UNIVERSITÉ
CÔTE D'AZUR

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

THÈSE DE DOCTORAT

Interprétation et Génération de Représentations Artistiques: Applications à la Modélisation par le Dessin et à la Stylisation de Vidéos

Johanna Delanoy

Inria Sophia Antipolis-Méditerannée

Présentée en vue de l'obtention du grade de docteur en Informatique d'Université Côte d'Azur
Dirigée par : Adrien Bousseau
Soutenue le : 4 Juin 2019

Devant le jury composé de :
Pierre Bénard, Maître de conférence, Université de Bordeaux
Adrien Bousseau, Chargé de Recherche, Inria Sophia Antipolis - Méditerranée
Frédéric Precioso, Professeur, Université Nice Sophia Antipolis
Daniel Sýkora, Professeur, Czech Technical University in Prague
Tien-Tsin Wong, Professeur, The Chinese University of Hong Kong

informatiques mathématiques
Inria

Interprétation et Génération de Représentations Artistiques: Applications à la Modélisation par le Dessin et à la Stylisation de Vidéos

---

Interpreting and Generating Artistic Depictions: Applications to Sketch-Based Modeling and Video Stylization

Jury:

Président du jury / President of the jury

Frédéric Precioso, Professeur, Université Nice Sophia Antipolis

Rapporteurs / Reviewers

Daniel Sýkora, Professeur, Czech Technical University in Prague

Tien-Tsin Wong, Professeur, The Chinese University of Hong Kong

Examinateurs / Examiners

Pierre Bénard, Maître de conférence, Université de Bordeaux

Directeur de thèse / Thesis supervisor

Adrien Bousseau, Chargé de Recherche, Inria Sophia Antipolis - Méditerranée

Acknowledgements

Résumé

Les outils digitaux ouvrent de nouvelles voies de création, aussi bien pour les artistes chevronnés que pour tout autre individu qui souhaite créer. Dans cette thèse, je m'intéresse à deux aspects complémentaires de ces outils : interpréter une création existante et générer du nouveau contenu.

Dans une première partie, j'étudie comment interpréter un dessin comme un objet 3D. Nous proposons une approche basée donnée qui aborde cette problématique en entrainant des réseaux convolutifs profonds (CNN) à prédire l'occupation d'une grille de voxels à partir de dessins. Nous intégrons ces CNNs dans un système de modélisation interactif qui permet à l'utilisateur de dessiner un objet, tourner autour pour voir sa reconstruction 3D et le raffiner en redessinant depuis une nouvelle vue. Nous complémentons cette approche par une méthode géométrique qui permet d'améliorer la qualité de l'objet final. Pour cela, nous entrainons un CNN à prédire des cartes de normales à plus haute résolution depuis chaque vue d'entrée. Nous fusionnons alors ces cartes de normales avec la grille de voxel en optimisant pour la surface finale. Nous entrainons l'ensemble de ces réseaux grâce à des rendus de contours d'objets abstraits générés procéduralement.

Dans une seconde partie, je présente une méthode pour générer des vidéos stylisées faisant penser à de l'animation traditionnelle. La plupart des méthodes existantes gardent le mouvement 3D originel de la vidéo, produisant un résultat ressemblant plus à une scène 3D couverte de peinture qu'à une peinture 2D de la scène. Inspirés par l'animation "cut-out", nous proposons de modifier le mouvement de la séquence afin qu'il soit composé de mouvements rigides en 2D. Pour y parvenir, notre approche segmente le mouvement et l'optimise afin d'approximer au mieux le flot optique d'entrée avec des transformations rigides par morceaux, et re-rend la vidéo de façon à ce que son contenu suive ce mouvement simplifié. En appliquant les méthodes de stylisations existantes sur notre nouvelle séquence, on obtient une vidéo stylisée plus proche d'une animation 2D.

Ces deux parties reposent sur des méthodes différentes mais toutes deux s'appuyent sur les techniques traditionnelles utilisées par les artistes : soit en comprenant comment ils dessinent un objet, soit en s'inspirant de leur façon de simplifier le mouvement.

Mots-clés: outils digitaux artistiques - création digitale - modélisation par le dessin - stylisation de vidéo

Abstract

Digital tools brings new ways of creation, for accomplished artists as well as for any individual willing to create. In this thesis, I am interested in two different aspects in helping artists: interpreting their creation and generating new content.

I first study how to interpret a sketch as a 3D object. We propose a data-driven approach that tackles this challenge by training deep convolutional neural networks (CNN) to predict occupancy of a voxel grid from a line drawing. We integrate our CNNs in an interactive modeling system that allows users to seamlessly draw an object, rotate it to see its 3D reconstruction, and refine it by re-drawing from another vantage point using the 3D reconstruction as guidance. We then complement this technique with a geometric method that allows to refine the quality of the final object. To do so, we train an additional CNN to predict higher resolution normal maps from each input view. We then fuse these normal maps with the voxel grid prediction by optimizing for the final surface. We train all of these networks by rendering synthetic contour drawings from procedurally generated abstract shapes.

In a second part, I present a method to generate stylized videos with a look reminiscent of traditional 2D animation. Existing stylization methods often retain the 3D motion of the original video, making the result look like a 3D scene covered in paint rather than a 2D painting of a scene. Inspired by cut-out animation, we propose to modify the motion of the sequence so that it is composed of 2D rigid motions. To achieve this goal, our approach applies motion segmentation and optimization to best approximate the input optical flow with piecewise-rigid transforms, and re-renders the video such that its content follows the simplified motion. Applying existing stylization algorithm to the new sequence produce a stylized video more similar to 2D animation.

Although the two parts of my thesis lean on different methods, they both rely on traditional techniques used by artists: either by understanding how they draw objects or by taking inspiration from how they simplify the motion in 2D animation.

Keywords: artistic digital tools - digital creation - sketch-based modeling - video stylization

Contents

Chapter 1

Introduction

From the seminal SketchPad [Sut64] up to recent tactile tablets, more and more artists use digital tools in their creation process. Digital tools bring new opportunities to assist artistic creation: developing new ways of creation (e.g. 3D sketching in virtual reality [Goo16]), emulating traditional medias with digital tools (e.g. digital painting or drawing [SLKD16]) or simplifying existing creation process (e.g drawing in-between frames for animation [DLKS18b]). In this thesis, I will discuss how digital tools can ease creation for accomplished or novice artists.

Accomplished artists often master traditional techniques. Digital tools can help them in the exploration of different choices (design space exploration [ADN$^+$17], color exploration [TEG18, MVH$^+$17]) or by automating tedious tasks that do not rely so much on creativity (physical simulation [DBB$^+$17, BBRF14], 3D modeling [DAI$^+$18, LPL$^+$18], in-between image generations [DLKS18b], sketch beautification and clean up [LRS18, LLW17]). The main challenge is thus to find the right scope for the task and to insert it in the traditional workflow that artists use. Successful integration requires to have a seamless transition between the automated or assisted task and the rest of the creative process.

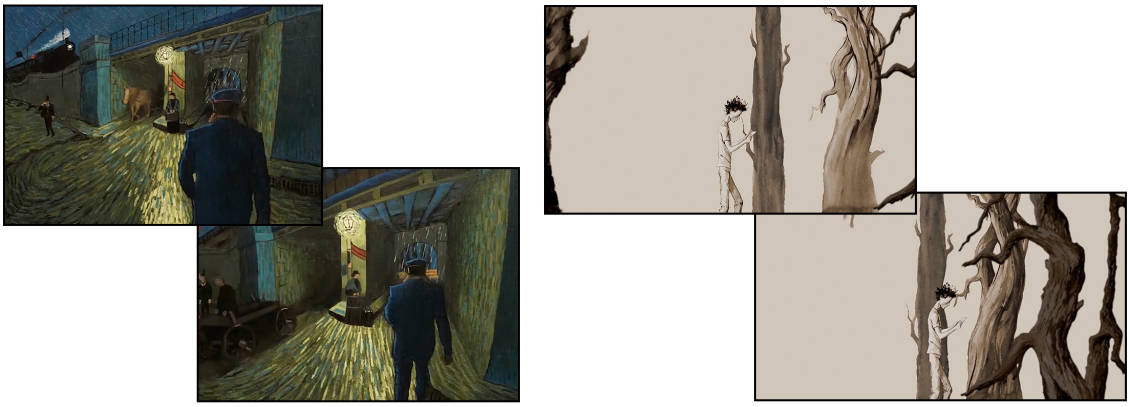
On the other hand, novice artists are often not fully familiar with traditional techniques. Digital tools then have the role of helping them to create despite their lack of skills. To account for this, the tools need to be robust to rough or erroneous inputs. Digital tools can also be made to help the user learn a specific technique (e.g. assistive tools for sketching [LZC11, IBT13, HLW$^+$17]), or allow novices to produce artistic depictions they can't create manually (stylization [SLKD16, GEB16]).

Overall digital tools target two fundamental tasks:

- Generating new content (a static or animated image, a 3D object)
- Interpreting existing content (a sketch, a painting)

Generating artistic depictions

Generating a **realistic** depiction has been well automated for a long time (photography, photo-realistic rendering) but obtaining an **artistic** depiction still involves significant manual work. This is a tedious task even for well accomplished artists. For example, to make the movie *Loving Vincent*, more than 125 artists had to paint each frame of the movie for 4 years. Figure 1.1 shows frames from *Loving Vincent* and *Happy Time*, a short movie realized by an artist specialized in watercolor animations. On the other hand, more and more novices want to produce artistic images but lack the skills to achieve pleasant results by hand. With the tremendous amount of visual content that is created nowadays, there is thus a real need to automate or ease the creation of such images, for all kind of users.



© Loving Vincent

© Happy Time, Skazka Studio

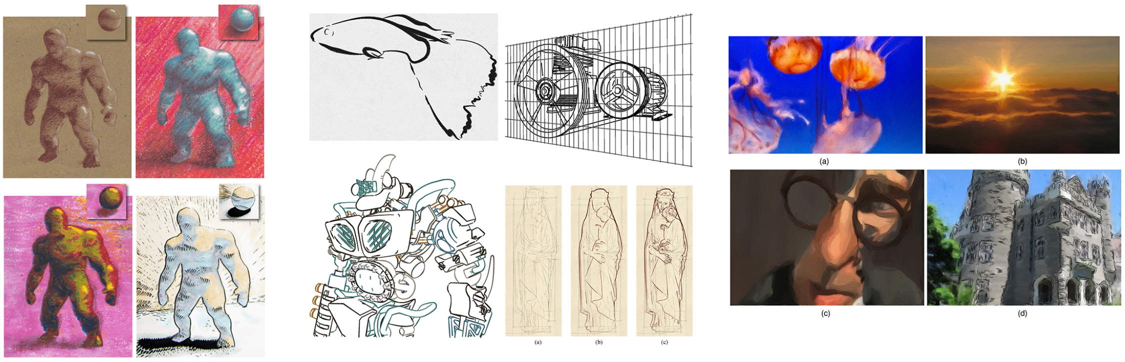
Figure 1.1: Two examples of animated movies made by manually painting each frame. *Loving Vincent* is a full-length movie using oil paint. *Happy Time* is a short movie made with watercolor.

The first question to ask ourselves is: what is an artistic depiction, what defines its style? For static images, the style is mostly defined by the appearance of medias such as traditional oil paint, pencil, charcoal or watercolor but also photography through image filtering and manipulation. In animated sequence, the motion is also usually stylized, for example with squash and stretch effects in cartoon animation or the restriction to planar motions in cut-out animation.

Static stylization

The role of digital stylization is to mimic traditional styles and accelerate the creation of content. The input of these tools can be either 3D or 2D. In both cases, the goal is to automatically produce an image that shows the original content in a stylized way. This can be done either by distributing style elements over the image (e.g. brush strokes) or by synthesizing texture (from an examplar or from parameters). Example results from a few existing methods are shown in Figure 1.2.

Non Photorealistic Rendering is widely used to recreate the traditional look of Japanese anime, more rarely to mimic other traditional medias such as painting or drawing. One reason might be the difficulty to control such styles for artists. Some recent efforts have been made to provide them with programmable rendering pipelines that allows to create such styles easily [MSS+18, GTDS10] but currently no industrial solution exist that handles a large enough variety of style with easy controls.



(a) Stylit, 2016

(b) Freestyle, 2010

(c) AniPaint, 2012

Figure 1.2: Example results from various stylization algorithm: a) StyLit [FJL+16] takes as input an examplar painting of a sphere and a rendering of a scene, b) Freestyle [GTDS10] is a programmable line rendering tool integrated in Blender, c) in Anipaint [OH12], brush strokes are distributed over frames of a video according to a user-guided flow.

Animated stylization

Applying stylization algorithms on animated images adds the extra challenge of motion. A common approach is to make the style follows the optical flow of the input video to ensure temporal consistency [Lit97, BCK+13, BNTS07, OH12, SED16, RDB18]. Unfortunately, following the optical flow too strictly does not reproduce the typical look of

2D animation and gives the impression to look at a 3D painted world. Several methods have thus been proposed to incorporate a controlled amount of noise [FLJ+14, KP11] so that the result looks less fake or to stylize the motion in itself [WDAC06, BSM+07]. However, the field has still not reach a point where the results are not distinguishable from man-made 2D animations.

Style models

There is a large variety of ways to model a given style. We can classify the existing approaches into roughly two categories of models:

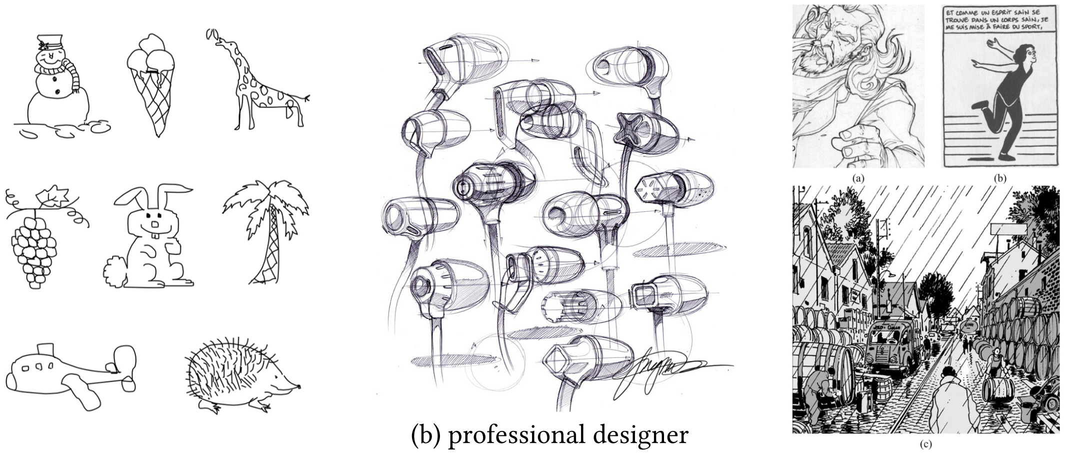
Explicit The style is explained by a series of carefully chosen operations and parameters, like extracting lines, distributing brush strokes or applying texture [Lit97, BNTS07, OH12]. Although these methods are highly customizable and produce a large variety of styles, it is often complicated to find the right parameters to achieve a given style. It can require a long trial and error process to reach the desired result, which can break the creative process of the artist.

