACM Transactions on Graphics*, vol. 39, no. 6. Cited pages 15 and 29.

[95] Hesham Elsayed, Mayra Donaji Barrera Machuca, Christian Schaarschmidt, Karola Marky, Florian Müller, Jan Riemann, Andrii Matviienko, Martin Schmitz, Martin Weigel, and Max Mühlhäuser, 2020. “VRsketchpen: unconstrained haptic assistance for sketching in virtual 3d environments”. In *Proceedings of the 26th ACM Symposium on Virtual Reality Software and Technology*, pages 1–11. Cited page 17.

[96] ephtracy, 2021. “Magicacsg”. https://ephtracy.github.io/index.html?page=magicacsg. Cited page 13.

[97] João Marcelo Evangelista Belo, Mathias N. Lystbæk, Anna Maria Feit, Ken Pfeuffer, Peter Kán, Antti Oulasvirta, and Kaj Grønbæk, 2022. “AUIT – the adaptive user interfaces toolkit for designing xr applications”. UIST ’22. Association for Computing Machinery, New York, NY, USA. doi:10.1145/3526113.3545651. Cited page 90.

[98] Alex Evans, 2015. “Learning from failure: a survey of promising, unconventional and mostly abandoned renderers for ‘dreams ps4’, a geometrically dense, painterly ugc game”. *Advances in Real-Time Rendering in Games. MediaMolecule, SIGGRAPH*, vol. 2. Cited page 91.

[99] Melvin Even, Pierre Bénard, and Pascal Barla, 2023. “Non-linear rough 2D animation using transient embeddings”. In *Computer Graphics Forum*, vol. 42, pages 411–425. Wiley Online Library. Cited page 23.

[100] Ricardo Fabbri and Benjamin B. Kimia, 2010. “3D curve sketch: Flexible curve-based stereo reconstruction and calibration”. In *IEEE Conference on Computer Vision and Pattern Recognition*. Cited page 30.

[101] Andreas Rene Fender, Thomas Roberts, Tiffany Luong, and Christian Holz, 2023. “InfinitePaint: Painting in virtual reality with passive haptics using wet brushes and a physical proxy canvas”. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–13. Cited page 17.

[102] Jakub Fišer, Paul Asente, and Daniel Sỳkora, 2015. “Shipshape: a drawing beautification assistant”. In *Proceedings of the workshop on Sketch-Based Interfaces and Modeling*, pages 49–57. Eurographics Association, Geneve, Switzerland. Cited page 17.

[103] Jakub Fišer, Ondřej Jamriška, Michal Lukáč, Eli Shechtman, Paul Asente, Jingwan Lu, and Daniel Sỳkora, 2016. “Stylit: illumination-guided example-based stylization of 3d renderings”. *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, pages 1–11. Cited page 90.

[104] Shachar Fleishman, Daniel Cohen-Or, and Cláudio T Silva, 2005. “Robust moving least-squares fitting with sharp features”. *ACM Transactions on Graphics*, vol. 24, no. 3. Cited page 29.

[105] Frikk Fossdal, Rogardt Heldal, and Nadya Peek, 2021. “Interactive digital fabrication machine control directly within a CAD environment”. In *Proceedings of the 6th Annual ACM Symposium on Computational Fabrication*, pages 1–15. Cited page 120.

[106] Foundry, 2022. “Nuke”. https://www.foundry.com/products/nuke-family/nuke.

Cited pages 94 and 95.

[107] Front, 2007. "Sketch furniture". http://www.frontdesign.se/sketch-furniture-performance-design-project. Cited pages 4 and 119.

[108] Chi-Wing Fu, Jiazhi Xia, and Ying He, 2010. "Layerpaint: A multi-layer interactive 3D painting interface". In *Proceedings of the sigchi conference on human factors in computing systems*, pages 811–820. Cited pages 20 and 73.

[109] Goro Fujita, 2023. "Beyond the fence". https://quill.art/stories_beyond_the_fence.html. Cited page 59.

[110] Masaki Fujita and Suguru Saito, 2017. "Hand-drawn animation with self-shaped canvas". In *ACM SIGGRAPH 2017 Posters*, pages 1–2. Cited page 23.

[111] Ran Gal, Olga Sorkine, Niloy J Mitra, and Daniel Cohen-Or, 2009. "iWIRES: An analyze-and-edit approach to shape manipulation". In *ACM SIGGRAPH 2009 papers*, pages 1–10. Cited pages 13 and 54.

[112] Tinsley A Galyean and John F Hughes, 1991. "Sculpting: An interactive volumetric modeling technique". *ACM SIGGRAPH Computer Graphics*, vol. 25, no. 4, pages 267–274. Cited page 4.

[113] Michael Garland and Paul S Heckbert, 1997. "Surface simplification using quadric error metrics". In *Annual conference on computer graphics and interactive techniques (SIGGRAPH)*, pages 209–216. Cited page 43.

[114] Yotam Gingold, Takeo Igarashi, and Denis Zorin, 2009. "Structured annotations for 2D-to-3D modeling". *ACM Transactions on Graphics (TOG)*, vol. 28, no. 5, page 148. doi:http://doi.acm.org/10.1145/1618452.1618494. Cited page 15.

[115] Yotam I. Gingold, Philip L. Davidson, Jefferson Y. Han, and Denis Zorin, 2006. "A direct texture placement and editing interface". In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology*, UIST '06, page 23–32. Association for Computing Machinery, New York, NY, USA. doi:10.1145/1166253.1166259. Cited pages 19 and 73.

[116] Dan B Goldman, Brian Curless, David Salesin, and Steven M Seitz, 2006. "Schematic storyboarding for video visualization and editing". *ACM Transactions on Graphics (Proc. SIGGRAPH)*, vol. 25, no. 3. Cited page 96.

[117] Dan B Goldman, Chris Gonterman, Brian Curless, David Salesin, and Steven M Seitz, 2008. "Video object annotation, navigation, and composition". In *Proc. ACM symposium on User Interface Software and Technology (UIST)*, pages 3–12. Cited pages 22 and 96.

[118] Alison Goodyear and Mu Mu, 2019. "Abstract painting practice: Expanding in a virtual world". In *ACM International Conference on Interactive Experiences for TV and Online Video (TVX'19)*. Cited pages 59 and 60.

[119] Giorgio Gori, Alla Sheffer, Nicholas Vining, Enrique Rosales, Nathan Carr, and Tao Ju, 2017. "Flowrep: Descriptive curve networks for free-form design shapes". *ACM Transaction on Graphics (Proc. SIGGRAPH)*, vol. 36, no. 4. Cited page 49.

[120] Yulia Gryaditskaya, Felix Hähnlein, Chenxi Liu, Alla Sheffer, and Adrien Bousseau, nov

2020. “Lifting freehand concept sketches into 3D”. *ACM Trans. Graph.*, vol. 39, no. 6. doi:10.1145/3414685.3417851. Cited pages 14 and 25.

[121] Yulia Gryaditskaya, Mark Sypesteyn, Jan Willem Hoftijzer, Sylvia Pont, Fredo Durand, and Adrien Bousseau, 2019. “Opensketch: A richly-annotated dataset of product design sketches”. *ACM Transactions on Graphics (TOG)*, vol. 38, no. 6, page 232. Cited pages 14 and 26.

[122] Martin Guay, Rémi Ronfard, Michael Gleicher, and Marie-Paule Cani, 2015. “Space-time sketching of character animation”. *ACM Transactions on Graphics (ToG)*, vol. 34, no. 4, pages 1–10. Cited page 22.

[123] Rıza Alp Güler, Natalia Neverova, and Iasonas Kokkinos, 2018. “Densepose: Dense human pose estimation in the wild”. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7297–7306. Cited page 111.

[124] Felix Hähnlein, Yulia Gryaditskaya, Alla Sheffer, and Adrien Bousseau, 2022. “Symmetry-driven 3D reconstruction from concept sketches”. In *ACM SIGGRAPH 2022 Conference Proceedings*, pages 1–8. Cited pages 14 and 15.