Implicit The style is explained by an exemplar and the tool automatically produces an image whose style is similar to the examplar one. These methods are referred to ”style transfer” and vary on the synthesis algorithm (by copying patches or optimizing the features of the final image) as well as on the way the exemplar is provided (taking an image in the wild [RDB18], or producing an exemplar given a specific protocol [FJL+16, BCK+13]). Although it allows to easily define the desired style, the user has little control on the output of the method.

Interpreting existing artistic depictions

Another way to assist artists is to automatically interpret the images they produce. One popular and important medium to interpret is sketches. Indeed, because of its rapidity to execute, sketches are widely used by either novices or professional artists as a preliminary representation in the creative process [Pip07]. It is thus crucial to be able to understand them.

The computer vision community has produced a large amount of work on how to interpret **natural images** and state of the art methods now allow to segment, label and understand almost any picture. However, interpreting an **artistic creation** brings new challenges due to the specificity of such representation: it is often incomplete or erroneous because of the artistic vision it conveys. As demonstrated in Figure 1.3, drawings can exhibit a large variety of styles. Sketches are often even more approximate since there are made very quickly with no mean of high accuracy. This problem is amplified when working with sketches from novice: they usually adopt representations that are not faithful to the reality (e.g. stickmen to represent humans) and not the most informative (orthographic frontal view) [FMK+03a] as shown in Figure 1.3a).


ET COMME UN ESPRIT SAIN SE TROUVE DANS UN CORPS SAIN, JE ME SUIS MISE À FAIRE DU SPORT,
(a)
(b)
(c)

(a) novices drawings

(b) professional designer concept sketches

(c) illustrative drawings

Figure 1.3: Drawings can exhibit a large variety of styles and completeness depending on the skill of the artist and on its purpose: a) novices drawings are schematic and not faithful to the reality (drawings taken from [EHA12]), b) designers draw objects in accurate perspective using construction lines (Drawings by Spencer Nugent, taken from [ADN+17]), c) illustrative drawings from comics can exhibit various styles and abstraction levels (Drawings by Denis Medri, Marjane Satrapi and Jacques Tardi, taken from [GTDS10])

Sketch based interactions

Being able to interpret concept sketches gives ways to artists to explore different possibilities from early stages of the design. For example, SketchSoup allows to explore the design space of shapes from a few sketches [ADN+17]. Sketches can also be used to quickly explore mechanisms for objects with movable parts [SLZ+13] or as an input for generative design [KGC+17]. In 2D animation, sketches allow to intuitively define character animations [TBvdP04] or poses [GCR13, HMC+15]. Finally, sketching gesture are also used to guide image or 3D scene edition, by roughly indicating flow directions or shapes [CGS16, ZHP+19].

Sketch reconstruction

The most common application of sketch interpretation is probably to create 3D objects. Sketch based modeling systems allow the user to create new 3D objects using sketching gesture but usually requires them to have good drawing skills. Often, users need to adapt their technique to the system, either because the tool ask for numerous 3D interactions [IMT99, NISA07a, BBS08] or because it needs specific inputs such as symmetry or ordering relationship [GIZ09]. For trained artists who know how to draw very quickly, these additional interactions can break their creative process. For novices, reaching the required accuracy in the drawing can be difficult. For this reason, systems designed for complete novices are often based on shape retrieval: the goal is then to retrieve a similar shape in a database instead of creating a new one [XXM+13, ERB+12].

Besides the academic research on this topic (please refer to Section 2.2), several industrial software have been released, either on their own or included in well known 3D modelers (e.g. CATIA Natural Sketch [Sys11]). But no system currently exists that allows the user to draw as they would do on paper and obtain a good estimate of the underlying 3D shape.

Sketch models

As for the generation of artistic images, the main challenge is to understand how artists create sketches so that we can model them. As previously, we can distinguish two ways of modeling the understanding of sketches:

Explicitly through the definition of constraints over the lines or over the whole sketch. The solution is then found by optimization to satisfy as much as possible these constraints [CSMS13, JHR+15, XCS+14]. However, defining any constraint on lines requires to be able to first understand what is a line in the drawing, which is usually solved by using vectorized input. Moreover, the possible solutions are quite restricted by the amount of constraints we are able to define and compute.

Implicitly through the use of a database of exemplars. In this case, the system learns to interpret a sketch by seeing a large number of exemplar interpretation, for example pairs of sketches and 3D objects [LPL+18, LYF17]. The possible solutions are then constrained by the space covered by the exemplars.

Scope of this thesis

In this thesis, I explore generating and interpreting artistic content through the presentation of two different tools:

- a sketch based modeling system that interprets one or several sketches as a 3D object,
- a method to generate stylized videos that reproduce the simplified motion of 2D animations

We took inspiration from traditional techniques of artists to guide the design of both tools.

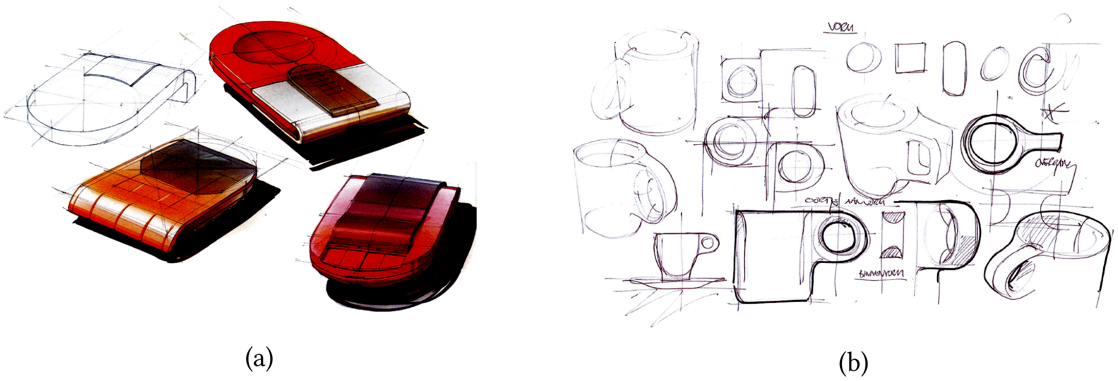
Sketch based modeling

For the first project, we looked at how professional designers use drawing to create new shapes. When drawing only one sketch, designers usually use a three-quarters perspective view, often recognized as the most informative viewpoint [ES11]. In addition, designers often create plates of drawings that contain several sketches of the same objects. In this case, they demonstrate the use of a larger variety of viewpoints as shown in Figure 1.4, and more specifically of orthogonal ones (face, top, side) that give a more precise representation of the shape.

Designers also usually distinguish two types of lines: *construction lines* that help them lay down the correct structure of a shape and guide the drawing of the *lines descriptive of the surfaces.*

These observations guided the design of our sketch based modeling system in different ways.

First, we designed the workflow so that the user can draw a complete shape from one viewpoint and get a 3D reconstruction without having to draw from different views at



(a)

(b)

Figure 1.4: a) When drawing each object from only one view, designers use a three-quarters perspective view. b) To explore the design of an object more precisely, designers create plates of drawings that contains a mix of perspective and orthographic views. Images from [Pip07, ES11]

first. We propose an interactive setup where the user can refine its shape from any number of viewpoints. We also create an interface that allows the user to draw construction lines as well as descriptive lines, although only these last ones are used by the system since they give the most information on the surface and are easier to synthesize.

To model the interpretation of sketches, we chose a data-driven approach and guided the design of the training database with the previously stated observations: the database contains objects drawn in perspective from near to three-quarters views and orthogonal viewpoints (used only when a first sketch has already been drawn). The knowledge about how designers draw is thus mainly incorporated in the conception of the training database that conditions what the system learns to interpret.

More precisely, we use convolutional neural networks (CNNs) to predict a voxel grid from several sketches as well as one normal map per input view. The voxel grid prediction gives a good approximation of the global shape while the addition of normal maps allows for more details on visible parts of the object.

Using deep learning allows the system to handle a large diversity of shapes. The system can easily be retrained with a different database depending on the user's need. It is also very fast to evaluate, which is essential in an interactive setup, and naturally robust to some amount of noise without requiring clean vectorized lines.

More explanations on the system and the voxel prediction can be found in Chapter 2.

Chapter 3 details the addition of normal maps to improve the final quality while Chapter 4 describes the construction of our database.

Video stylization

For the second project, we looked at traditional 2D animations and observed how animators simplify the motion, in addition to the appearance stylization. Our major source of inspiration is ”cut-out” animation, in which the motion is simplified to an extreme: pieces of textured paper are moved in front of the camera, resulting in regions moving rigidly in the 2D plane (following only translation, rotation and scaling). This effect also shows up in other cartoon types such as the famous *South Park* TV show or in Disney multi-plane cell animation. Inspired by these artworks, we decided to simplify the motion such that it will be piecewise rigid: we thus decompose the video into different segments that we make follow a rigid motion at each frame.

We modeled explicitly this problem in several constraint-based optimizations for both finding the right segmentation, the motion associated to each segment and the final motion. More precisely, we first segment the video in a temporally coherent manner to find pieces whose motion could be approximated by a rigid motion. The user gives guidance on the segmentation by scribbling to choose which parts should be separated or in the same rigid segment. This gives us a rigid motion that each pixel should follow at each frame. We then optimize for the final motion so that the resulting trajectories do not deviates too much from the input ones.

All details about this technique are presented in Chapter 5.

Contributions

These projects have led to two publications in international conferences and journals:

- Johanna Delanoy, Mathieu Aubry, Phillip Isola, Alexei A. Efros, Adrien Bousseau. 3D Sketching using Multi-View Deep Volumetric Prediction, in *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 1 (21), 2018
- Johanna Delanoy, Adrien Bousseau, Aaron Hertzmann. Video Motion Stylization by 2D Rigidification, in *Proceedings of the ACM/EG Expressive Symposium*, 2019, to appear

as well as one submission to an international conference (acceptance pending)

- Johanna Delanoy, David Coeurjolly, Jacques-Olivier Lachaud, Adrien Bousseau. Combining Voxel and Normal Predictions for Multi-View 3D Sketching, *Shape Modeling International*, 2019, under review

Chapter 2

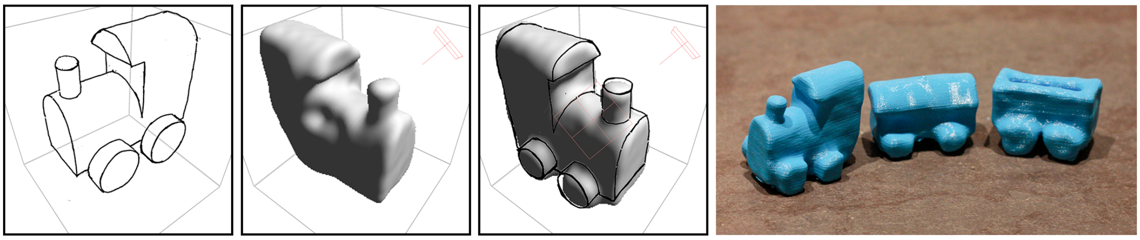
3D sketching using multi-view deep volumetric prediction

This work was done in collaboration with Mathieu Aubry (École des Ponts), Phillip Isola (MIT) and Alexei A. Efros (UC Berkeley). It was presented at *I3d 2018* and published in *Proceedings of the ACM on Computer Graphics and Interactive Techniques* in 2018.

2.1 Introduction

The ambition of sketch-based modeling is to bring the ease and immediacy of sketches to the 3D world to provide *"an environment for rapidly conceptualizing and editing approximate 3D scenes"* [ZHH96]. However, while humans are extremely good at perceiving 3D objects from line drawings, this task remains very challenging for computers. In addition to the ill-posed nature of 3D reconstruction from a 2D input, line drawings lack important shape cues like texture and shading, are often composed of approximate sketchy lines, and even when multiple drawings of a shape are available, their level of inaccuracy prevents the use of geometric algorithms like multi-view stereo. We introduce a data-driven sketch-based modeling system that addresses these challenges by *learning* to predict 3D volumes from one or several freehand bitmap drawings. Our approach builds on the emerging field of generative deep networks, which recently made impressive progress on image [CK17] and shape synthesis [FSG17] but has been little used for interactive creative tasks.

Figure 2.1 illustrates a typical modeling session with our system. The user starts by drawing an object from a 3/4 view, which is the viewpoint preferred by designers to illustrate multiple sides of a shape in a single drawing. Thanks to training on a large collection of 3D shapes, our approach produces a complete volumetric reconstruction of the object, including occluded parts. This initial reconstruction allows the user to rotate the object and inspect it from a different vantage point. The user can then either re-draw



(a) Initial drawing (b) 3D prediction seen from another viewpoint (c) New drawing and updated prediction (d) 3D printed objects

Figure 2.1: Our sketch-based modeling system can process as little as a single perspective drawing (a) to predict a volumetric object (b). Users can refine this prediction and complete it with novel parts by providing additional drawings from other viewpoints (c). This iterative sketching workflow allows quick 3D concept exploration and rapid prototyping (d).

the object from this new viewpoint to correct errors in the reconstruction, or move on to drawing new parts of the object. In both cases, the temporary 3D reconstruction acts as a reference that significantly helps the user create new drawings of the 3D shape. Since all interactions occur in a shared 3D space, this workflow provides us with multiple registered drawings of the object along with their respective calibrated cameras, which form the input to our 3D reconstruction algorithm.

At the core of our system are deep convolutional neural networks (CNNs) that we train to predict occupancy in a voxel grid, given one or several contour drawings as input. These CNNs form a flexible and robust 3D reconstruction engine that can interpret bitmap drawings without requiring complex, hand-crafted optimizations [LS96, XCS$^+$14] nor explicit correspondences between strokes in multiple views [BBS08, RDI10]. However, applying deep learning to sketch-based modeling raises several major new challenges. First, we need a network architecture capable of fusing the information provided by multiple, possibly inconsistent, drawings. Our solution combines a single-view network, which generates an initial reconstruction from one drawing, with an *updater network* that iteratively refines the prediction as additional drawings are provided. This iterative strategy allows us to handle drawings created from an arbitrary number of views, achieving a continuum between single-view [GIZ09] and multi-view [BBS08] sketch-based modeling systems.

Then, we describe how to co-design the training data and the user interface to reduce ambiguity in the prediction. In particular, we restrict viewpoints for the first drawing to avoid depth ambiguity for the single-view network, while we allow greater freedom for the subsequent drawings that are handled by the updater network.

Once trained, our system can generate a coherent multi-view prediction in less than a second, which makes it suited for interactive modeling. One restriction of our current implementation is that the resolution of the voxel grid hinders the recovery of thin structures. We thus target quick 3D design exploration rather than detailed modeling.

In summary, we introduce an interactive sketch-based modeling system capable of reconstructing a 3D shape from one or several freehand bitmap drawings. In addition to the overall system, we make the following technical contributions:

- An iterative *updater network* that predicts coherent 3D volumes from multiple drawings created from different viewpoints.
- A multi-view drawing interface that we co-design with our synthetic data to help users create drawings similar to the ones used for training.

Note that our approach is modular and could adapt to other drawing techniques and shapes than the ones used in this paper.

2.2 Related work

Our work builds on recent progress in deep learning to tackle the long standing problem of sketch-based modeling. We refer the interested reader to recent surveys for extended discussions of these two fields [SSM$^+$16, OSSJ09, CSGC16].

2.2.1 Sketch-based modeling

The problem of creating 3D models from line drawings has been an active research topic in computer graphics for more than two decades [ZHH96, LS96, IMT99]. While sketching is one of the most direct ways for people to represent imaginary 3D objects, recovering 3D information from 2D strokes poses significant challenges since an infinity of 3D shapes can potentially re-project on the same drawing [BT81]. Various approaches have been proposed to tackle the inherent ambiguity of this inverse problem.

Constrained-based approaches assume that the lines in a drawing represent specific shape features, from which geometric constraints can be derived and imposed in an optimization framework. Popular constraints include surface orientation along smooth silhouettes [MM89], orthogonality and parallelism of edges on polyhedral models [LS96], symmetry [CSMS13], and surface developability [JHR+15] among others. However, the assumptions made by these methods often restrict them to specific classes of shapes, or specific drawing techniques such as polyhedral scaffolds [SKSK09], curvature-aligned cross-sections [SBSS12, XCS+14] or cartoon isophotes [XGS15a]. In addition, most of these methods require clean vector drawings as input to facilitate the detection of suitable constraints, as well as to compute the various energy terms that drive the optimization. Unfortunately, converting rough sketches into clean vector drawings is a difficult problem in its own right [FLB16], while methods capable of directly recovering 3D information from noisy drawings are prohibitively expensive [IBB15]. In this work, we bypass all the challenges of defining, detecting and optimizing for multiple geometric constraints by training a deep convolutional neural network (CNN) to automatically predict 3D information from bitmap line drawings.

Interactive approaches reduce ambiguity in 3D reconstruction by leveraging user annotations. Single-image methods allow users to create 3D models from existing imagery by snapping parametric shapes to image contours [CZS+13, SAG+13] or by indicating geometric constraints such as equal length and angle, alignment and symmetry [GIZ09] or depth ordering [SKv+14]. Other methods adopt an incremental workflow where users progressively build complex shapes by drawing, modifying and combining simple, easy to reconstruct 3D parts. Existing systems differ in the type of assumptions they make to reconstruct intermediate shapes from user strokes, such as smooth shapes inflated from silhouettes [IMT99, NISA07b], symmetric or multi-view pairs of 3D curves related by epipolar constraints [OK12, BBS08], curves lying on pre-defined planes or existing surfaces [BBS08, ZLDM16], visual hulls carved from orthogonal viewpoints [RDI10]. The main drawback of such methods is that users have to mentally decompose the shape they wish to obtain, and construct it by following a carefully ordered series of sketching operations, often performed from multiple viewpoints. In contrast, while our system supports incremental modeling, our CNN-based reconstruction engine does not rely on restrictive assumptions on the drawn shapes and allows users to draw a complete object from one viewpoint before visualizing and refining it from other viewpoints.

Data-driven approaches exploit large collections of 3D objects to build priors on the shapes that users may draw. Early work focused on retrieving complete objects from a database [FMK+03b, ERB+12], which was later extended to part-based retrieval and assembly [LF08, XXM+13] and to parameter estimation of pre-defined procedural shapes [NGDGA+16, HKYM16]. While our approach also learns shape features from object databases, we do not require these objects to be expressible by a known parametric model, nor be aligned and co-segmented into reconfigurable parts. Instead, our deep network learns to generate shapes directly from pairs of line drawings and voxel grids, which allows us to train our system using both existing 3D model databases and procedurally-generated shapes. Our approach is also related to the seminal work of Lipson and Shpitalni [LS00], who used a database of random polyhedrons to learn geometric correlations between 2D lines in a drawing and their 3D counterpart. The considered correlations include the angles between pairs and triplets of lines, as well as length ratios. These priors are then used to evaluate the quality of a 3D reconstruction in a stochastic optimization. In a similar spirit, Cole et al. [CIF+12] generate a large number of abstract blobs to serve as exemplars for a patch-based synthesis algorithm that converts line drawings into normal maps. While we build on these initial attempts, deep learning alleviates the need for custom feature extraction and optimization and allows us to handle a wider diversity of shapes. In addition, we integrate our 3D reconstruction engine in an interactive system capable of fusing information from multiple sketches drawn from different viewpoints.

2.2.2 Deep learning

Our work is motivated by the recent success of deep convolutional neural networks in solving difficult computer vision problems such as image classification [KSH12], semantic segmentation [LSD15], depth and normal prediction [EF15, WFG15]. In particular, our single-view volumetric reconstruction network follows a similar encoder-decoder architecture as depth prediction networks, although we also propose a multi-view extension that iteratively refines the prediction as new sketches are drawn by the user. This extension is inspired by iterative networks that implement a feedback loop to impose structural constraints on a prediction, for instance to refine hand [OWL15] and human pose [CAFM16].

Our architecture also shares similarities with deep networks tailored to multi-view 3D reconstruction. Choy et al. [CXG+16] train a recurrent neural network (RNN) to predict a voxel reconstruction of an object from multiple uncalibrated photographs. Similarly, our iterative updater network can be seen as a recurrent network that is unrolled to simplify training. In addition, our modeling interface provides us with calibrated cameras by construction, since we know from which viewpoint each drawing is created. Unrolling the network allows us to apply the camera transformations explicitly as we iterate over each viewpoint. Ji et al. [JGZ+17] describe a multi-view reconstruction network that fuses two aligned voxel grids, each being filled with the color rays originating from the pixels of two calibrated input views. Their method extends to more than two views by averaging the predictions given by multiple pairs of views. Our updater network follows a similar strategy of implicitly encoding the camera orientation in the voxel grid. However, we iterate our updater over all drawings, one at a time, rather than combining multiple pairwise predictions at once. This design choice makes our method more sensitive to the order in which the drawings are created.

While CNNs have been mostly applied to photographs, they have also demonstrated impressive performances on tasks similar to ours, such as sketch cleanup [SSISI16], sketch colorization [SLF+17], sketch-based retrieval [WKL15, SBHH16], and sketch-based modeling of parametric shapes [NGDGA+16, HKYM16, HGY17]. CNNs have also recently achieved promising results in *synthesizing* images [YYSL16, NDY+16, PYY+17, CK17] and even 3D models [WSK+15, DSTB16, WZX+16, LXC+17, FSG17] from low-dimensional feature vectors and attributes. We pursue this trend by training a deep network to generate voxelized objects from line drawings, offering precise user control on the shape being generated.

Two recent methods with similar goals have been developed concurrently to ours. First, Liu et al. [LYF17] combine a voxel sculpting interface with a generative network to project the coarse voxel shapes modeled by the user onto a manifold of realistic shapes. We see our sketch-based interface as an alternative to voxel-sculpting. Second, Lun et al. [LGK+17] propose a method to reconstruct a 3D object from sketches drawn from one to three orthographic views. We share several ideas with this latter work, such as training with synthetic drawings and predicting 3D shapes from multiple views. On the one hand, Lun et al. achieve finer reconstructions than ours by extracting a 3D surface from multiple depth maps rather than from a voxel grid. On the other hand, they train separate networks to process different combinations of front/side/top views, while our updater network allows us to fuse information from any of the 13 viewpoints available

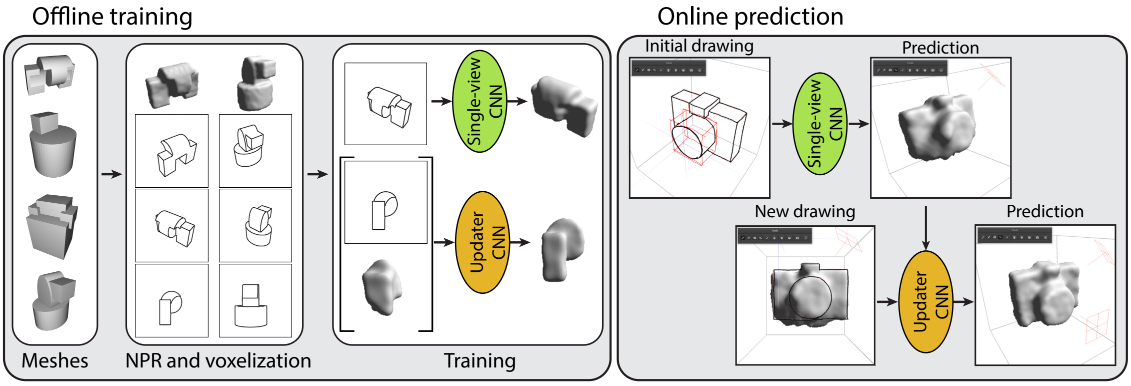


Figure 2.2: Overview of our method. Left: We train our system with a large collection of 3D models, from which we generate voxel grids and synthetic drawings. We train a single-view CNN to predict an initial reconstruction from a single drawing, as well as an updater CNN to refine a reconstruction given a new drawing. Right: The single-view CNN allows users to obtain a complete 3D shape from a single drawing. Users can refine this initial result by drawing the shape from additional viewpoints. The updater CNN combines all the available drawings to generate the final output.

in our interface. In addition, we integrated our approach in an interactive system to demonstrate the novel workflow it enables.

2.3 Overview

Figure 2.2 provides an overview of our system and the underlying convolutional neural networks. The left part of the figure illustrates the offline training of the deep neural networks. Given a dataset of 3D models, we first generate a voxel representation of each object, along with a series of line drawings rendered from different viewpoints. Our single-view CNN takes a drawing as input and generates a voxel grid with probabilistic occupancy. Our updater CNN also takes a drawing as input, and complements it with an initial 3D reconstruction provided by the single view network. Note that we transform this reconstruction according to the camera matrix of the input drawing, so that the updater CNN does not have to learn the mapping between the 3D volume and a given viewpoint. The updater network fuses the information from these two inputs to output a new 3D reconstruction. In practice, we repeatedly loop the updater over all available drawings of a shape to converge towards a multi-view coherent solution.

The right part of Figure 2.2 illustrates our online modeling workflow. The main motivation of our approach is to provide a workflow that seamlessly combines 2D sketching and 3D visualization. At the beginning of a modeling session, our interface displays an empty 3D space seen from a 3/4 view. We additionally display perspective guidance to help users draw with the same perspective as the one used to generate the training data, as detailed in Section 2.6. Once an initial drawing is completed, the user can invoke our single-view CNN to obtain its 3D reconstruction, which she can visualize from any viewpoint. The user can then refine the shape by re-drawing it from a new viewpoint, using the current reconstruction as a reference. We feed each new drawing to the updater network to generate an improved 3D reconstruction.

2.4 Volumetric prediction from line drawings

The key enabler of our modeling system is a deep convolutional network that we train to predict voxelized objects from line drawings. We first present our single-view network that takes as input one drawing to generate an initial 3D reconstruction. We then introduce our *updater network* that iteratively fuses multi-view information by taking as input a drawing and an existing volumetric prediction. We illustrate our network in Figure 2.3 and provide a detailed description in Appendix A. We discuss and compare our design choices against alternative solutions in Section 2.7.

2.4.1 Single view prediction

Our single-view network follows an encoder-decoder architecture typical of image generation tasks such as depth prediction [EF15], colorization [SLF+17], novel view synthesis [PYY+17]. The encoder passes the input image through a series of convolutions of stride 2 and rectified linear units to progressively reduce spatial resolution while increasing feature dimensionality, effectively extracting a compact representation of the image content. The decoder passes this representation through a series of deconvolutions of stride 2 and rectified linear units to progressively generate a new visualization of the image content, in our case in the form of a voxel grid.

Following [RFB15], we also include skip connections between the encoder and decoder layers of equal resolution. These skip connections allow local information to bypass the encoder bottleneck, providing the decoder with multi-scale features that capture

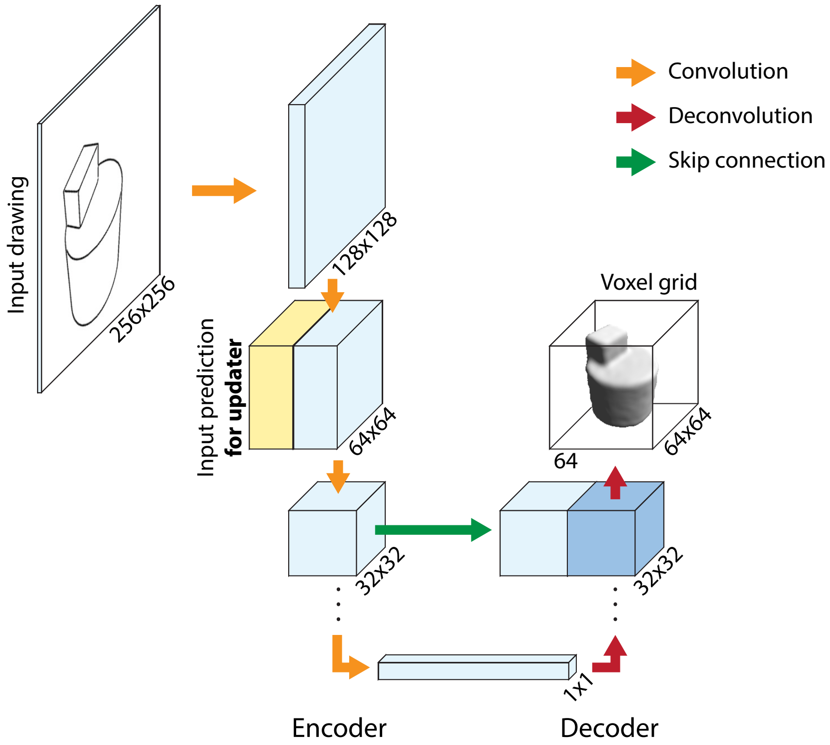


Figure 2.3: Our network follows a so-called "U-net" encoder-decoder architecture. The input drawing is processed by a series of convolution and rectified linear units to extract high-dimensional features at low spatial resolution. These features are then processed by deconvolutions and rectified linear units to generate the multi-channel image that represents our voxel grid. Skip connections, shown in green, concatenate the output of convolutional layers to the output of deconvolutional layers of the same resolution. These connections allow high-resolution features to bypass the encoder bottleneck, allowing the network to exploit multi-scale information for decoding. The updater network also takes an existing prediction as input, shown in yellow.

both global context and fine image details. Isola et al. [IZZE17] have demonstrated the effectiveness of a similar "U-net" architecture for image-to-image translation tasks.