[125] Pat Hanrahan and Paul Haeberli, 1990. “Direct WYSIWYG painting and texturing on 3D shapes”. *ACM SIGGRAPH computer graphics*, vol. 24, no. 4, pages 215–223. Cited pages 20, 21, and 73.

[126] Beverly L Harrison, Hiroshi Ishii, Kim J Vicente, and William AS Buxton, 1995. “Transparent layered user interfaces: An evaluation of a display design to enhance focused and divided attention”. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 317–324. Cited page 90.

[127] Andrew Head, Amber Xie, and Marti A Hearst, 2022. “Math augmentation: How authors enhance the readability of formulas using novel visual design practices”. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, pages 1–18. Cited page 60.

[128] João F Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista, 2014. “High-speed tracking with kernelized correlation filters”. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 3, pages 583–596. Cited page 97.

[129] Laura M Herman and Stefanie Hutka, 2019. “Virtual artistry: Virtual reality translations of two-dimensional creativity”. In *Proceedings of the 2019 on Creativity and Cognition*, pages 612–618. Cited page 60.

[130] Florian Hoenig and Andrea Interguglielmi, 2022. “Unbound”. https://www.unbound.io/. Cited page 13.

[131] Josh Holinaty, Alec Jacobson, and Fanny Chevalier, 2021. “Supporting reference imagery for digital drawing”. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2434–2442. Cited page 60.

[132] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle, 1992. “Surface reconstruction from unorganized points”. In *Annual conference on computer graphics and interactive techniques (SIGGRAPH)*, pages 71–78. Cited pages 27, 29, and 43.

[133] Alexander Hornung and Leif Kobbelt, 2006. "Robust reconstruction of watertight 3D models from non-uniformly sampled point clouds without normal information". In *Symposium on geometry processing*, pages 41–50. Cited page 29.

[134] Eric Horvitz, 1999. "Principles of mixed-initiative user interfaces". In *Proc. ACM SIGCHI conference on Human Factors in Computing Systems*. Cited page 99.

[135] PC Hsu and C Lee, 2003. "Field functions for blending range controls on soft objects". In *Computer Graphics Forum*, vol. 22, pages 233–242. Wiley Online Library. Cited page 13.

[136] H. Huang, S. Wu, M. Gong, D. Cohen-Or, U. Ascher, and H. Zhang, 2013. "Edge-aware point set resampling". *ACM Transactions on Graphics*, vol. 32. Cited page 29.

[137] Zhiyang Huang, Nathan Carr, and Tao Ju, 2019. "Variational implicit point set surfaces". *ACM Transactions on Graphics (Proc. SIGGRAPH)*, vol. 38, no. 4. Cited pages 27, 29, 31, 41, 42, and 48.

[138] Edwin L Hutchins, James D Hollan, and Donald A Norman, 1985. "Direct manipulation interfaces". *Human–computer interaction*, vol. 1, no. 4, pages 311–338. Cited pages 1, 2, 65, and 94.

[139] Hilary Hutchinson, Wendy Mackay, Bo Westerlund, Benjamin B Bederson, Allison Druin, Catherine Plaisant, Michel Beaudouin-Lafon, Stéphane Conversy, Helen Evans, Heiko Hansen, et al., 2003. "Technology probes: inspiring design for and with families". In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 17–24. Cited pages 60 and 87.

[140] Icosa, 2020. "Open brush". `https://openbrush.app/`. Cited pages 16, 21, 28, 30, 59, 60, 72, and 74.

[141] Takeo Igarashi, Satoshi Matsuoka, Sachiko Kawachiya, and Hidehiko Tanaka, 2007. "Interactive beautification: a technique for rapid geometric design". In *ACM SIGGRAPH 2007 courses*, pages 18–es. ACM, New York, NY, USA. Cited page 17.

[142] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka, 2006. "Teddy: a sketching interface for 3D freeform design". In *ACM SIGGRAPH 2006 Courses*, pages 11–es. Cited pages 15 and 16.

[143] Takeo Igarashi, Tomer Moscovich, and John F Hughes, 2005. "As-rigid-as-possible shape manipulation". *ACM transactions on Graphics (TOG)*, vol. 24, no. 3, pages 1134–1141. Cited page 23.

[144] Riwano Ikeda and Issei Fujishiro, 2021. "SpiCa: Stereoscopic effect design with 3D pottery wheel-type transparent canvas". In *ACM SIGGRAPH Asia 2021 Technical Communications*. Cited pages 21 and 113.

[145] Tim Ingold, 2010. "The textility of making". *Cambridge journal of economics*, vol. 34, no. 1, pages 91–102. Cited page 118.

[146] Savage Interactive, 2011. "Procreate". `https://procreate.com/`. Cited pages 65, 71, and 75.

[147] Hossam Isack and Yuri Boykov, 2012. "Energy-based geometric multi-model fitting". *International journal of computer vision*, vol. 97, no. 2, pages 123–147. Cited pages 27, 32, 35, 36, and 39.

[148] Johann Habakuk Israel, Eva Wiese, Magdalena Mateescu, Christian Zöllner, and Rainer Stark, 2009. "Investigating three-dimensional sketching for early conceptual design —- results from expert discussions and user studies". *Computers & Graphics*, vol. 33, no. 4, pages 462–473. Cited pages 16, 25, 54, and 60.

[149] Allan Jabri, Andrew Owens, and Alexei A Efros, 2020. "Space-time correspondence as a contrastive random walk". *Advances in Neural Information Processing Systems*. Cited page 102.

[150] Bret Jackson and Daniel F Keefe, 2016. "Lift-off: Using reference imagery and freehand sketching to create 3D models in VR". *IEEE transactions on visualization and computer graphics*, vol. 22, no. 4, pages 1442–1451. Cited page 17.

[151] Robert JK Jacob, Audrey Girouard, Leanne M Hirshfield, Michael S Horn, Orit Shaer, Erin Treacy Solovey, and Jamie Zigelbaum, 2008. "Reality-based interaction: a framework for post-WIMP interfaces". In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 201–210. Cited pages 12 and 65.

[152] Jennifer Jacobs, Joel R Brandt, Radomír Meěh, and Mitchel Resnick, 2018. "Dynamic brushes: Extending manual drawing practices with artist-centric programming tools". In *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–4. Cited page 119.

[153] Alec Jacobson, Ilya Baran, Jovan Popovic, and Olga Sorkine, 2011. "Bounded biharmonic weights for real-time deformation." *ACM Trans. Graph.*, vol. 30, no. 4, page 78. Cited pages 12 and 13.

[154] Wenzel Jakob, Marco Tarini, Daniele Panozzo, and Olga Sorkine-Hornung, Nov. 2015. "Instant field-aligned meshes". *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*, vol. 34, no. 6. Cited page 42.

[155] Ghita Jalal, Nolwenn Maudet, and Wendy E Mackay, 2015. "Color portraits: From color picking to interacting with color". In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 4207–4216. Cited pages 60 and 71.

[156] Philipp Jenke, Michael Wand, Wolfgang Straßer, and A AKA, 2008. "Patch-graph reconstruction for piecewise smooth surfaces". In *VMV*, pages 3–12. Citeseer. Cited page 29.

[157] Ying Jiang, Congyi Zhang, Hongbo Fu, Alberto Cannavò, Fabrizio Lamberti, Y K Henry Lau, and Wenping Wang, 2021. "HandPainter - 3D sketching in VR with hand-based physical proxy". In *ACM Conference on Human Factors in Computing Systems (CHI)*. ACM, New York, NY, USA. Cited pages 17, 59, and 74.

[158] David Jourdan, Pierre-Alexandre Hugron, Camille Schreck, Jonàs Martínez, and Sylvain Lefebvre, Dec. 2023. "Shrink & Morph: 3D-printed self-shaping shells actuated by a shape memory effect". *ACM Trans. Graph.*, vol. 42, no. 6. doi:10.1145/3618386. Cited page 120.

[159] Amaury Jung, Stefanie Hahmann, Damien Rohmer, Antoine Begault, Laurence Boissieux, and Marie-Paule Cani, 2015. "Sketching folds: Developable surfaces from non-planar silhouettes". *Acm Transactions on Graphics (TOG)*, vol. 34, no. 5, pages 1–12. Cited page 15.

[160] Robert D Kalnins, Lee Markosian, Barbara J Meier, Michael A Kowalski, Joseph C Lee, Philip L Davidson, Matthew Webb, John F Hughes, and Adam Finkelstein, 2002. "WYSIWYG NPR:

Drawing strokes directly on 3D models". In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 755–762. Cited page 21.

[161] Levent Burak Kara and Kenji Shimada, 2007. "Sketch-based 3D-shape creation for industrial styling design". *IEEE Computer Graphics and Applications*, vol. 27, no. 1, pages 60–71. Cited page 14.

[162] Olga Karpenko, John F Hughes, and Ramesh Raskar, 2002. "Free-form sketching with variational implicit surfaces". In *Computer Graphics Forum*, vol. 21, pages 585–594. Wiley Online Library. Cited page 15.

[163] Yoni Kasten, Dolev Ofri, Oliver Wang, and Tali Dekel, 2021. "Layered neural atlases for consistent video editing". *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*, vol. 40, no. 6, pages 1–12. Cited page 96.

[164] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe, 2006. "Poisson surface reconstruction". In *Proceedings of the fourth Eurographics symposium on Geometry processing*, vol. 7. Cited pages 27 and 29.

[165] Rubaiat Habib Kazi, Fanny Chevalier, Tovi Grossman, Shengdong Zhao, and George Fitzmaurice, 2014. "Draco: Bringing life to illustrations with kinetic textures". In *Proc. ACM SIGCHI Conference on Human Factors in Computing Systems*. Cited page 23.

[166] Daniel Keefe, Robert Zeleznik, and David Laidlaw, 2007. "Drawing on air: Input techniques for controlled 3D line illustration". *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 5, pages 1067–1081. Cited page 17.

[167] KenTools, 2022. "Geotracker". https://keentools.io/products/geotracker-for-after-effects. Cited page 95.

[168] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis, Jul. 2023. "3D Gaussian Splatting for Real-Time Radiance Field Rendering". *ACM Transactions on Graphics*, vol. 42, no. 4, pages 1–14. URL: https://inria.hal.science/hal-04088161, doi:10.1145/3592433. Cited page 91.

[169] Os Keyes, Josephine Hoy, and Margaret Drouhard, 2019. "Human-computer insurrection: Notes on an anarchist HCI". In *Proceedings of the 2019 CHI conference on human factors in computing systems*, pages 1–13. Cited page 115.

[170] Yeojin Kim, Byungmoon Kim, and Young J Kim, 2018. "Dynamic deep octree for high-resolution volumetric painting in virtual reality". In *Computer Graphics Forum*, vol. 37, pages 179–190. Wiley Online Library. Cited pages 74 and 91.

[171] Yongjin Kim, Holger Winnemöller, and Seungyong Lee, 2013. "WYSIWYG stereo painting". In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 169–176. Cited page 113.

[172] Yongkwan Kim, Sang-Gyun An, Joon Hyub Lee, and Seok-Hyung Bae, 2018. "Agile 3D sketching with air scaffolding". In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–12. Cited pages 14, 15, and 54.

[173] Yongkwan Kim and Seok-Hyung Bae, 2016. "SketchingWithHands: 3D sketching handheld products with first-person hand posture". In *ACM Symposium on User Interface Software and Technology (UIST)*. Cited pages 14, 27, 46, and 120.

[174] Felix Klose, Oliver Wang, Jean-Charles Bazin, Marcus Magnor, and Alexander Sorkine-Hornung, 2015. "Sampling based scene-space video processing". *ACM Transactions on Graphics (Proc. SIGGRAPH)*, vol. 34, no. 4, pages 1–11. Cited page 96.

[175] Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo, 2019. "Abc: A big cad model dataset for geometric deep learning". In *IEEE Conference on Computer Vision and Pattern Recognition*. Cited page 49.

[176] Johannes Kopf, Michael F. Cohen, and Richard Szeliski, 2014. "First-person hyper-lapse videos". *ACM Transactions on Graphics (Proc. SIGGRAPH)*, vol. 33, no. 4. Cited page 96.

[177] Johannes Kopf, Kevin Matzen, Suhib Alsisan, Ocean Quigley, Francis Ge, Yangming Chong, Josh Patterson, Jan-Michael Frahm, Shu Wu, Matthew Yu, et al., 2020. "One shot 3D photography". *ACM Transactions on Graphics (Proc. SIGGRAPH)*, vol. 39, no. 4. Cited page 96.

[178] Johannes Kopf, Xuejian Rong, and Jia-Bin Huang, 2021. "Robust consistent video depth estimation". In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Cited pages 94, 95, 96, and 101.

[179] Vojtěch Krs, Ersin Yumer, Nathan Carr, Bedrich Benes, and Radomír Měch, 2017. "Skippy: Single view 3D curve interactive modeling". *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, pages 1–12. Cited pages 14 and 113.

[180] Wolfgang Kruger, C-A Bohn, Bernd Frohlich, Heinrich Schuth, Wolfgang Strauss, and Gerold Wesche, 1995. "The responsive workbench: A virtual work environment". *Computer*, vol. 28, no. 7, pages 42–48. Cited page 16.

[181] Sarah Anne Kushner, Paul H Dietz, and Alec Jacobson, 2022. "Interactive 3D zoetrope with a strobing flashlight". In *Adjunct Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*, pages 1–3. Cited page 21.

[182] Kin Chung Kwan and Hongbo Fu, 2019. "Mobi3Dsketch: 3D sketching in mobile ar". In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–11. Cited pages 17 and 74.

[183] Nick Ladd, 2019. "Advanced quill tutorial - lathes and smooth shapes". https://www.youtube.com/watch?v=FO_GM2Lb1Dw. Cited pages 61 and 67.

[184] Nick Ladd, 2023. "Retropolis 2: Never say goodbye". https://store.steampowered.com/app/2293440/Retropolis_2_Never_Say_Goodbye/. Cited page 59.

[185] Simon Laing and Mark Apperley, 2020. "The relevance of virtual reality to communication design". *Design Studies*, vol. 71, page 100,965. Cited page 60.

[186] Maria Larsson, Hironori Yoshida, and Takeo Igarashi, 2019. "Human-in-the-loop fabrication of 3D surfaces with natural tree branches". In *Proceedings of the 3rd Annual ACM Symposium on*

Computational Fabrication, pages 1–12. Cited page 120.

[187] Jérémy Laviole and Martin Hachet, 2012. "PapARt: interactive 3D graphics and multi-touch augmented paper for artistic creation". In *2012 IEEE symposium on 3D user interfaces (3DUI)*, pages 3–6. IEEE. Cited page 17.

[188] Germán Leiva, Cuong Nguyen, Rubaiat Habib Kazi, and Paul Asente, 2020. "Pronto: Rapid augmented reality video prototyping using sketches and enaction". In *Proc. ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 1–13. Cited pages 23, 96, 99, and 120.

[189] Changjian Li, Hao Pan, Yang Liu, Xin Tong, Alla Sheffer, and Wenping Wang, 2017. "Bendsketch: Modeling freeform surfaces through 2D sketching". *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, pages 1–14. Cited pages 15, 16, and 29.

[190] Changjian Li, Hao Pan, Yang Liu, Xin Tong, Alla Sheffer, and Wenping Wang, 2018. "Robust flow-guided neural prediction for sketch-based freeform surface modeling". *ACM Transactions on Graphics (TOG)*, vol. 37, no. 6, pages 1–12. Cited page 15.

[191] Jingyi Li, Sonia Hashim, and Jennifer Jacobs, 2021. "What we can learn from visual artists about software development". In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–14. Cited pages 10, 60, 118, and 119.