The task of our network is to classify each voxel as occupied or empty. We model the voxel grid as a multi-channel image, where each channel corresponds to one depth slice. Given this representation, our network can be seen as an extension of existing depth prediction networks [EF15], where we not only predict the depth of the visible surface but also all occluded voxels along the viewing ray corresponding to each pixel of the

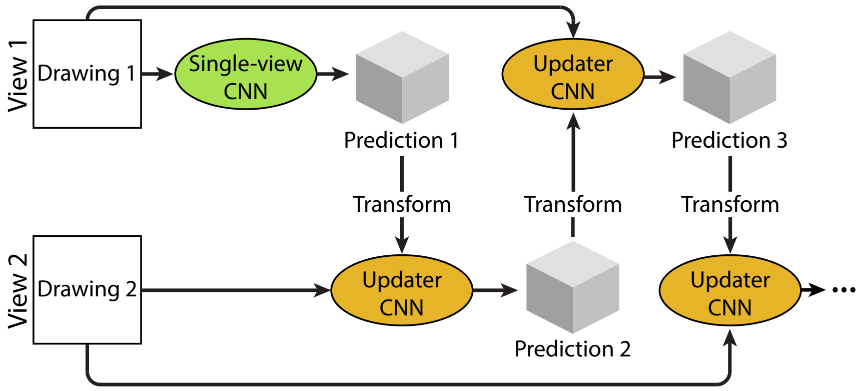


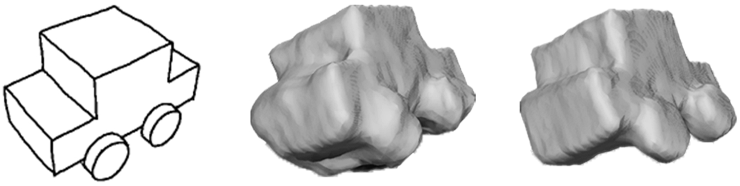
Figure 2.4: We apply the updater network iteratively, alternating between views to converge towards a multi-view coherent solution. Here we illustrate a few iterations between two views, although we loop over more views when available.

input drawing. Since our modeling interface employs a perspective camera model, the voxel grid associated to a drawing actually forms a pyramid in 3D space. While we considered using an orthographic camera model for simplicity, our early experiments suggest that perspective cues significantly help the network to predict depth for regular shapes.

2.4.2 Multi-view prediction

Our updater network adopts a similar architecture as the one described above, except that it also takes as input an existing volumetric prediction and uses the input drawing to refine it. In practice, we concatenate the existing prediction with the output of the second convolution layer, as illustrated in Figure 2.3 (yellow block). Note that we do not threshold the probabilities of occupancy in the existing prediction, which allows the updater network to account for the uncertainty of each voxel.

Iterative update. The updater network processes one drawing at a time, which allows us to handle an unbounded number of views. However, each update may modify the prediction in a way that is not coherent with the other views. We found that we can achieve multi-view consistency by iteratively applying the updater network until convergence, akin to a coordinate descent optimization. Figure 2.4 illustrates this process with two views: the first drawing is given as input to the single-view network to generate a first



Input sketch Single-view prediction 5 updates

Figure 2.5: The updater network can refine the prediction even when only one drawing is available.

prediction. This prediction is then given to the updater network along with the second drawing to produce a refined solution. The resulting voxel grid can now be processed again by the updater, this time taking the first drawing as input. This process generalizes to more views by looping the updater over all drawings in sequence. In practice, we used 5 iterations for all results in the paper. We evaluate the convergence of this iterative scheme in Section 2.7.

Resampling the voxel grid. As mentioned in Section 2.4.1, we designed our networks to process and generate voxel grids that are expressed in the coordinate system of the input drawing. When dealing with multiple drawings, the prediction obtained with any drawing needs to be transformed and resampled to be passed through the updater network with another drawing. In practice, we store the prediction in a reference voxel grid in world coordinates, and transform this grid to and from the coordinate system of each drawing on which we run the updater network.

Single-view refinement. While we designed the updater network to fuse information between multiple views, we found that it is also able to refine a single-view prediction when used as a feedback loop on a single drawing, as shown in Figure 2.5. This observation may seem counter-intuitive, since the updater does not have more information than the single-view network in that configuration. We hypothesize that iterating the updater on a single drawing emulates a deeper network with higher capacity. Note also that a similar iterative refinement has been demonstrated in the context of pose estimation [CAFM16, OWL15].

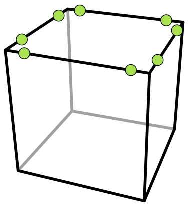
2.5 Training

Details about the design of the training database are given in Chapter 4. For the results shown in this chapter, we use procedurally generated objects rendered with image-space contours. We detail here the viewpoints used to render contours and the training procedure of our system.

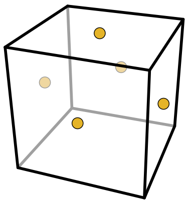
2.5.1 Viewpoints

Viewpoint is a major source of ambiguity for line drawing interpretation. We now describe our strategies to significantly reduce ambiguity for the single-view network by restricting camera orientation and position. We relax these restrictions for the updater network since it can handle more ambiguity thanks to the existing prediction it also takes as input.

Camera orientation. Representing a 3D object with a single drawing necessarily induces ambiguity. The design literature [ES11] as well as other sketching systems [BBS08, SBSS12, XCS+14] recommend the use of "informative" perspective viewpoints that reduce ambiguity by showing the 3D object with minimal foreshortening on all sides. We follow this practice to train our single-view network. We render each object from eight viewpoints positioned near the top corners of its bounding box, as shown in inset.



In addition, designers frequently adopt so-called "accidental" viewpoints when a representing shape with several drawings, such as the common front, side and top views. We include these viewpoints in the training set of our updater network since we found them useful to refine axis-aligned shapes. However, we do not use these viewpoints with the single-view network because they often yield significant occlusions, which make them very challenging to interpret in the absence of additional information. The inset shows the additional viewpoints available to the updater network.



Camera position. Line drawings also have an inherent depth ambiguity: the same drawing can represent a small object close to the camera, or a big object far from the

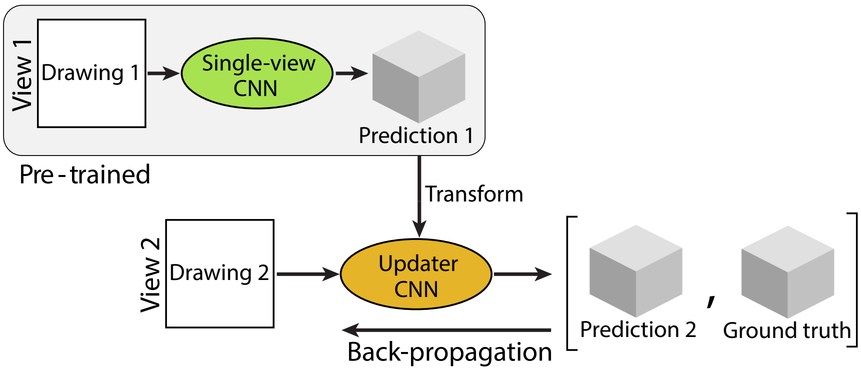


Figure 2.6: We first train our single-view network on ground truth data, then use its predictions as training data for the updater network.

camera. We reduce such ambiguity for the single-view network by positioning the 3D object at a constant distance to the camera. In addition, we achieve invariance to 2D translations in the image plane by displacing the camera by a random vector perpendicular to the view direction.

However, a 2D translation in one view potentially corresponds to a translation in depth in another view, which prevents us imposing a constant distance to the camera for the updater network. We thus train the updater network with random 3D displacements of the camera. We found that the updater network succeeds in exploiting the existing prediction to position the object in depth.

2.5.2 Training procedure

We train our single view network by providing a line drawing as input and a ground truth voxel grid as output. However, training our updater network is more involved since we also need to provide an existing prediction as input. Given a drawing and its associated 3D model, we obtain an initial prediction by running the single-view network on another viewpoint of the same object. Figure 2.6 illustrates this process. We thus need to train the single-view network before training the updater.

We trained our system using the Adam solver [KB14], using batch normalization [IS15] to accelerate training. We fixed Adam's parameters to $\beta_1 = 0.5$, $\beta_2 = 0.999$, $\epsilon = 1e - 8$. We fixed the learning rate to 0.0002 and trained the networks for $1,000,000$ iterations. Training the complete system took around a week on a NVidia TitanX GPU.

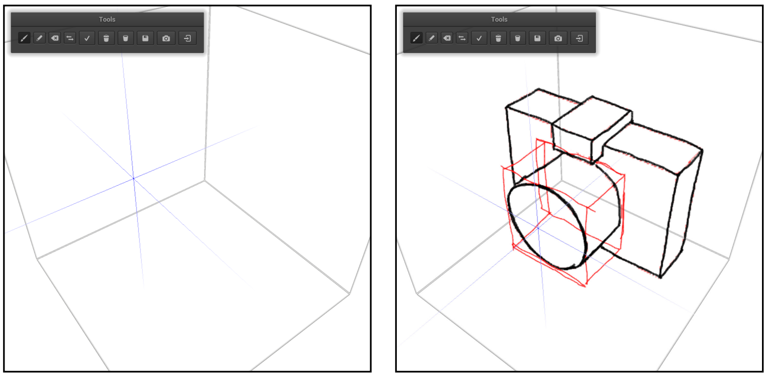


Figure 2.7: Screenshots of our user interface. We display axis-aligned lines around the cursor to guide perspective drawing (left, shown in blue). We also allow users to draw construction lines (right, shown in red). Only the black lines are processed by our 3D reconstruction engine.

2.6 User interface

Figure 2.7 shows the interactive interface that we built around our deep 3D reconstruction engine. We designed this interface to reproduce traditional pen-on-paper freehand drawing. However, we introduced several key features to guide users in producing drawings that match the characteristics of our training data in terms of viewpoints and perspective.

Similarly to the seminal Teddy system [IMT99], the working space serves both as a canvas to draw a shape and as a 3D viewer to visualize the reconstruction from different viewpoints. While we allow free viewpoint rotations for visualization, we restrict the drawing viewpoints to the ones used for training. In particular, we impose a 3/4 view for the first drawing, and snap the camera to one of the 13 viewpoints available to the updater for subsequent drawings.

The menu in the top left allows users to switch from 2D drawing to 3D navigation and also provides basic drawing tools (pen and eraser). In addition, we provide a "construction line" mode to draw scaffolds [SKSK09] and other guidance that will not be sent to the network (shown in red in our interface). We found such lines especially useful to lay down the main structure of the object before drawing precise contours (shown in black). We further facilitate perspective drawing by displaying three orthogonal vanishing lines centered on the pen cursor (shown in blue) and by delineating the working space with a wireframe cube.

For each voxel, our networks estimate the probability that it is occupied. We render the shape by ray-casting the 0.5 iso-surface of this volume, using the volumetric gradient to compute normals for shading. We also export the shape as a triangle mesh, which we obtain by apply a marching cube [LC87] followed by a bilateral filter to remove aliasing [JDD03].

2.7 Results and evaluation

We now evaluate the expressivity and robustness of our method and compare it to alternative approaches. We use the dataset of abstract procedural shapes described in next chapter for these comparisons. All results were obtained with a voxel grid of resolution 64^3.

In all cases we evaluate the quality of a volumetric reconstruction against ground-truth using the intersection-over-union (IoU) metric, which computes the ratio between the intersection and the union of the two shapes [HTM17, RUBG17]. The main advantage of this metric over the classification accuracy is that it ignores the many correctly-classified empty voxels far away from the shapes.

2.7.1 Creative modeling by experts

Figure 2.8 presents several 3D scenes modeled with our system by two expert users. These results were created with the version trained on abstract procedural shapes, which succeeds in interpreting these drawings of diverse man-made shapes. In particular, the CNNs manage to segment the foreground object from its background, combine information from different drawings to reconstruct occluded parts, create holes and concavities such as on the armchairs and on the last wagon of the train. Figure 2.9 shows the more challenging case of a house with a slanted roof, which is well reconstructed even though the networks were only trained with shapes made of axis-aligned cuboids and cylinders.

We provide screen captures of a few modeling sessions in the video accompanying the publication[1], showing how users iterate between 2D sketching and 3D navigation within a single workspace. In particular, users can draw a complete shape from one viewpoint before rotating the 3D model to continue working on it from another viewpoint. This

[1] https://youtu.be/DGIYzmlm2pQ

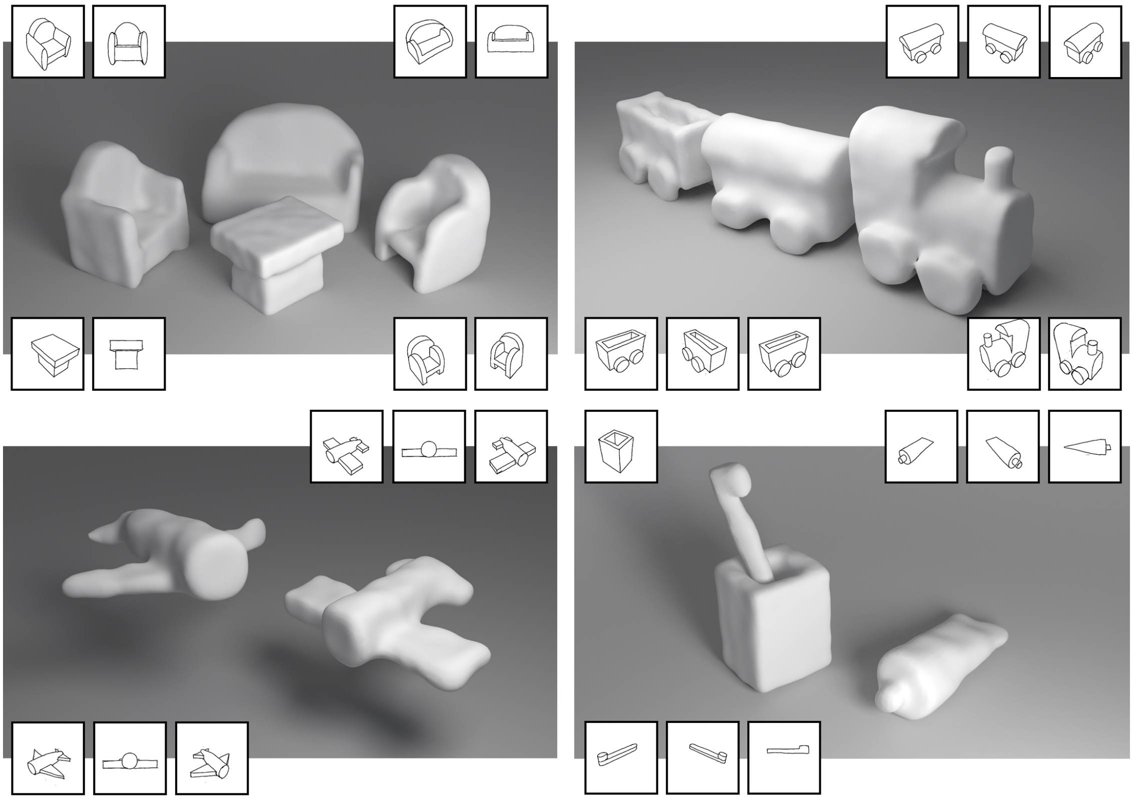


Figure 2.8: 3D scenes modeled using our system. Each object was modeled with two to three hand drawings, shown in insets.