[192] Jingyi Li, Eric Rawn, Jacob Ritchie, Jasper Tran O'Leary, and Sean Follmer, 2023. "Beyond the artifact: Power as a lens for creativity support tools". Cited pages 118 and 119.

[193] Minchen Li, Danny M Kaufman, Vladimir G Kim, Justin Solomon, and Alla Sheffer, 2018. "OptCuts: Joint optimization of surface cuts and parameterization". *ACM Transactions on Graphics (TOG)*, vol. 37, no. 6, pages 1–13. Cited page 19.

[194] Wenbin Li, Fabio Viola, Jonathan Starck, Gabriel J Brostow, and Neill DF Campbell, 2016. "Roto++ accelerating professional rotoscoping using shape manifolds". *ACM Transactions on Graphics (Proc. SIGGRAPH)*, vol. 35, no. 4. Cited page 97.

[195] Yangyan Li, Xiaokun Wu, Yiorgos Chrysathou, Andrei Sharf, Daniel Cohen-Or, and Niloy J Mitra, 2011. "Globfit: Consistently fitting primitives by discovering global relations". In *ACM Transactions on Graphics (Proc. SIGGRAPH)*. Cited page 30.

[196] Yuwei Li, Xi Luo, Youyi Zheng, Pengfei Xu, and Hongbo Fu, 2017. "SweepCanvas: Sketch-based 3D prototyping on an RGB-D image". In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, pages 387–399. Cited pages 14, 15, 99, 113, and 120.

[197] Pengpeng Liang, Yifan Wu, Hu Lu, Liming Wang, Chunyuan Liao, and Haibin Ling, 2018. "Planar object tracking in the wild: A benchmark". In *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. Cited page 97.

[198] Jian Liao, Adnan Karim, Shivesh Singh Jadon, Rubaiat Habib Kazi, and Ryo Suzuki, 2022. "RealityTalk: Real-time speech-driven augmented presentation for AR live storytelling". In *Proc. ACM Symposium on User Interface Software and Technology (UIST)*. Cited page 96.

[199] Geng Lin, Chen Gao, Jia-Bin Huang, Changil Kim, Yipeng Wang, Matthias Zwicker, and Ayush Saraf, October 2023. "OmnimatteRF: Robust omnimatte with 3d background modeling". In

Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV). Cited page 114.

[200] Markus Lipp, Peter Wonka, and Pascal Müller, 2014. "PushPull++". *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, pages 1–9. Cited pages 11 and 12.

[201] Jingyuan Liu, Hongbo Fu, and Chiew-Lan Tai, 2020. "PoseTween: Pose-driven tween animation". In *Proc. ACM Symposium on User Interface Software and Technology (UIST)*. Cited pages 22 and 97.

[202] Sean J Liu, Maneesh Agrawala, Stephen DiVerdi, and Aaron Hertzmann, 2022. "ZoomShop: Depth-aware editing of photographic composition". In *Computer Graphics Forum*, vol. 41, pages 57–70. Wiley Online Library. Cited page 96.

[203] Shaowei Liu, Subarna Tripathi, Somdeb Majumdar, and Xiaolong Wang, 2022. "Joint hand motion and interaction hotspots prediction from egocentric videos". In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3282–3292. Cited page 111.

[204] Jorge Lopez-Moreno, Popov Stefan, Adrien Bousseau, Maneesh Agrawala, and George Drettakis, 2013. "Depicting stylized materials with vector shade trees". *ACM Transactions on Graphics*, vol. 32, no. 4. Cited pages 71 and 75.

[205] Ling Luo, Pinaki Nath Chowdhury, Tao Xiang, Yi-Zhe Song, and Yulia Gryaditskaya, 2023. "3D VR sketch guided 3D shape prototyping and exploration". In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9267–9276. Cited page 18.

[206] Ling Luo, Yulia Gryaditskaya, Tao Xiang, and Yi-Zhe Song, 2022. "Structure-aware 3D VR sketch to 3D shape retrieval". In *2022 International Conference on 3D Vision (3DV)*, pages 383–392. IEEE. Cited page 18.

[207] Xuan Luo, Jia-Bin Huang, Richard Szeliski, Kevin Matzen, and Johannes Kopf, 2020. "Consistent video depth estimation". *ACM Transactions on Graphics (Proc. SIGGRAPH)*, vol. 39, no. 4, pages 71–1. Cited page 96.

[208] Zhongjin Luo, Jie Zhou, Heming Zhu, Dong Du, Xiaoguang Han, and Hongbo Fu, 2021. "SimpModeling: Sketching implicit field to guide mesh modeling for 3D animalmorphic head design". In *The 34th Annual ACM Symposium on User Interface Software and Technology*, UIST '21, page 854–863. Association for Computing Machinery, New York, NY, USA. doi:10.1145/3472749.3474791. Cited pages 15 and 16.

[209] Dor Ma'ayan, Wode Ni, Katherine Ye, Chinmay Kulkarni, and Joshua Sunshine, 2020. "How domain experts create conceptual diagrams and implications for tool design". In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–14. Cited page 60.

[210] Mayra D Barrera Machuca, Paul Asente, Wolfgang Stuerzlinger, Jingwan Lu, and Byungmoon Kim, 2018. "Multiplanes: Assisted freehand VR sketching". In *Proceedings of the Symposium on Spatial User Interaction*, pages 36–47. ACM, New York, NY, USA. Cited pages 17, 28, and 59.

[211] Mayra Donaji Barrera Machuca, Wolfgang Stuerzlinger, and Paul Asente, 2019. "The effect of spatial ability on immersive 3D drawing". In *Proceedings of the ACM Conference on Creativity & Cognition (C&C'19). https://doi. org/10.1145/3325480.3325489*. Cited pages 4, 16, 28, 59, 69, and 70.

[212] Wendy E Mackay, 2023. "DOIT: The design of interactive things. selected methods for quickly and effectively designing interactive systems from the user's perspective". In *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–3. Cited pages 7 and 61.

[213] Edward Madojemu, 2022. "Mescaform hill: The missing five". https://play.mescaformhill.com/project/missing-five. Cited page 59.

[214] Nolwenn Maudet, 2017. *Concevoir les outils numériques du design*. Ph.D. thesis. Thèse de doctorat dirigée par Beaudouin-Lafon, Michel Informatique Université Paris-Saclay (ComUE) 2017. URL: http://www.theses.fr/2017SACLS486. Cited pages 60 and 118.

[215] Maxon, 2023. "ZBrush". https://www.maxon.net/en/zbrush. Cited pages 12 and 53.

[216] Maxon, 2023. "Zbrush - uv map: Unwrap". https://docs.pixologic.com/user-guide/3d-modeling/exporting-your-model/uv-mapping/uv-map-unwrap/. Cited page 19.

[217] Maximilian Mayer, Philipp Trenz, Sebastian Pasewaldt, Mandy Klingbeil, Jürgen Döllner, Matthias Trapp, and Amir Semmo, 2021. "MotionViz: Artistic visualization of human motion on mobile devices". In *ACM SIGGRAPH 2021 Appy Hour*. Cited page 97.

[218] Elie Michel and Tamy Boubekeur, 2021. "DAG amendment for inverse control of parametric shapes". *ACM Transactions on Graphics (TOG)*, vol. 40, no. 4, pages 1–14. Cited pages 13 and 119.

[219] Media Molecule, 2020. "Dreams". https://indreams.me/. Cited pages 13 and 21.

[220] Aron Monszpart, Nicolas Mellado, Gabriel J Brostow, and Niloy J Mitra, 2015. "Rapter: rebuilding man-made scenes with regular arrangements of planes". *ACM Transactions on Graphics (Proc. SIGGRAPH)*, vol. 34, no. 4. Cited page 30.

[221] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos, 2015. "ORB-SLAM: a versatile and accurate monocular slam system". *IEEE Transactions on Robotics*, vol. 31, no. 5. Cited page 96.

[222] Brad A Myers, Ashley Lai, Tam Minh Le, YoungSeok Yoon, Andrew Faulring, and Joel Brandt, 2015. "Selective undo support for painting applications". In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 4227–4236. Cited pages 60, 71, and 75.

[223] Naam, 2017. "Art spotlight: Naam, creating art in oculus quill". https://sketchfab.com/blogs/community/art-spotlight-naam-creating-art-oculus-quill/. Cited pages 61 and 67.

[224] Bonnie Nardi, Bill Tomlinson, Donald J Patterson, Jay Chen, Daniel Pargman, Barath Raghavan, and Birgit Penzenstadler, 2018. "Computing within limits". *Communications of the ACM*, vol. 61, no. 10, pages 86–93. Cited page 114.

[225] Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa, 2006. "Laplacian mesh optimization". In *Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*, pages 381–389. Cited page 40.

[226] Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa, 2007. "FiberMesh: designing freeform surfaces with 3D curves". In *ACM SIGGRAPH 2007 papers*, pages 41–es. Cited pages 15, 29, and 54.

[227] Cuong Nguyen, Yuzhen Niu, and Feng Liu, 2013. "Direct manipulation video navigation in 3D". In *Proc. ACM SIGCHI Conference on Human Factors in Computing Systems*. Cited pages 22 and 96.

[228] NVRMIND, 2018. "Animvr". https://www.meta.com/en-gb/experiences/pcvr/1741124389277542/. Cited pages 16, 21, and 74.

[229] Seoung Wug Oh, Joon-Young Lee, Kalyan Sunkavalli, and Seon Joo Kim, 2018. "Fast video object segmentation by reference-guided mask propagation". In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Cited page 97.

[230] Makoto Okabe, Yasuyuki Matsushita, Takeo Igarashi, and Heung-Yeung Shum, 2006. "Illumination brush: Interactive design of image-based lighting". In *ACM SIGGRAPH 2006 Research posters*, pages 141–es. Cited page 90.

[231] Pauline Olivier, Renaud Chabrier, Damien Rohmer, Eric De Thoisy, and Marie-Paule Cani, 2019. "Nested explorative maps: A new 3D canvas for conceptual design in architecture". *Computers & Graphics*, vol. 82, pages 203–213. Cited pages 14 and 113.

[232] Luke Olsen, Faramarz F Samavati, Mario Costa Sousa, and Joaquim A Jorge, 2009. "Sketch-based modeling: A survey". *Computers & Graphics*, vol. 33, no. 1, pages 85–103. Cited page 14.

[233] GüNay Orbay and Levent Burak Kara, 2012. "Sketch-based surface design using malleable curve networks". *Computers & Graphics*, vol. 36, no. 8, pages 916–929. Cited pages 27 and 28.

[234] Michaël Ortega and Thomas Vincent, 2014. "Direct drawing on 3D shapes with automated camera control". In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2047–2050. Cited pages 20 and 73.

[235] Alfred Oti and Nathan Crilly, 2021. "Immersive 3D sketching tools: Implications for visual thinking and communication". *Computers & Graphics*, vol. 94, pages 111–123. Cited page 60.

[236] Patrick Paczkowski, Min H Kim, Yann Morvan, Julie Dorsey, Holly E Rushmeier, and Carol O'Sullivan, 2011. "Insitu: sketching architectural designs in context." *ACM Trans. Graph.*, vol. 30, no. 6, page 182. Cited page 14.

[237] Hao Pan, Yang Liu, Alla Sheffer, Nicholas Vining, Chang-Jian Li, and Wenping Wang, 2015. "Flow aligned surfacing of curve networks". *ACM Transactions on Graphics (Proc. SIGGRAPH)*, vol. 34, no. 4. Cited pages 28, 49, and 51.

[238] Karran Pandey, J Andreas Bærentzen, and Karan Singh, 2022. "Face extrusion quad meshes". In *ACM SIGGRAPH 2022 Conference Proceedings*, pages 1–9. Cited page 11.

[239] Karran Pandey, Fanny Chevalier, and Karan Singh, 2023. "Juxtaform: interactive visual summarization for exploratory shape design". *ACM Transactions on Graphics (TOG)*, vol. 42, no. 4, pages 1–14. Cited page 119.

[240] Theo Pavlidis and Christopher J Van Wyk, 1985. "An automatic beautifier for drawings and illustrations". *ACM SIGGRAPH Computer Graphics*, vol. 19, no. 3, pages 225–234. Cited page 17.

[241] Hans Køhling Pedersen, 1996. "A framework for interactive texturing on curved surfaces". In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 295–302. Cited page 19.

[242] Daniel Martin Peixe, 2020. "Creating faces in quill with daniel martin peixe (part 1/2)". https://www.youtube.com/watch?v=EqRmMYjp6Tg. Cited pages 61 and 67.

[243] F. Perazzi, J. Pont-Tuset, B. McWilliams, L. Van Gool, M. Gross, and A. Sorkine-Hornung, 2016. "A benchmark dataset and evaluation methodology for video object segmentation". In *Computer Vision and Pattern Recognition*. Cited page 105.

[244] Lohit Petikam, Ken Anjyo, and Taehyun Rhee, 2021. "Shading rig: Dynamic art-directable stylised shading for 3d characters". *ACM Transactions on Graphics (TOG)*, vol. 40, no. 5, pages 1–14. Cited pages 21 and 90.

[245] Trung T. Pham, Markus Eich, Ian Reid, and Gordon Wyeth, 2016. "Geometrically consistent plane extraction for dense indoor 3d maps segmentation". In *IROS*. Cited page 30.

[246] Polysketch, 2023. "Polysketch". https://www.polysketchvr.com/. Cited page 11.

[247] Jordi Pont-Tuset, Federico Perazzi, Sergi Caelles, Pablo Arbeláez, Alexander Sorkine-Hornung, and Luc Van Gool, 2017. "The 2017 DAVIS challenge on video object segmentation". *arXiv:1704.00675*. Cited page 105.

[248] Roi Poranne, Marco Tarini, Sandro Huber, Daniele Panozzo, and Olga Sorkine-Hornung, 2017. "Autocuts: simultaneous distortion and cut optimization for UV mapping". *ACM Transactions on Graphics (TOG)*, vol. 36, no. 6, pages 1–11. Cited page 19.

[249] Narjes Pourjafarian, Fjolla Mjaku, Marion Koelle, Martin Schmitz, Jan Borchers, and Jürgen Steimle, 2023. "Handheld tools unleashed: Mixed-initiative physical sketching with a robotic printer". In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–14. Cited page 120.

[250] Frano Rajič, Lei Ke, Yu-Wing Tai, Chi-Keung Tang, Martin Danelljan, and Fisher Yu, 2023. "Segment anything meets point tracking". *arXiv:2307.01197*. Cited page 114.

[251] Gonzalo Ramos, George Robertson, Mary Czerwinski, Desney Tan, Patrick Baudisch, Ken Hinckley, and Maneesh Agrawala, 2006. "Tumble! splat! helping users access and manipulate occluded content in 2D drawings". In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '06, page 428–435. Association for Computing Machinery, New York, NY, USA. doi:10.1145/1133265.1133351. Cited page 90.

[252] Sverker Rasmuson, Erik Sintorn, and Ulf Assarsson, 2020. "User-guided 3D reconstruction using multi-view stereo". In *Symposium on Interactive 3D Graphics and Games*, I3D '20. Association for Computing Machinery, New York, NY, USA. doi:10.1145/3384382.3384530. Cited page 11.

[253] Alex Rav-Acha, Pushmeet Kohli, Carsten Rother, and Andrew Fitzgibbon, 2008. "Unwrap

mosaics: A new representation for video editing". *ACM Transactions on Graphics (Proc. SIGGRAPH)*. Cited page 96.

[254] Miguel A Renom, Baptiste Caramiaux, and Michel Beaudouin-Lafon, 2023. "Interaction knowledge: Understanding the 'mechanics' of digital tools". In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–14. Cited pages 65 and 83.