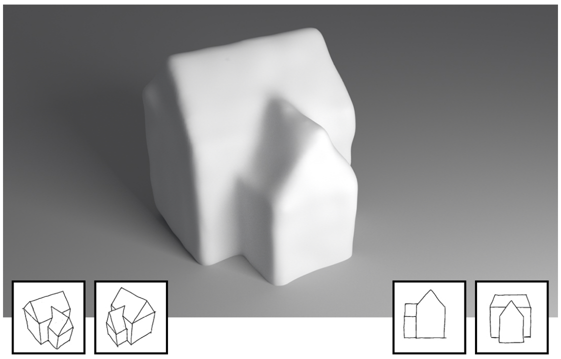
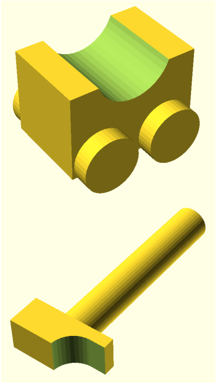


Figure 2.9: Our system manages to reconstruct the slanted roof of this house, even though it was only trained on shapes composed from axis-aligned cuboids and cylinders.

workflow contrasts with the one of existing sketching systems that require users to decompose the object in simple parts [NGDGA+16] or to provide multiple drawings of the shape before obtaining its reconstruction [RDI10]. The video also shows animated 3D visualizations of the objects.

2.7.2 Evaluation by novice users



While we designed our system for artists who know how to draw in perspective, we also conducted a small study to evaluate whether our interface is usable by novices. We recruited six participants with limited drawing and 3D modeling skills (average score of 2.8 and 2.3 respectively on a 5-point Likert scale from 1 = *poor* to 5 = *good*). All participants followed a 15 minutes tutorial to learn how to draw a cube and a cylinder within our interface. We then asked each participant to model one of the two objects shown in inset, which we designed to be expressible by our shape grammar. Figure 2.10 shows the drawings and 3D models they created.

Overall, participants quickly managed to use our system (average score of 5.5 on a 7-point Likert scale from 1 = *hard to use* to 7 = *easy to use*). However, many participants were disappointed by the lack of details of the reconstruction and gave an average score of 3.8 when asked if the 3D model corresponds well to their drawings (1 = *not at all*, 7 = *very well*). The best results were obtained by participants who planned their drawings ahead to best represent the shape centered on screen (P1 and P6). In contrast, two participants did not obtain a complete shape because they drew the object too small to capture details (P2) or too big to fit in the drawing area (P5). This observation suggests the need for additional guidance to help novices compose a well-proportioned perspective drawing.

All participants judged the on-cursor vanishing lines helpful to draw in perspective (6.6 on average on a 7-point Likert scale from 1 = *not helpful* to 7 = *very helpful*). P3 commented *"Sometimes it seems to me that the guides point to wrong directions, but that is just my sense of perspective that is wrong!"*. All the participants followed the vanishing lines to draw cuboid shapes. However, several participants commented that they would have liked guidance to draw 3D cylinders. In particular, P2 drew very approximate cylinders to represent the wheels of his car, which our system failed to interpret properly.

Finally, even though P1 and P6 created many drawings, several are redundant and did not help our system improve its prediction. We believe that users would interact more

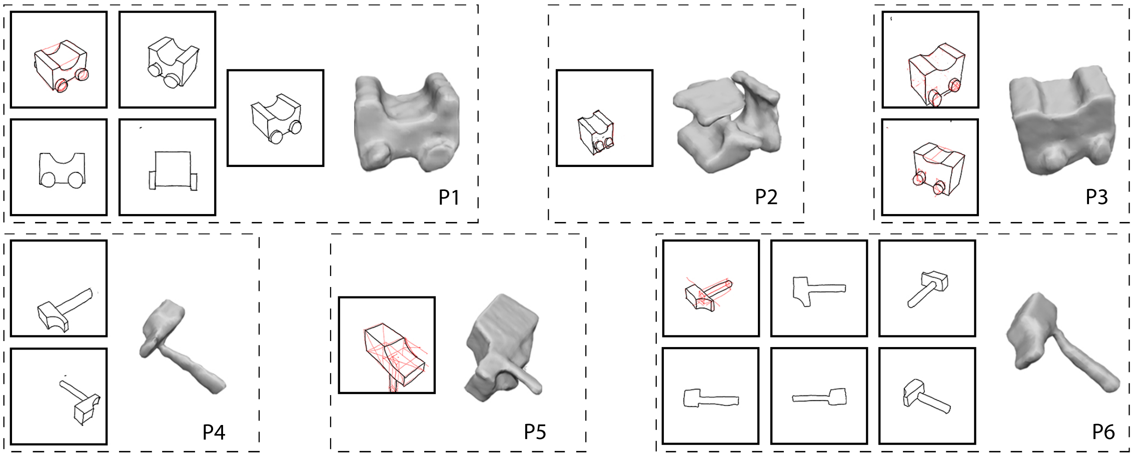


Figure 2.10: Drawings and 3D objects created by our six novice participants. P1 and P6 obtained the best results by drawing the object in the center of the canvas, with proper perspective. In contrast, P2 drew the object too small and with too approximate perspective to be reconstructed by our system, while P5 left too little room for the handle of the hammer.

effectively with our system if we could indicate which regions of the shape is under-constrained. Recent work on uncertainty quantification in deep networks form a promising direction to tackle this challenge [KG17].

2.7.3 Convergence of the updater

In what follows, we count one iteration each time the updater network visits all views in sequence. Figure 2.11(left) plots the L^2 distance between successive iterations, averaged over 50 abstract shapes rendered from two, three and four random views. While we have no formal proof of convergence, this experiment shows that the algorithm quickly stabilizes to a unique solution. However, Figure 2.11(right) shows that the accuracy decreases slightly with iterations. We suspect that this loss of accuracy is due to the fact that the updater is only trained on the output of the single-view network, not on its own output. However, training the updater recursively would be more involved. We found that 5 iterations provide a good trade-off between multi-view coherence and accuracy.

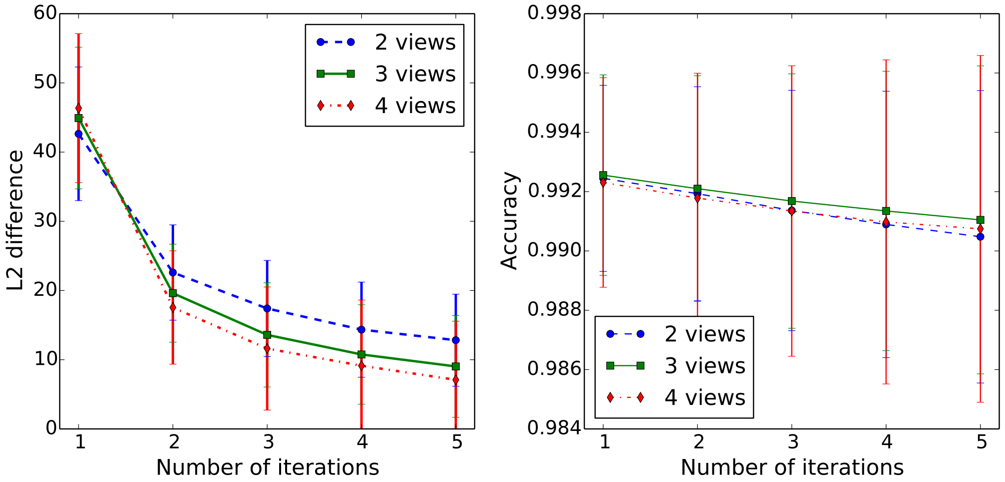


Figure 2.11: Left: Difference of prediction between successive iterations of the updater network, showing that the network quickly converges towards a stable solution. Right: The accuracy decreases slightly during the iterations. 5 iterations offer a good trade-off between multi-view coherence and accuracy.

2.7.4 Comparisons

To the best of our knowledge, our method is the first that can automatically reconstruct a 3D model from a set of multiple perspective bitmap drawings. As a baseline, we compare our approach with a silhouette carving algorithm [MA83]. We implemented two versions of silhouette carving for this comparison. The first version takes as input the same drawings as the ones provided to our method, which necessarily includes a 3/4 view for the first drawing to be fed to the single-view network, and different random views for the other drawings. The second version only takes drawings from orthogonal views, which is the most informative setup for silhouette carving. As shown in Figure 2.12 (right) (left), our approach outperforms silhouette carving in both conditions. In particular, our method achieves a high IoU ratio with as little as one view. Figure 2.12 provides a visual comparison between our reconstructions and the ones by silhouette carving. Our approach is especially beneficial in the presence of concavities.

Figure 2.13 evaluates our network architecture against several alternative designs. We perform this evaluation on the single-view network since any improvement made on it would directly benefit the updater. A first important design choice to evaluate is the

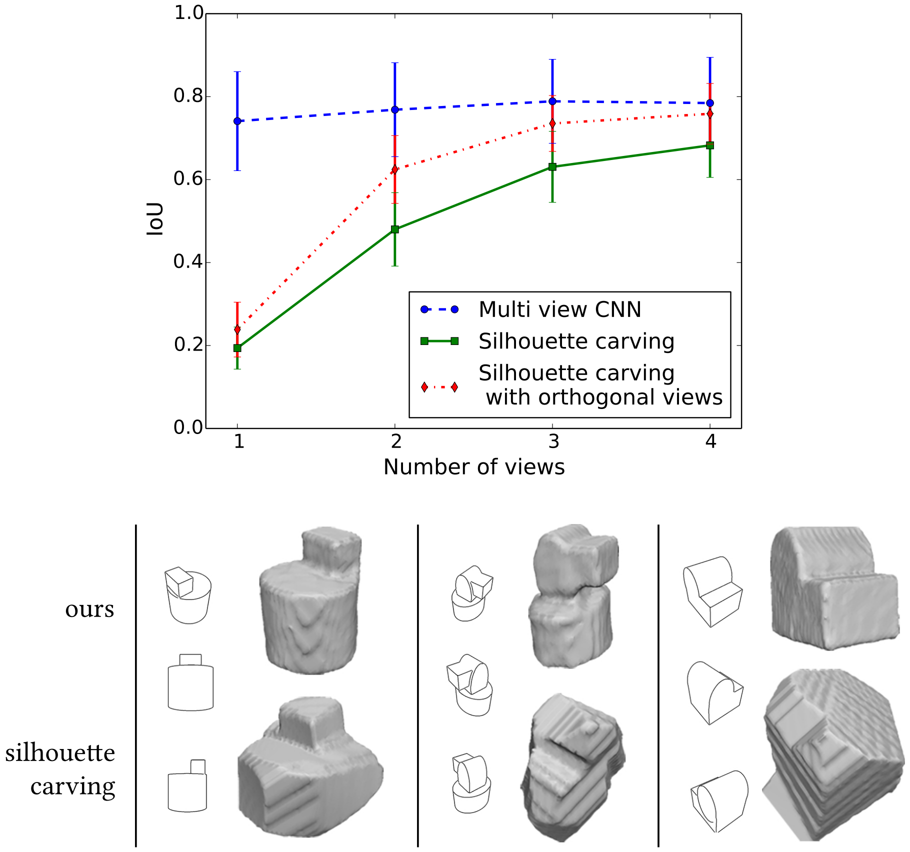


Figure 2.12: Top: Comparison between our method (blue) and silhouette carving (green and red). The strength of our approach is that it achieves high accuracy from only one view, and remains competitive with silhouette carving with four views. Bottom: Reconstructed objects using our method (top row) and silhouette carving (bottom row) with 3 random views. Our method can handle concavities that cannot be recovered by carving.

choice of the volumetric representation. While we chose a binary representation of the volume, we also considered a signed distance function. However, our experiments reveal that this alternative representation reduces quality slightly, producing smoother predictions than ours. We also compare our U-net architecture with the multi-scale depth prediction network proposed by Eigen and Fergus [EF15], which we modified to output a multi-channel image. This network follows a similar encoder-decoder strategy as ours but does not include as many skip-connections between multi-scale layers, which also reduces the quality of the prediction.

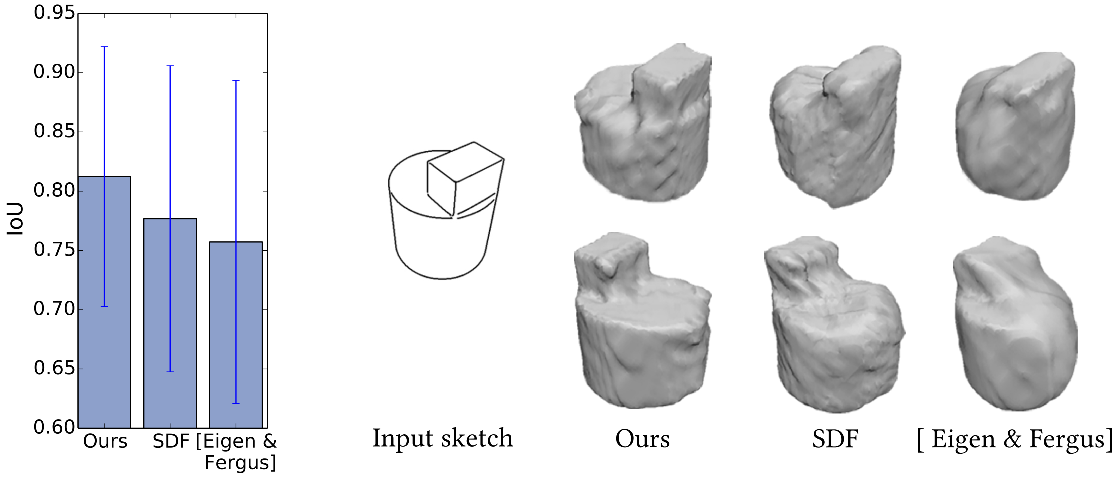


Figure 2.13: We compare our single-view network with the one of Eigen and Fergus [EF15] and with a network trained to predict a signed-distance function rather than a binary voxel grid. Our design outperforms these two alternatives.

2.7.5 Limitations

Figure 2.14 shows drawings with thin structures that are challenging to reconstruct for our current implementation based on a 64^3 voxel grid. High-resolution volumetric representations is an active topic in deep learning [HTM17, RUBG17, FSG17] and we hope to benefit from progress in that field in the near future. An alternative approach is to predict multi-view depth maps, as proposed by Lun et al. [LGK$^+$17], although these depth maps need to be registered and fused by an optimization method to produce the final 3D surface Nevertheless, we present in Chapter 3 an approach to combine voxels and normal maps in order to recover sharp surface features.

Our deep networks also have difficulty interpreting drawings with many occlusions, as shown in Figure 2.15. Fortunately, designers tend to avoid viewpoints with many occlusions since they are not the most informative. Nevertheless, occlusions are inevitable on objects composed of many parts, and we observed that the quality of the reconstruction can reduce as users add more details to their drawings. A simple solution to this limitation would consist in letting the user freeze the reconstruction before adding novel parts. This feature could be implemented by copying the reconstruction in a temporary buffer, and flagging all the lines as construction lines to be ignored by the system. Users

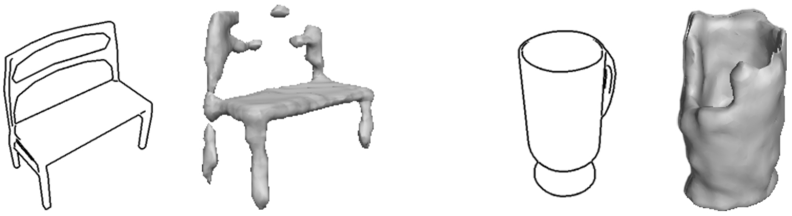


Figure 2.14: Thin structures are challenging to capture by the 64^3 voxel grid.

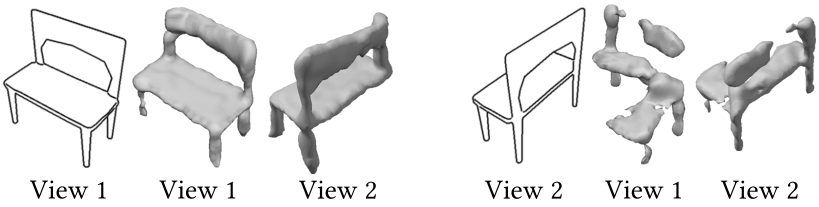


Figure 2.15: The single-view network performs best on informative viewpoints that minimize occlusions (left). Drawing the same shape from a viewpoint with significant occlusions results in an erroneous prediction (right).

could then proceed with drawing new parts which would be interpreted as a new object, and we could display the existing reconstruction and the new parts together by taking the union of their volumes.

2.7.6 Performances

We implemented our system using the Caffe library for deep learning [JSD+14] and OpenGL for real-time rendering in the user interface. Table 2.1 provides timings at test time for an increasing number of views, measured on a desktop computer with an NVidia TitanX GPU, and on a MacBook Pro laptop using only the CPU. Our 3D reconstruction engine scales linearly with the number of views and outputs a prediction in less than a second using GPU and within a few seconds using CPU, on a 64^3 voxel grid with 5 iterations of the updater. Our single-view and updater networks occupy 775MB of memory together.

Table 2.1: Our method scales linearly with the number of input drawings, generating the prediction in less than a second for a 64^3 voxel grid on a modern GPU.

	1 view	2 views	3 views	4 views
Desktop GPU (ms)	140	210	280	350
Laptop CPU (s)	1	1.5	2.2	2.9

2.8 Conclusion

In this chapter, we explored the use of deep learning for sketch based modeling. We trained convolutional neural networks to predict voxel grids from bitmap sketches. Our system, composed of a single-view network and an updater network, allows to get a 3D volume from one sketch and fuse information from multiple drawings. We integrated these networks into an interactive application where the user can draw a shape from one viewpoint, rotate around the predicted shape, and refine it from new viewpoints.

Chapter 3

Combining voxel and normal predictions for multi-view 3D sketching

This work was done in collaboration with David Coeurjolly (Université de Lyon - CNRS) and Jacques-Olivier Lachaud (Université de Savoie). It is under review to be presented at *Shape Modeling International 2019*.

3.1 Introduction

A challenge when using generative networks for sketch-based modeling is the choice of a geometric representation that can both represent the important features of the shape while also being compatible with convolutional neural networks. In the previous chapter, we used voxel grids that form a natural 3D extension to images. However, the memory consumption of voxel grids limits their resolution, resulting in smooth surfaces that lack details. Alternatively, several methods adopt image-based representations, predicting depth and normal maps from one or several drawings [LGK+17, LPL+18, SDY+18]. While these maps can represent finer details than voxel grids, each map only shows part of the surface, and multiple maps from different viewpoints need to be fused to produce a closed object.

Motivated by the complementary strengths of voxel grids and normal maps, we propose to combine both representations within the same system. We complement our volumetric approach with a normal prediction network similar to the one used by Su et al. [SDY+18], which we use to obtain a normal map for each input sketch. The voxel grid thus provides us with a complete, closed surface, while the normal maps allow us to recover details in the parts seen from the sketches.

Our originality is to not only use the voxel grid as a preliminary prediction to be shown to the user, but also as a support for normal map fusion. To do so, we first locate the voxels delineating the object's boundary, and re-project the normal maps on the resulting surface to obtain a distribution of candidate normals for each surface element. We then solve for the smoothest normal field that best agrees with these observations [CFGL16]. Finally, we optimize the surface elements to best align with this normal field [CGL17]. We evaluate our approach on the dataset presented in Chapter 4, on which we recover smoother surfaces with sharper discontinuities.

3.2 Related work

As stated in previous chapter, reconstructing 3D shapes from line drawings has a long history in computer vision and computer graphics. Recent work exploit deep neural networks to predict 3D information from as little as a single bitmap line drawing. However, convolutional neural networks have been originally developed to work on images, and several alternative solutions have been proposed to adapt such architectures to produce 3D shapes.

A first family of methods focuses on parametric shapes such as buildings [NGDGA+16], trees [HKYM16], and faces [HGY17], and train deep networks to regress their parameters. While these methods produce 3D shapes of very high quality, extending them to new classes of objects require designing novel parametric models by hand.

A second family of methods target arbitrary shapes and rely on encoder-decoder networks to convert the input drawing into 3D representations. The system presented in Chapter 2 falls into this family. However, it is limited to voxel grids of resolution 64^3, which is too little to accurately capture sharp features. Alternatively, Su et al. [SDY+18] and Li et al. [LPL+18] propose encoder-decoder networks to predict normal and depth maps respectively. While these maps only represent the geometry visible in the input drawing, Li et al. allow users to draw the object from several viewpoints and fuse the resulting depth maps to obtain a complete object. A similar image-based representation has been proposed by Lun et al. [LGK+17], who designed a deep network to predict depth maps from 16 viewpoints, given one to three drawings as input. In both cases, fusing the multiple depth maps requires careful point set registration and optimization to compensate for misalignment. Our approach combines the strength of both voxel-based and image-based representations. On the one hand, per-sketch normal maps provide high-resolution details about the shape, while on the other hand, the voxel grid provides an estimate of the complete shape as well as a support surface for normal fusion.

Line drawing interpretation is related to the problem of 3D reconstruction from photographs, for which numerous deep-learning solutions have been proposed by the computer vision community. While many approaches rely on voxel-based [CXG+16, JGZ+17] and image-based [TDB16] representations as discussed above, other representations have been proposed to achieve finer reconstructions. Octrees have long been used to efficiently represent volumetric data, although their implementation in convolutional networks requires the definition of custom operations, such as convolutions on hash tables [TDB17] or cropping of octants [HTM17]. Point sets have also been considered as an alternative to voxel-based or image-based representations [FSG17], and can be converted to surfaces in a post-process as done for depth map fusion. More recently, several methods attempted to directly predict surfaces. Pixel2Mesh [WZL+18] relies on graph convolutional networks [BBL+17] to predict deformations of a template mesh. However, this approach is limited to shapes that share the same topology as the template, an ellipsoid in their experiments. In contrast, Groueix et al. [GFK+18] can handle arbitrary topology by predicting multiple surface patches that cover the shape. Since these patches do not form a single, closed surface, their approach can also be used to generate a dense point set from which a surface can be computed as a post-process. In contrast to the above approaches, we chose to combine voxel-based and image-based representations because both can be implemented using standard convolutional networks on regular grids.

3.3 Overview

Our method takes as input several sketches of a shape drawn from different viewpoints (Figure 3.1a). We first use existing deep neural networks [DAI+18, IZZE17] to predict a volumetric reconstruction of the shape, along with one normal map per sketch (Figure 3.1b). We then project the normal maps on the surface of the volumetric reconstruction and combine this information with the initial surface normal to obtain a distribution of normals for each surface element (Figure 3.1c,d). While the normals coming from different sources mostly agree, some parts of the shape exhibit significant ambiguity due to erroneous predictions and misalignment between the input sketches and the volumetric reconstruction. Therefore in the next step of our approach we reconstruct a piecewise-smooth normal field by a variational method [CFGL16] that filters the noisy distribution of normals and locates sharp surface discontinuities (Figure 3.1e). The reconstruction energy is weighted by the variance of the distribution of normal vectors within each

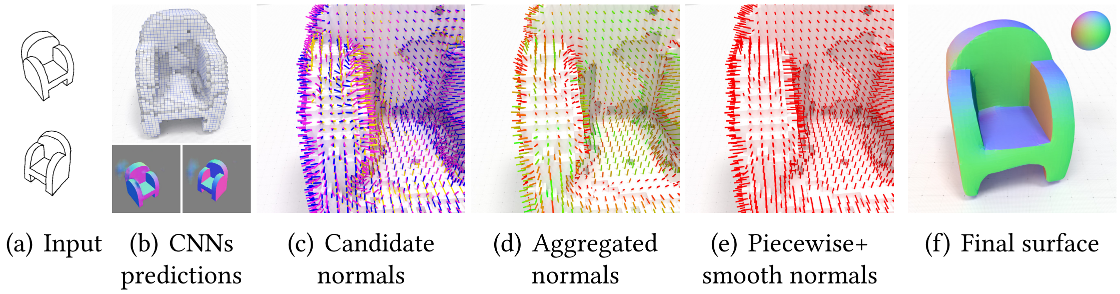


Figure 3.1: Overview of our method. Our method takes as input multiple sketches of an object (a). We first apply existing deep neural networks to predict a volumetric reconstruction of the shape as well as one normal map per sketch (b). We re-project the normal maps on the voxel grid (c, blue and yellow needles), which complement the surface normal computed from the volumetric prediction (c, pink needles). We aggregate these different normals into a distribution represented by a mean vector and a standard deviation (d, colors denote low variance in green and high variance in red). We optimize this normal field to make it piecewise smooth (e) and use it to regularize the surface (f). The final surface preserves the overall shape of the predicted voxel grid as well as the sharp features of the predicted normal maps.

surface element, which acts as a confidence estimate. Finally, we regularize the initial surface such that its quads and edges align with this normal field [CGL17], resulting in a piecewise-smooth object that follows the overall shape of the volumetric prediction as well as the crisp features of the predicted normal maps (Figure 3.1f).

3.4 Volumetric and normal prediction

Our method builds on prior work to obtain its input volumetric and image-based predictions of the shape. The full description of the volumetric prediction can be found in Chapter 2. For completeness, we briefly describe here the two types of predictions and refer the reader either to previous chapter or to the original paper for additional details.

3.4.1 Volumetric prediction

The approach presented in Chapter 2 relies on two deep convolutional networks. First, the *single-view* network is in charge of predicting occupancy in a voxel grid given one drawing as input. Then, the *updater* network refines this prediction by taking another drawing as input. When multiple drawings are available, the updater network is applied iteratively over the sequence of drawings to achieve a multi-view coherent reconstruction. Both networks follow a standard U-Net architecture [RFB15] where the drawing is processed by a series of convolution, non-linearity and down-scaling operations before being expanded back to a voxel grid, while skip-connections propagate information at multiple scales. This method produces a voxel grid of resolution 64^3 from drawings of resolution 256^2.

3.4.2 Normal prediction

We obtain our normal prediction using a U-Net similar to the one we use for volumetric prediction. The network takes as input a drawing of resolution 256^2 and predicts a normal map of the same resolution. Lun et al. [LGK+17] and Su et al. [SDY+18] have shown that this type of architecture performs well on the task of normal prediction from sketches. We base our implementation on Pix2Pix [IZZE17], from which we remove the discriminator network for simplicity.

3.5 Data fusion

The main novelty of our method is to combine a coarse volumetric prediction with per-view normal maps to recover sharp surface features. However, these different sources of information are often not perfectly aligned due to errors in the predictions as well as in the input line drawings. Prior work on multi-view prediction of depth maps [LGK+17, LPL+18] tackle a similar challenge by aligning the corresponding point sets using iterative non-rigid registration. We instead implement this data fusion in two stages, each one being the solution of a different variational formulation.

In the first stage, we project the normal predictions onto the surface of the volumetric prediction, and complement this information with normals estimated directly from the voxel grid. We then solve for the piecewise-smooth normal field that best agrees with all these candidate normals, such that sharp surface discontinuities automatically emerge at their most likely locations [CFGL16]. In the second stage, we optimize the surface of the voxel grid such that it respects the normal field resulting from the first stage, while staying close to the initial predicted voxel geometry [CGL17].

3.5.1 Generation of the candidate normal field

We begin by thresholding the volumetric prediction to obtain a binary voxel grid. The boundary of this collection of voxels forms a quadrangulated surface Q made of isothetic unit squares, which we call *surface elements* in the following. We then project the center of each surface element into each normal map where it appears to look up the corresponding predicted normal. We use a simple depth test to detect if a given surface element is visible from the point of view of the normal map. We also compute the gradient of the volumetric prediction using finite differences, which we use as an additional estimate of the surface normal. We aggregate these various estimates into a spherical Gaussian distribution, with normalized mean $\bar{\mathbf{n}}$ and standard deviation $\sigma_{\mathbf{n}}$. For surface elements not visible in any normal map, we set $\bar{\mathbf{n}}$ to the estimate given by the volumetric prediction.

3.5.2 Reconstruction of a piecewise-smooth normal vector field

For each surface element, we now have a unique normal vector $\bar{\mathbf{n}}$ as well as an estimate of its standard deviation $\sigma_{\mathbf{n}}$. We obtain our final piecewise-smooth normal field $\mathbf{n}^*$ by minimizing a discrete variant of the Ambrosio-Tortorelli energy [CFGL16].

On a manifold Ω, the components $\{n_0^*, n_1^*, n_2^*\}$ of $\mathbf{n}^*$ and a scalar function v that captures discontinuities are optimized to minimize

$$\begin{aligned} AT_\varepsilon(\mathbf{n}^*, v) := \int_\Omega \alpha \sum_i |n_i^* - \bar{n}_i|^2 + \sum_i v^2 |\nabla n_i^*|^2 \\ + \lambda\varepsilon|\nabla v|^2 + \frac{\lambda}{\varepsilon}\frac{(1-v)^2}{4} ds\,, \end{aligned} \tag{3.1}$$

for some parameters $\alpha, \lambda, \varepsilon \in \mathbb{R}$. Note that the scalar function v tends to be close to 0 along sharp features and close to 1 elsewhere.

The first term ensures that the output normal $\mathbf{n}^*$ is close to the noisy input $\bar{\mathbf{n}}$. The second term encourages $\mathbf{n}^*$ to be smooth where there is no discontinuity. The last two terms control the smoothness of the discontinuity field v and encourage it to be close to 1 almost everywhere by penalizing its overall length. Note that fixing all the n_i^* (resp. v), the functional becomes quadratic and its gradient is linear in v (resp. all the n_i^*), leading to an efficient alternating minimization method to obtain the final $\mathbf{n}^*$ and v. Parameter α controls the balance between data fidelity and smoothness. A high value better preserves the input while a low value produces a smoother field away from discontinuities. Parameter λ controls the length of the discontinuities – the smaller it is, the more discontinuities will be allowed on the surface. We use the same values $\lambda = 0.05$ and $\alpha = 0.1$ for all our results. The last parameter ε is related to the Γ-convergence of the functional and decreases during the optimization. We used the sequence $(4, 2, 1, 0.5)$ for all our results. Please refer to [CFGL16] for more details about the discretization of Equation (3.1) onto the digital surface Q and its minimization.