[255] Mitchel Resnick, Brad Myers, Kumiyo Nakakoji, Ben Shneiderman, Randy Pausch, Ted Selker, and Mike Eisenberg, 2005. "Design principles for tools to support creative thinking". Cited pages 2 and 25.

[256] Alec Rivers, Andrew Adams, and Frédo Durand, 2012. "Sculpting by numbers". *ACM Transactions on Graphics (TOG)*, vol. 31, no. 6, pages 1–7. Cited page 120.

[257] Alec Rivers, Takeo Igarashi, and Frédo Durand, 2010. "2.5D cartoon models". *ACM Transactions on Graphics (Proc. SIGGRAPH)*, vol. 29, no. 4. Cited page 23.

[258] Scott Robertson and Thomas Bertling, 2014. *How to Render: the fundamentals of light, shadow and reflectivity*. Cited pages 71 and 75.

[259] Zoe Roellin, 2020. "But my granddad still sees gentian". https://zoeroellin.ch/gentian/. Cited page 59.

[260] Enrique Rosales, Chrystiano Araújo, Jafet Rodriguez, Nicholas Vining, Dongwook Yoon, and Alla Sheffer, 2021. "AdaptiBrush: Adaptive general and predictable VR ribbon brush". *ACM Transaction on Graphics (Proc. SIGGRAPH Asia)*, vol. 40, no. 1. Cited pages 30 and 74.

[261] Enrique Rosales, Jafet Rodriguez, and Alla Sheffer, 2019. "SurfaceBrush: From virtual reality drawings to manifold surfaces". *ACM Transaction on Graphics*, vol. 38, no. 4. doi:https://doi.org/10.1145/3306346.3322970. Cited page 30.

[262] Runway, 2022. "Runwayml". https://app.runwayml.com/. Cited page 95.

[263] Emanuel Sachs, Andrew Roberts, and David Stoops, 1991. "3-Draw: A tool for designing 3D shapes". *IEEE Computer Graphics and Applications*, , no. 6, pages 18–26. Cited page 16.

[264] Bardia Sadri and Karan Singh, 2014. "Flow-complex-based shape reconstruction from 3d curves". *ACM Transactions on Graphics (TOG)*, vol. 33, no. 2. Cited pages 27 and 28.

[265] Stephanie Santosa, Fanny Chevalier, Ravin Balakrishnan, and Karan Singh, 2013. "Direct space-time trajectory control for visual media editing". In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1149–1158. Cited page 22.

[266] Nazmus Saquib, Rubaiat Habib Kazi, Li-Yi Wei, and Wilmot Li, 2019. "Interactive body-driven graphics for augmented video performance". In *Proc. ACM CHI Conference on Human Factors in Computing Systems*. Cited page 97.

[267] Steven Schkolne, Michael Pruett, and Peter Schröder, 2001. "Surface drawing: creating organic 3d shapes with the hand and tangible tools". In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 261–268. Cited pages 4, 16, 17, 21, 59, and 74.

[268] Johannes Schmid, Martin Sebastian Senn, Markus Gross, and Robert W Sumner, 2011.

"OverCoat: an implicit canvas for 3D painting". In *ACM SIGGRAPH 2011 papers*, pages 1–10. Cited pages 21 and 75.

[269] Ryan Schmidt, Cindy Grimm, and Brian Wyvill, 2006. "Interactive decal compositing with discrete exponential maps". In *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, page 605–613. Association for Computing Machinery, New York, NY, USA. doi:10.1145/1179352.1141930. Cited page 19.

[270] Ryan Schmidt, Azam Khan, Gord Kurtenbach, and Karan Singh, 2009. "On expert performance in 3D curve-drawing tasks". In *Proc. Symposium on Sketch-Based Interfaces and Modeling (SBIM)*. Cited page 100.

[271] Ryan Schmidt, Azam Khan, Karan Singh, and Gord Kurtenbach, 2009. "Analytic drawing of 3D scaffolds". *ACM transactions on graphics (TOG)*, vol. 28, no. 5, pages 1–10. Cited pages 14 and 28.

[272] Ryan Schmidt, Brian Wyvill, Mario Costa Sousa, and Joaquim A Jorge, 2007. "Shapeshop: Sketch-based solid modeling with blobtrees". In *ACM SIGGRAPH 2007 courses*, pages 43–es. Cited page 15.

[273] Ruwen Schnabel, Patrick Degener, and Reinhard Klein, 2009. "Completion and reconstruction with primitive shapes". In *Computer Graphics Forum*, vol. 28. Cited page 29.

[274] Johannes Lutz Schönberger and Jan-Michael Frahm, 2016. "Structure-from-Motion revisited". In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Cited pages 94, 95, 96, and 101.

[275] Silvia Sellán and Alec Jacobson, 2022. "Stochastic poisson surface reconstruction". *ACM Transactions on Graphics*. Cited page 54.

[276] Ticha Sethapakdi, Daniel Anderson, Adrian Reginald Chua Sy, and Stefanie Mueller, 2021. "Fabricaide: Fabrication-aware design for 2D cutting machines". In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–12. Cited page 120.

[277] Vadim Shapiro, 2002. "Solid modeling." *Handbook of computer aided geometric design*, vol. 20, pages 473–518. Cited page 13.

[278] Vishal Sharma, Neha Kumar, and Bonnie Nardi, 2023. "Post-growth human–computer interaction". *ACM Transactions on Computer-Human Interaction*. Cited page 114.

[279] Alla Sheffer and John C Hart, 2002. "Seamster: inconspicuous low-distortion texture seam layout". In *IEEE Visualization, 2002. VIS 2002.*, pages 291–298. IEEE. Cited pages 19 and 73.

[280] Jianbo Shi and Jitendra Malik, 2000. "Normalized cuts and image segmentation". *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 8. Cited page 39.

[281] Evan Shimizu, Matt Fisher, Sylvain Paris, and Kayvon Fatahalian, 2019. "Finding layers using hover visualizations". In *Proceedings of the 45th Graphics Interface Conference on Proceedings of Graphics Interface 2019*, pages 1–9. Cited page 90.

[282] Ben Shneiderman, 1983. "Direct manipulation: A step beyond programming languages". *Computer*, vol. 16, no. 08, pages 57–69. Cited page 1.

[283] Maria Shugrina, 2021. *The design of playful and intelligent creative tools.* Ph.D. thesis. Cited page 114.

[284] Maria Shugrina, Chin-Ying Li, and Sanja Fidler, 2022. "Neural brushstroke engine: Learning a latent style space of interactive drawing tools". *ACM Transactions on Graphics (TOG)*, vol. 41, no. 6. Cited page 119.

[285] Maria Shugrina, Jingwan Lu, and Stephen Diverdi, 2017. "Playful palette: an interactive parametric color mixer for artists". *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, pages 1–10. Cited pages 60, 71, and 119.

[286] Karan Singh and Eugene Fiume, 1998. "Wires: a geometric deformation technique". In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 405–414. Cited pages 12 and 54.

[287] Gravity Sketch, 2017. "Gravity sketch". https://www.gravitysketch.com/. Cited pages 11, 16, 17, 28, and 74.

[288] Harrison Jesse Smith, Qingyuan Zheng, Yifei Li, Somya Jain, and Jessica K Hodgins, 2023. "A method for animating children's drawings of the human figure". *ACM Transactions on Graphics*, vol. 42, no. 3, pages 1–15. Cited pages 22 and 23.

[289] Smoothstep, 2021. "Quill". https://quill.art/. Cited pages 16, 21, 23, 28, 59, 61, 72, 74, and 75.

[290] Noah Snavely, C Lawrence Zitnick, Sing Bing Kang, and Michael Cohen, 2006. "Stylizing 2.5-D video". In *Proc. Symposium on Non-Photorealistic Animation and Rendering*. Cited page 96.

[291] Olga Sorkine and Marc Alexa, 2007. "As-rigid-as-possible surface modeling". In *Symposium on Geometry processing*, vol. 4, pages 109–116. Citeseer. Cited page 12.