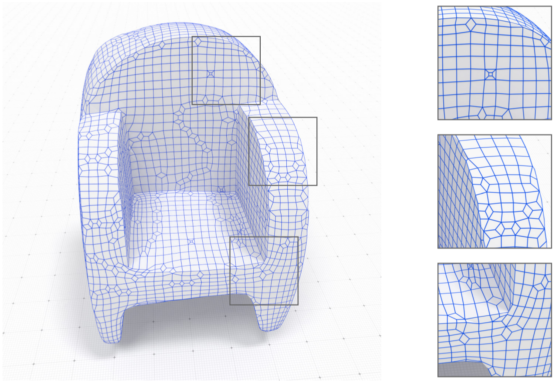


Figure 3.2: Surface reconstruction obtained from the normal field regularized with our weighted Ambrosio-Tortorelli functional (see Fig.3.1b for the input voxel grid). The insets show how the quadrangulation perfectly recovers the surface singularities.

We further incorporate our knowledge about the distribution of normals at each surface element by defining α as a function of the standard deviation $\sigma_{\mathbf{n}}$. Intuitively, we parameterize α such that it takes on a low value over elements of high variance, effectively increasing the influence of the piecewise-smoothness term in those areas:

$$\alpha(s) := 2(1 - \sigma_{\mathbf{n}}(s))^4 \, .$$

at a surface element $s \in Q$. This local weight allows the Ambrosio-Tortorelli energy to diffuse normals from reliable areas to ambiguous ones. We set $\alpha(s)$ to 0.8 for surface elements not visible in any normal map.

3.5.3 Surface reconstruction

Equipped with a piecewise-smooth normal field $\mathbf{n}^*$, we next reconstruct a regularized surface whose quads are orthogonal to the prescribed normals. We achieve this goal using the variational model proposed in [CGL17]. Let V denotes the vertices $\{\mathbf{v}_i\}$ of the input digital surface Q, F the set of (quadrilateral) faces and $\mathbf{n}^*_f$ the prescribed normal vector on face f, we solve for the quad surface vertex positions $P^* = \{\mathbf{p}^*_i\}$ that minimize the following energy function:

$$E(P^*) := \alpha' \sum_{i=1}^{n} \|\mathbf{p}_i - \mathbf{p}^*_i\|^2 + \beta' \sum_{f \in F} \sum_{e_j \in \partial f} (e_j \cdot \mathbf{n}^*_f)^2 + \gamma' \sum_{i=1}^{n} \|\mathbf{p}^*_i - \mathbf{b}^*_i\|^2 . \tag{3.2}$$

where " $\cdot$ " is the standard $\mathbb{R}^3$ scalar product, $e_j \in \partial f$ is an edge of the face f (equal to some $\mathbf{p}_k - \mathbf{p}_l$), $\mathbf{b}^*_i$ is the barycenter of the vertices adjacent to $\mathbf{p}^*_i$, and $\alpha', \beta'\gamma' \in \mathbb{R}$ (set respectively to $10^{-3}, 1$ and 10^{-1} in all our experiments).

The first term encourages the regularized surface to stay close to the input surface. The second term encourages the faces to be as orthogonal as possible to the prescribed normal vector field $\mathbf{n}^*$. The last term enhances the aspect ratio of the final quads by displacing the vertices onto their tangent planes. Since Equation (3.2) is a sum of quadratic terms with linear gradients with respect to the vertex positions, the optimal positions P^* can be obtained by solving a sparse linear system (see [CGL17] for details). As illustrated in Figure 3.2, this surface reconstruction guided by our piecewise-smooth normal vector field effectively aligns quad edges with sharp surface discontinuities.

3.6 Evaluation

We first study the impact of the different components of our method, before comparing it against prior work. For all these results, we use the dataset described in Chapter 2 and 4. This dataset contains shapes rendered from front, side, top and 3/4 views. Note however that we only train and use the normal map predictor on 3/4 views because the other views are often highly ambiguous.

3.6.1 Ablation study

Figure 3.3 compares the surface reconstructions obtained with different sources of normal guidance, and different strategies of normal fusion. We color surfaces according to their orientations, as shown by the sphere in inset. As a baseline, we first extract the surface best aligned with the gradient of the volumetric prediction, similarly to previous chapter. Because the volumetric prediction is noisy and of low resolution, this naive approach produces bumpy surfaces that lack sharp features (second column). Optimizing the normal field according to the Ambrosio-Tortorelli energy removes some of

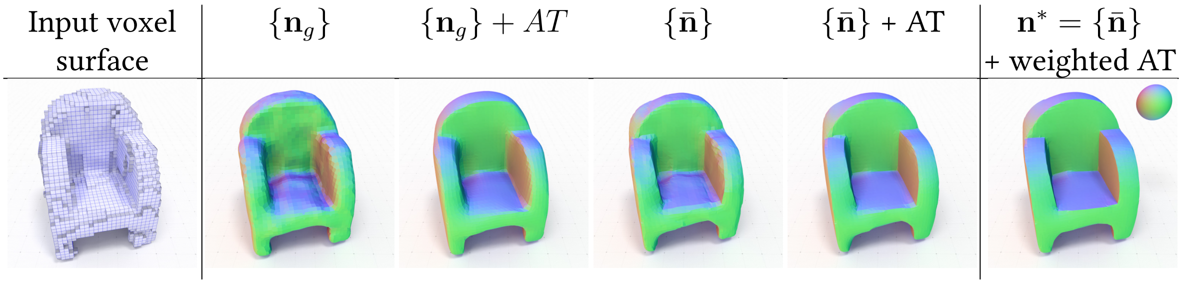


Figure 3.3: Ablation study showing the surface obtained using various normal fields as guidance. The volumetric gradient $\mathbf{n}_g$ produces bumpy surfaces that lack sharp features (second column), even after being optimized according to the Ambrosio-Tortorelli energy (third column). Our aggregated normal field $\bar{\mathbf{n}}$ yields multiple surface discontinuities where the normal maps are misaligned, such as on the arms and the seat of the armchair (fourth and fifth column). We obtain the best results by reducing the influence of the aggregated normals in areas of low confidence (last column, $\mathbf{n}^*$).

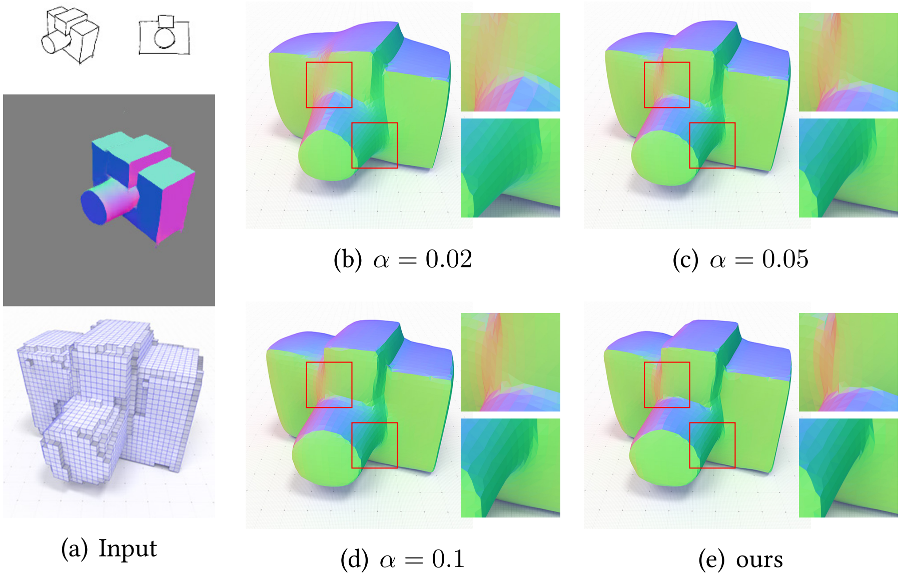
the bumps, but still produces rounded corners (third column). Aggregating the volumetric and image-based normals into a single normal field produces smoother surfaces, but yield bevels where the normal maps are misaligned (fourth and fifth column). We improve results by weighting the aggregated normal field according to its confidence, which gives the Ambrosio-Tortorelli energy greater freedom to locate surface discontinuities in ambiguous areas (last column).

We further evaluate the importance of our local weighting scheme in Figure 3.4. We first show surfaces obtained using a constant α in the Ambrosio-Tortorelli energy. A low α produces sharp creases and smooth surfaces but the final shape deviates from the input, as seen on the cylindrical lens of the camera that becomes conic (Figure 3.4b). On the other hand, a high α yields a surface that remain close to the input, but misses some sharp surface transitions (Figure 3.4d). By defining α as a function of the confidence of the normal field, our formulation produces a surface that is close to the input shape and locates well sharp transitions even in areas where the normal maps are misaligned (Figure 3.4e).

3.6.2 Performances

We implemented the deep networks in Caffe [JSD+14] and the normal field and surface optimizations in DGtal[1]. Both the prediction and optimization parts of our method take

[1] https://dgtal.org/


(a) Input

(b) $\alpha = 0.02$

(c) $\alpha = 0.05$

(d) $\alpha = 0.1$

(e) ours

Figure 3.4: Ambrosio-Tortorelli with a fixed α deviates from the input shape (b) or misses sharp discontinuities (d). Our spatially-varying α allows the recovery of sharp features in areas where the aggregated normal field has a low confidence (e).

approximately the same time. The volumetric prediction takes between 150 and 350 milliseconds, depending on the number of input sketches. The normal prediction takes around 15 milliseconds per sketch. In contrast, normal field optimization takes around 700 milliseconds and surface optimization takes around 30 milliseconds. Note however that we measured these timings using GPU acceleration for the deep networks, while the normal field and surface optimizations are performed on the CPU.

3.6.3 Comparisons

Figure 3.5 compares our surfaces with the ones obtained in Chapter 2, where we apply a marching cube algorithm on the volumetric prediction. Our method produces much smoother surfaces while capturing sharper discontinuities. While our method benefits from the guidance of the predicted normal maps, it remains robust to inconsistencies between these maps and the voxel grid, as shown on the armchair (top right) where one of the normal maps suggests a non-flat back due to a missing line in the input drawing.

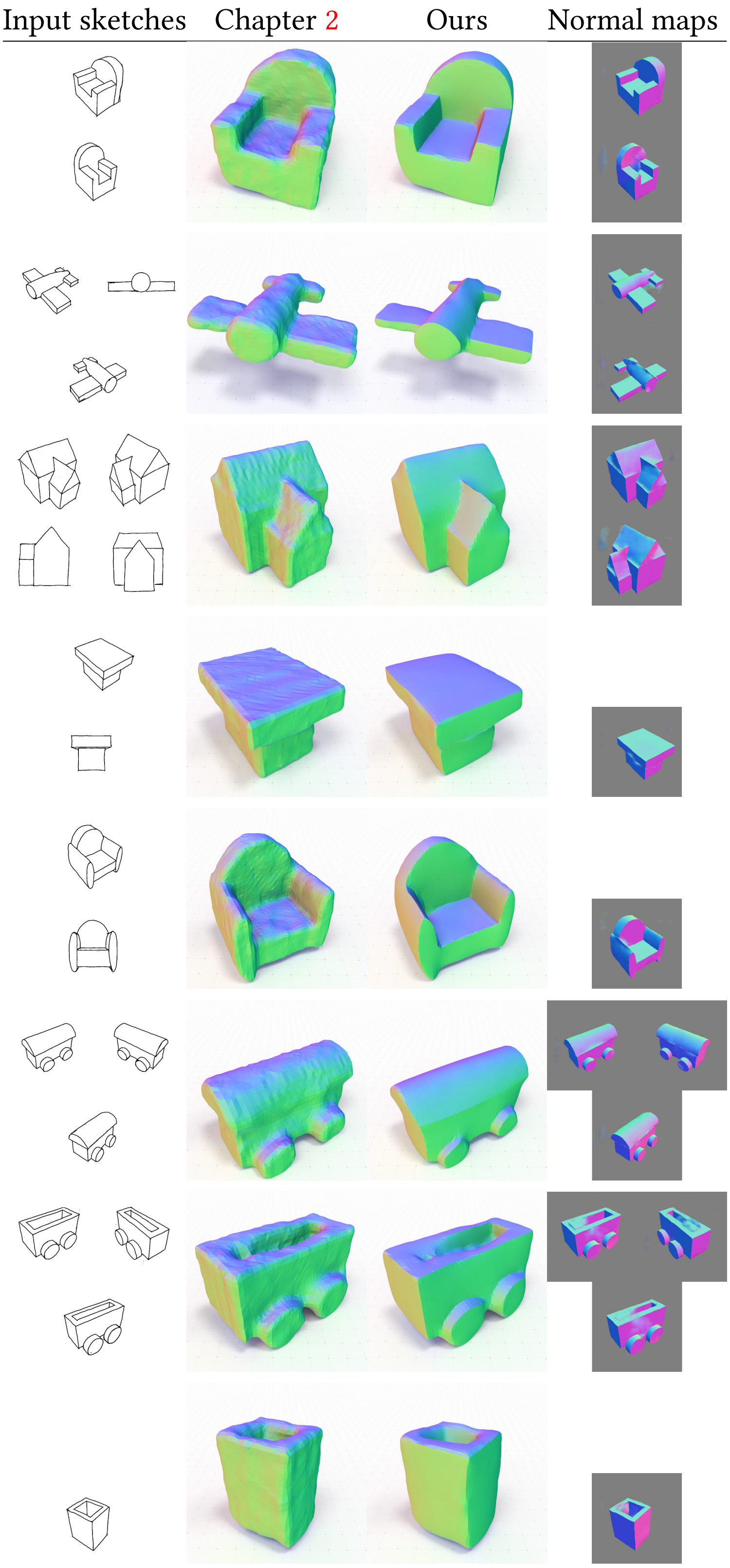


Figure 3.5: Comparison to volumetric prediction only (Chapter 2) on a variety of objects.

3.6.4 Robustness

Figure 3.6 evaluates the robustness of our method to noisy volumetric predictions, showing that our combination of normal map guidance and piecewise-smooth regularization yields stable results even in the presence of significant noise. We also designed our method to be robust to normal map misalignment, common in a sketching context. Figure 3.7 demonstrates that our method is stable in the presence of global and local misalignment. We simulate a global misalignment by shifting one of the normal maps by 5 pixels, and a local misalignment by replacing each normal by another normal, sampled in a local neighborhood.

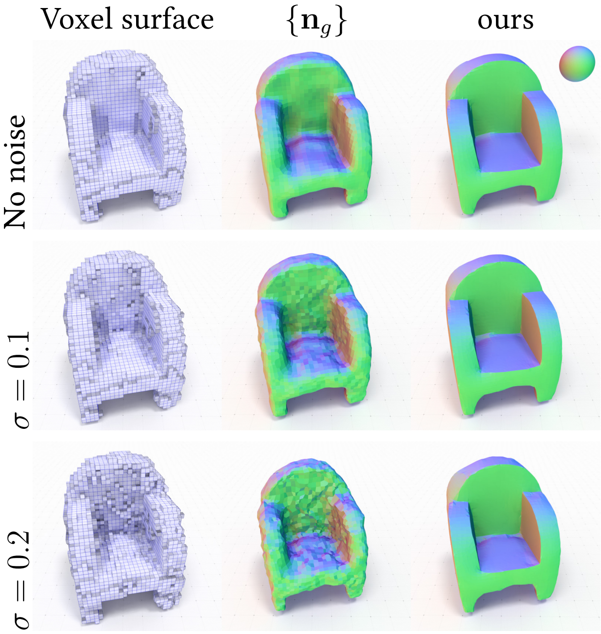


Figure 3.6: Robustness to noisy volumetric prediction. Adding gaussian noise to the input volumetric prediction has little impact on the final result.