[292] Olga Sorkine and Daniel Cohen-Or, 2004. "Least-squares meshes". In *Proceedings Shape Modeling Applications, 2004.*, pages 191–199. IEEE. Cited page 42.

[293] Tibor Stanko, Stefanie Hahmann, Georges-Pierre Bonneau, and Nathalie Saguin-Sprynski, 2016. "Smooth interpolation of curve networks with surface normals". In *Eurographics 2016 Short Papers*, pages 21–24. Eurographics Association. Cited page 28.

[294] Tibor Stanko, Stefanie Hahmann, Georges-Pierre Bonneau, and Nathalie Saguin-Sprynski, 2017. "Shape from sensors: Curve networks on surfaces from 3D orientations". *Computers & Graphics (Proc. SMI)*, vol. 66. Cited page 104.

[295] Qingkun Su, Xue Bai, Hongbo Fu, Chiew-Lan Tai, and Jue Wang, 2018. "Live sketch: Video-driven dynamic deformation of static drawings". In *Proc. ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 1–12. Cited pages 22, 23, 97, 102, and 110.

[296] Jiatian Sun, Longxiulin Deng, Triantafyllos Afouras, Andrew Owens, and Abe Davis, 2023. "Eventfulness for interactive video alignment". *ACM Transactions on Graphics (TOG)*, vol. 42, no. 4, pages 1–10. Cited page 113.

[297] Lingyun Sun, Yue Yang, Yu Chen, Jiaji Li, Danli Luo, Haolin Liu, Lining Yao, Ye Tao, and Guanyun Wang, 2021. "ShrinCage: 4D printing accessories that self-adapt". In *Proceedings of the*

2021 CHI Conference on Human Factors in Computing Systems, pages 1–12. Cited page 120.

[298] Qian Sun, Long Zhang, Minqi Zhang, Xiang Ying, Shi-Qing Xin, Jiazhi Xia, and Ying He, 2013. "Texture brush: an interactive surface texturing interface". In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 153–160. Cited page 19.

[299] Hemant Bhaskar Surale, Aakar Gupta, Mark Hancock, and Daniel Vogel, 2019. "Tabletinvr: Exploring the design space for using a multi-touch tablet in virtual reality". In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, page 1–13. Association for Computing Machinery. doi:10.1145/3290605.3300243. Cited page 11.

[300] Ivan E Sutherland, 1963. "Sketchpad: A man-machine graphical communication system". In *Proceedings of the May 21-23, 1963, spring joint computer conference*, pages 329–346. Cited page 1.

[301] Ivan E Sutherland, 1968. "A head-mounted three dimensional display". In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pages 757–764. Cited page 15.

[302] Ryo Suzuki, Rubaiat Habib Kazi, Li-yi Wei, Stephen DiVerdi, Wilmot Li, and Daniel Leithinger, 2020. "RealitySketch: Embedding responsive graphics and visualizations in AR through dynamic sketching". In *Proc. ACM Symposium on User Interface Software and Technology (UIST)*. Cited page 96.

[303] Andrea Tagliasacchi, Hao Zhang, and Daniel Cohen-Or, 2009. "Curve skeleton extraction from incomplete point cloud". In *ACM SIGGRAPH 2009 papers*, pages 1–9. Cited page 29.

[304] Yuka Takahashi, Tsukasa Fukusato, and Takeo Igarashi, 2019. "Paintersview: Automatic suggestion of optimal viewpoints for 3d texture painting". In *SIGGRAPH Asia 2019 Technical Briefs*, pages 99–102. Cited pages 20 and 73.

[305] Jianchao Tan, Marek Dvorožňák, Daniel Sýkora, and Yotam Gingold, Jul. 2015. "Decomposing time-lapse paintings into layers". *ACM Transactions on Graphics (TOG)*, vol. 34, no. 4, pages 61:1–61:10. URL: http://doi.acm.org/10.1145/2766960, doi:10.1145/2766960. Cited page 75.

[306] Jianchao Tan, Jyh-Ming Lien, and Yotam Gingold, Nov. 2016. "Decomposing images into layers via RGB-space geometry". *ACM Transactions on Graphics (TOG)*, vol. 36, no. 1. URL: http://doi.acm.org/10.1145/2988229, doi:10.1145/2988229. Cited page 75.

[307] T. Tasdizen, J.-P. Tarel, and D.B. Cooper, 2000. "Improving the stability of algebraic curves for applications". *IEEE Transactions on Image Processing*, vol. 9, no. 3. Cited pages 37 and 38.

[308] Gabriel Taubin, 1993. "An improved algorithm for algebraic curve and surface fitting". In *1993 (4th) International Conference on Computer Vision*, pages 658–665. IEEE. Cited page 35.

[309] Zachary Teed and Jia Deng, 2020. "RAFT: Recurrent all-pairs field transforms for optical flow". In *European Conference on Computer Vision (ECCV)*, pages 402–419. Cited pages 95 and 101.

[310] Matthew Thorne, David Burke, and Michiel Van De Panne, 2004. "Motion doodles: an interface for sketching character motion". *ACM Transactions on Graphics (ToG)*, vol. 23, no. 3, pages 424–431. Cited page 22.

[311] James Townsend, Niklas Koep, and Sebastian Weichwald, 2016. "Pymanopt: A python toolbox for optimization on manifolds using automatic differentiation". *Journal of Machine Learning Research*, vol. 17, no. 137, page 1–5. Cited page 105.

[312] Anil Usumezbas, Ricardo Fabbri, and Benjamin B. Kimia, 2017. "The surfacing of multiview 3d drawings via lofting and occlusion reasoning". In *IEEE Conference on Computer Vision and Pattern Recognition*. Cited page 30.

[313] Rodolphe Vaillant, Loïc Barthe, Gaël Guennebaud, Marie-Paule Cani, Damien Rohmer, Brian Wyvill, Olivier Gourmel, and Mathias Paulin, 2013. "Implicit skinning: Real-time skin deformation with contact modeling". *ACM Transactions on Graphics (TOG)*, vol. 32, no. 4, pages 1–12. Cited page 21.

[314] Julien Valentin, Adarsh Kowdle, Jonathan T Barron, Neal Wadhwa, Max Dzitsiuk, Michael Schoenberg, Vivek Verma, Ambrus Csaszar, Eric Turner, Ivan Dryanovski, et al., 2018. "Depth from motion for smartphone AR". *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*, vol. 37, no. 6, pages 1–19. Cited page 96.

[315] W. Vaughan, 2012. *Digital Modeling*. [digital] Series. New Riders. URL: https://books.google.co.in/books?id=nzJ2QgAACAAJ. Cited page 11.

[316] The Verge, 2021. "Meta's oculus quest 2 has shipped 10 million units, according to qualcomm". https://www.theverge.com/2021/11/16/22785469/meta-oculus-quest-2-10-million-units-sold-qualcomm-xr2. Cited page 1.

[317] Floor Verhoeven and Olga Sorkine-Hornung, 2019. "RodMesh: Two-handed 3D surface modeling in virtual reality". Cited page 54.

[318] Philipp Wacker, Adrian Wagner, Simon Voelker, and Jan Borchers, 2018. "Physical guides: An analysis of 3d sketching performance on physical objects in augmented reality". In *Proceedings of the 2018 ACM Symposium on Spatial User Interaction*, pages 25–35. Cited page 17.

[319] Ludwig Wilhelm Wall, Alec Jacobson, Daniel Vogel, and Oliver Schneider, 2021. "Scrappy: Using scrap material as infill to make fabrication more sustainable". In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–12. Cited page 120.

[320] Qianqian Wang, Yen-Yu Chang, Ruojin Cai, Zhengqi Li, Bharath Hariharan, Aleksander Holynski, and Noah Snavely, 2023. "Tracking everything everywhere all at once". In *International Conference on Computer Vision*. Cited page 114.

[321] Andrew M Webb, Andruid Kerne, Zach Brown, Jun-Hyun Kim, and Elizabeth Kellogg, 2016. "Layerfish: Bimanual layering with a fisheye in-place". In *Proceedings of the 2016 ACM International Conference on Interactive Surfaces and Spaces*, pages 189–198. Cited page 90.

[322] Gerold Wesche and Hans-Peter Seidel, 2001. "FreeDrawer: a free-form sketching system on the responsive workbench". In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 167–174. Cited page 16.

[323] Eva Wiese, Johann Habakuk Israel, Achim Meyer, and Sara Bongartz, 2010. "Investigating the learnability of immersive free-hand sketching". In *Proceedings of the seventh sketch-based interfaces and modeling symposium*, pages 135–142. Cited pages 59 and 69.

[324] Nora S. Willett, Wilmot Li, Jovan Popovic, Floraine Berthouzoz, and Adam Finkelstein, 2017. "Secondary motion for performed 2D animation". In *Proc. ACM Symposium on User Interface Software and Technology (UIST)*. Cited page 113.

[325] Karl D.D. Willis, Cheng Xu, Kuan-Ju Wu, Golan Levin, and Mark D. Gross, 2010. "Interactive fabrication: New interfaces for digital fabrication". In *Proceedings of the Fifth International Conference on Tangible, Embedded, and Embodied Interaction*, TEI '11, page 69–72. Association for Computing Machinery, New York, NY, USA. doi:10.1145/1935701.1935716. Cited page 120.

[326] Jianhua Wu and Leif Kobbelt, 2005. "Structure recovery via hybrid variational surface approximation". *Computer Graphics Forum*, vol. 24, no. 3. Cited page 30.

[327] Shanel Wu and Laura Devendorf, 2020. "Unfabricate: designing smart textiles for disassembly". In *proceedings of the 2020 CHI conference on human factors in computing systems*, pages 1–14. Cited page 120.

[328] Haijun Xia, Bruno Araujo, Tovi Grossman, and Daniel Wigdor, 2016. "Object-oriented drawing". In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 4610–4621. Cited page 90.

[329] Zhijie Xia, Kyzyl Monteiro, Kevin Van, and Ryo Suzuki, 2023. "RealityCanvas: Augmented reality sketching for embedded and responsive scribble animation effects". In *Proc. ACM Symposium on User Interface Software and Technology (UIST)*. Cited page 114.

[330] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox, 2018. "PoseCNN: A convolutional neural network for 6D object pose estimation in cluttered scenes". *Robotics: Science and Systems (RSS)*. Cited page 97.

[331] Chufeng Xiao, Wanchao Su, Jing Liao, Zhouhui Lian, Yi-Zhe Song, and Hongbo Fu, 2022. "DifferSketching: How differently do people sketch 3D objects?" *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH Asia 2022)*, vol. 41, no. 4, pages 1–16. Cited page 14.

[332] Jun Xing, Li-Yi Wei, Takaaki Shiratori, and Koji Yatani, 2015. "Autocomplete hand-drawn animations". *ACM Transactions on Graphics (TOG)*, vol. 34, no. 6, pages 1–11. Cited page 23.

[333] Baoxuan Xu, William Chang, Alla Sheffer, Adrien Bousseau, James McCrae, and Karan Singh, 2014. "True2Form: 3D curve networks from 2D sketches via selective regularization". Cited pages 14, 27, 28, and 46.

[334] Pengfei Xu, Hongbo Fu, Youyi Zheng, Karan Singh, Hui Huang, and Chiew-Lan Tai, 2018. "Model-guided 3D sketching". *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 10, pages 2927–2939. Cited page 14.

[335] Dong-Ming Yan, Wenping Wang, Yang Liu, and Zhouwang Yang, 2012. "Variational mesh segmentation via quadric surface fitting". *Computer-Aided Design*, vol. 44, no. 11. Cited page 30.

[336] Hui Ye, Kin Chung Kwan, and Hongbo Fu, 2021. "3D curve creation on and around physical objects with mobile AR". *IEEE transactions on visualization and computer graphics*, vol. 28, no. 8, pages 2809–2821. Cited page 17.

[337] Emilie Yu, Rahul Arora, Tibor Stanko, J Andreas Bærentzen, Karan Singh, and Adrien

Bousseau, 2021. "CASSIE: Curve and surface sketching in immersive environments". In *ACM Conference on Human Factors in Computing Systems (CHI)*, pages 1–14. Cited pages 17, 28, 46, 54, and 59.

[338] Xue Yu, Stephen DiVerdi, Akshay Sharma, and Yotam Gingold, 2021. "ScaffoldSketch: Accurate industrial design drawing in vr". In *Proceedings of ACM Symposium on User Interface Software and Technology*, UIST. Cited pages 17, 28, and 59.

[339] Cédric Zanni, Frédéric Claux, and Sylvain Lefebvre, May 2018. "HCSG: Hashing for real-time CSG modeling". In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. Montreal, Canada. URL: https://inria.hal.science/hal-01792866, doi:10.1145/3203198. Cited pages 13 and 81.

[340] Jiayi Eris Zhang, Seungbae Bang, David IW Levin, and Alec Jacobson, 2020. "Complementary dynamics". *ACM Transactions on Graphics (TOG)*, vol. 39, no. 6, pages 1–11. Cited page 113.

[341] Sharon Zhang, Jiaju Ma, Jiajun Wu, Daniel Ritchie, and Maneesh Agrawala, 2023. "Editing motion graphics video via motion vectorization and transformation". *arXiv preprint arXiv:2309.14642*. Cited page 21.

[342] Xiuming Zhang, Tali Dekel, Tianfan Xue, Andrew Owens, Qiurui He, Jiajun Wu, Stefanie Mueller, and William T Freeman, 2018. "Mosculp: Interactive visualization of shape and time". In *Proc. ACM Symposium on User Interface Software and Technology (UIST)*. Cited page 97.

[343] Y Zhang, C Nguyen, RH Kazi, and LF Yu, 2023. "PoseVEC: Authoring adaptive pose-aware effects using visual programming and demonstrations". In *ACM Symposium on User Interface Software and Technology*. Cited page 113.

[344] Zhoutong Zhang, Forrester Cole, Richard Tucker, William T Freeman, and Tali Dekel, 2021. "Consistent depth of moving objects in video". *ACM Transactions on Graphics (Proc. SIGGRAPH)*, vol. 40, no. 4, pages 1–12. Cited pages 94 and 96.

[345] Haisen Zhao, Max Willsey, Amy Zhu, Chandrakana Nandi, Zachary Tatlock, Justin Solomon, and Adriana Schulz, mar 2022. "Co-optimization of design and fabrication plans for carpentry". *ACM Trans. Graph.*, vol. 41, no. 3. doi:10.1145/3508499. Cited page 120.

[346] Nanxuan Zhao, Nam Wook Kim, Laura Mariah Herman, Hanspeter Pfister, Rynson WH Lau, Jose Echevarria, and Zoya Bylinskii, 2020. "Iconate: Automatic compound icon generation and ideation". In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–13. Cited page 60.

[347] Rebecca Zheng, Marina Fernández Camporro, Hugo Romat, Nathalie Henry Riche, Benjamin Bach, Fanny Chevalier, Ken Hinckley, and Nicolai Marquardt, 2021. "Sketchnote components, design space dimensions, and strategies for effective visual note taking". In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–15. Cited page 60.

[348] Yang Zhou, Kangxue Yin, Hui Huang, Hao Zhang, Minglun Gong, and Daniel Cohen-Or, 2015. "Generalized cylinder decomposition". *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*, vol. 34, no. 6. Cited pages 30 and 53.

[349] Yixin Zhuang, Ming Zou, Nathan Carr, and Tao Ju, 2013. "A general and efficient method for finding cycles in 3d curve networks". *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*, vol. 32, no. 6. Cited pages 27 and 28.

[350] Amit Zoran and Joseph A Paradiso, 2013. "FreeD: a freehand digital sculpting tool". In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 2613–2616. Cited page 120.

[351] Shoshana Zuboff, 2023. "The age of surveillance capitalism". In *Social Theory Re-Wired*, pages 203–213. Routledge. Cited page 1.