3.6.5 Limitations

Figure 3.8 illustrates typical limitations of our approach. Since our method relies on normal maps to guide the surface reconstruction, it sometimes misses surface discontinuities between co-planar surfaces, as shown on the top of the locomotive. An additional drawing would be needed in this example to show the discontinuity from bellow. A side effect of the surface optimization energy (Equation 3.2) is to induce a slight loss of vol-

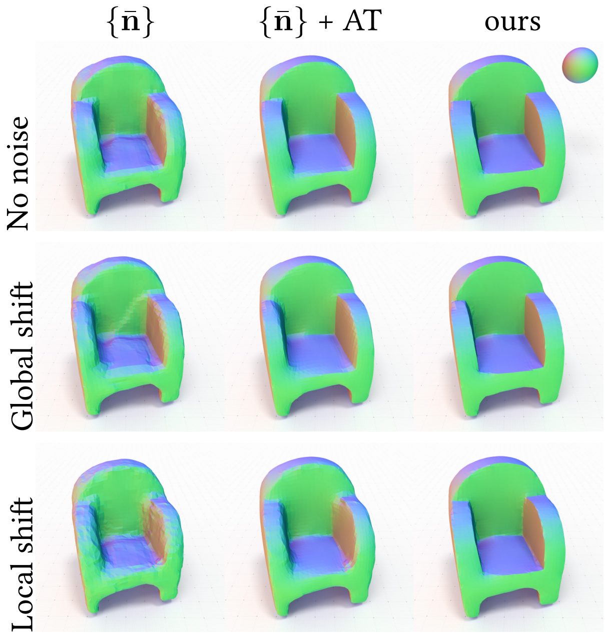


Figure 3.7: Robustness to misaligned normal maps. Here we simulate global misalignment by shifting an entire normal map by the same amount (second row) or by shifting each normal by a random amount (third row). While these perturbations degrades the result of the baseline methods, our method remains stable.

ume, which is especially visible on thin structures like the wings of the airplane and the toothbrush. Possible solutions to this issue includes iterating between regularizing the surface and restoring volume by moving each vertex in its normal direction. Another limitation of our approach is that normal maps only help recovering fine details on visible surfaces, while hidden parts are solely reconstructed from the volumetric prediction, as shown on the back of the camera in Figure 3.6.5. Finally, because we favor piecewise-smooth surfaces, our approach is better suited to man-made objects than to organic shapes made of intricate details.

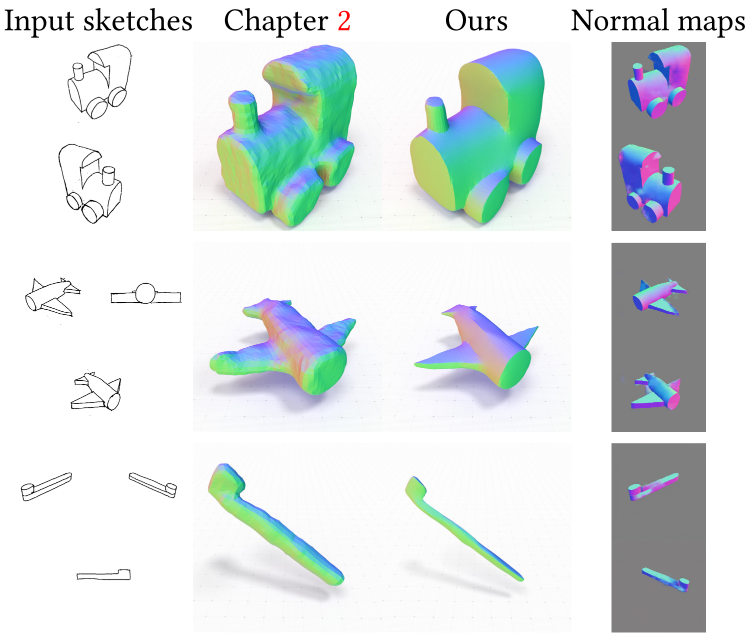


Figure 3.8: Limitations of our method. Our method cannot recover surface discontinuities that are not captured by the normal maps, such as the top of the locomotive. The surface optimization tends to shrink the object, as seen on thin structures like the wings of the airplane and the toothbrush.

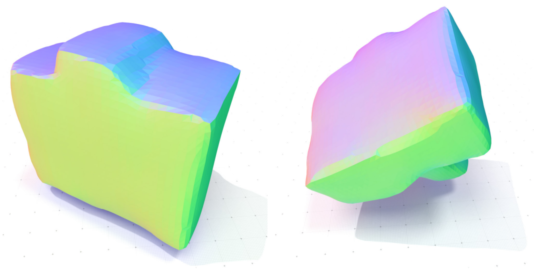


Figure 3.9: Since normal maps only capture visible surfaces, the back and bottom of this camera is solely defined by the volumetric prediction. Nevertheless, the method reconstructs a smooth surface in such cases as it still benefits from the piecewise-smoothness of the Ambrosio-Tortorelli energy.

3.7 Conclusion

In this chapter, we showed how volumetric and normal map predictions can be combined, using the volumetric representation to capture hidden parts and the image-based representation to capture sharp details. We use the volumetric representation as a support for normal map fusion by solving for a piecewise-smooth normal field over the voxel surface. This method is especially well suited to man-made objects dominated by a few sharp discontinuities.

Chapter 4

Generating synthetic drawings for training

Part of this work was done in collaboration with Mathieu Aubry (École des Ponts), Phillip Isola (MIT) and Alexei A. Efros (UC Berkeley). It was presented at *I3d 2018* and published in *Proceedings of the ACM on Computer Graphics and Interactive Techniques* in 2018. Bastien Wailly implemented the contour detector and noise functions presented in Section 4.3.3.

4.1 Introduction

One of the major challenge of using deep learning is the access to training data. In our case, collecting thousands of hand-drawings registered with 3D objects would be very costly and time consuming. Similarly to prior data-driven approaches [ERB+12, XXM+13, HKYM16, NGDGA+16], we alleviate the need for collecting real-world drawings by generating *synthetic* line drawings from 3D objects using non-photorealistic rendering. This allows us to easily adapt our system to the design of different types of objects by generating training data that is representative of such objects.

However, it is not obvious to find a database that would allow the design of a large variety of shapes. In online repositories of real objects, like ShapeNet, a few classes such as cars or chairs concentrate the majority of objects. This could prevent the user to design shapes out of this space. We instead propose to use abstract shapes generated procedurally. This allows to generate a sufficient amount of varied objects so that the system learns to interpret the lines instead of learning to recognize categories of objects. We design a simple shape grammar that assemble geometric primitives (cuboids, cylinders) in a random way to produce symmetric objects and show that it allows to design a large variety of shapes. We also train our single view network on a small dataset of chairs and vases and show that, even if each network performs best on the database on which it was trained, our procedural dataset allows to capture the global shape quite accurately for other datasets.

We generate our contours using standard contour extraction methods. But these clean line drawings are not representative of how human draw. Even good artists draw noisy lines because of slight shaking of the pen or imprecise direction of the line. We thus create a new dataset of contours that mimics some of the errors made by humans. Unfortunately, our preliminary experiment do not allow to conclude on the usefulness of this addition.

4.2 Related work

4.2.1 Synthetic data for machine learning

Access to training data is notoriously a major challenge for deep learning methods. The use of synthetic images has become common in a variety of applications for natural images, such as segmentation, normal estimation, 3D reconstruction or material acquisition. Depending of the targeted application, the method to generate images varies from physically based rendering of indoor scenes [ZSY+17] to rendering of individual objects composited with random background [SQLG15], but also capturing frames from realistic videogames [RVRK16] or generating fake materials with dedicated softwares [DAD+18].

For sketch-related applications, the use of synthetic data is even more crucial. While it is possible to collect a large quantity of (non labeled) photographs online, collecting sketches is much harder. Datasets of cartoon drawings of objects have been collected via crowd-sourcing [ERB+12] but building a dataset of perspective drawings registered with 3D models raises many additional challenges. In particular, we assume that users of our system are sufficiently skilled to draw 3D objects in approximate perspective, which may not be the case for the average Mechanical Turk worker [EHA12]. In addition, crowd-sourcing such a drawing task would require significant time and money, which prevents iterative design of the dataset, for instance to adjust the complexity and diversity of the 3D shapes. For these reason, synthetic drawings extracted from 3D objects have been used in all sketch based retrieval or modeling systems, since the method of Eitz et al. in 2012 [ERB+12] up to the most recent ones [HKYM16, NGDGA+16, LPL+18, LGK+17]. All of them use standard contour extraction methods.

4.2.2 Non Photorealistic Rendering

The problem of extracting line contours from 3D objects has been long studied in the Non Photorealistic Rendering community. We refer the reader to the recent tutorial by Hertzmann and Bénard [BH18] for a survey of the different types of contours that can be extracted and how they can be stylized. In 2008, Cole et al. [CGL+08] analyze how human-drawn lines align with these line contours and show that the majority but not all lines drawn by artists align with synthetic contour lines.

For simplicity reasons, we only use contours in this project but several works have shown that other types of sketching features can be rendered such as hatching [HZ00, KNBH12a], shading [XGS15b] or construction lines [SIJ+07]. Moreover, a programmable Non Photorealistic Renderer for line is now available in Blender [GTDS04], making easier the rendering of more complex styles.

4.3 Generation of contours

Similarly to recent work on sketch-based procedural modeling [NGDGA+16, HKYM16], we bypass the challenges of real-world data collection by generating our training data using non-photorealistic rendering. This approach gives us full control over the variability of the dataset in terms of shapes, rendering styles, and viewpoints.

4.3.1 3D objects

The main strength of our data-driven approach is its ability to capture the characteristic features of a class of objects. We experimented with two sources of 3D object datasets: online repositories and shape grammars.

Online repositories. A first usage scenario of our system is to train it with specific object classes. For instance, a furniture designer could train the system with chairs and tables, while a car designer could train the system with various vehicles. In practice, we tested this approach with the two largest classes of the ShapeCOSEG dataset [WAvK+12], which contain 400 chairs and 300 vases, some of which are shown in Figure 4.1. For each dataset, we used 90% of the objects for training and the other 10% for testing.

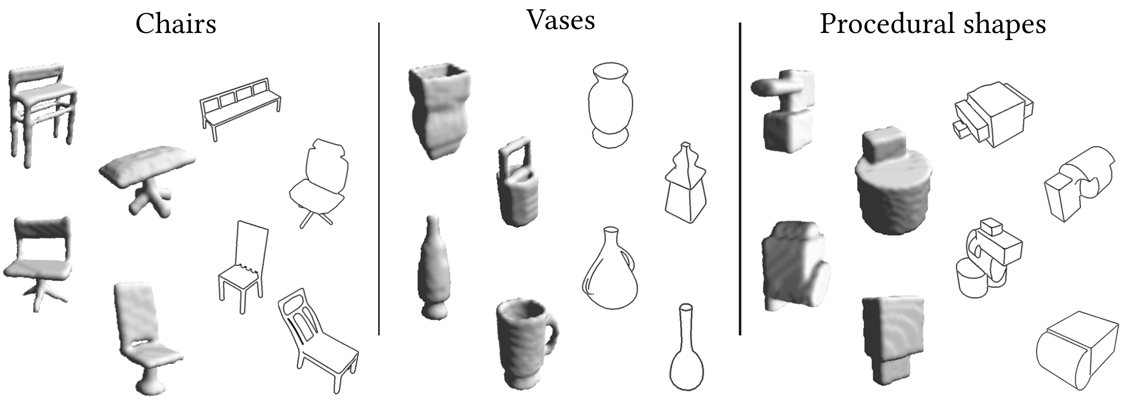


Figure 4.1: Representative voxelized objects and drawings from our three datasets.

Shape grammars. One drawback of online shape repositories is that they are dominated by a few object classes, such as tables, chairs, cars and airplanes [CFG+15], which may not cover the types of objects the user wants to model. In addition, many of these objects are very detailed, while we would like our system to also handle simple shapes to allow coarse-to-fine explorative design. We address these limitations by training our system with abstract shapes with varying degrees of complexity.

We designed a simple procedure that generates shapes by combining cubes and cylinders with CSG operations. Our procedure iteratively constructs a shape by adding or substracting random primitives. At each iteration, we position the new primitive on a random face of the existing shape, scale it by a random factor in each dimension and displace it by a small random vector while maintaining contact. The primitive is either merged with or subtracted from the existing shape. We inject high-level priors in this procedure by aligning each primitive with one of the three world axes, and by symmetrizing the shape with respect to the xy plane in world coordinates. The resulting axis-aligned, symmetric shapes resemble man-made objects dominated by flat orthogonal faces, yet also contain holes, concavities and curved parts. We generated $20,000$ random shapes with this procedure, some of which are shown in Figure 4.1. We isolated 50 of these shapes for testing, and used the rest for training.

Voxelization. We voxelize each object at a resolution of 64^3 voxels using Binvox [NT03, Min16]. We scale each object so that the voxel grid covers 120% of the largest side of the object's bounding box.

4.3.2 Line rendering

For the results presented in Chapter 2 and some in this chapter, we adopt the simple image-space contour rendering approach of Saito and Takahashi [ST90], who apply an edge detector over the normal and depth maps of the object rendered from a given viewpoint. Edges in the depth map correspond to depth discontinuities, while edges in the normal map correspond to sharp ridges and valleys. We render each drawing at a resolution of 256^2 pixels.

We adopt a different object-based detector to experiment with noisy drawings. Indeed, object-space contours provide us with vectorial lines that facilitate stylization. This rendering is used only for the study on the influence of noise and details about how we generate them are presented in the next section.

4.3.3 Incorporating noise

As shown in Figure 4.2, our volumetric network trained with perfect data is moderately robust to the noise present in human drawings. It offers robustness to small amount of noise, such as wavy, incomplete or overshot lines and slight perspective distortions although the quality of the final objects slightly decreases. However, drawings made under drastically different or wrong perspectives yield distorted shapes. We also observed sensitivity to over-sketching and varying line thickness.

In order to reduce this sensitivity to noise, we generated a new dataset of contours that incorporate synthetic noise. We first extract contours from the objects in 3D space by detecting occluding contours and sharp changes of orientation (if the angle between two triangles is larger than 60 degrees). This simple detector is sufficient for our procedural objects as they do not contain smooth ridges and valleys. We then test the visibility of each portion and chain them so that each visible line in 2D is represented by a unique vectorial line. Finally, we apply noise functions on each of these lines to mimic errors made by human.

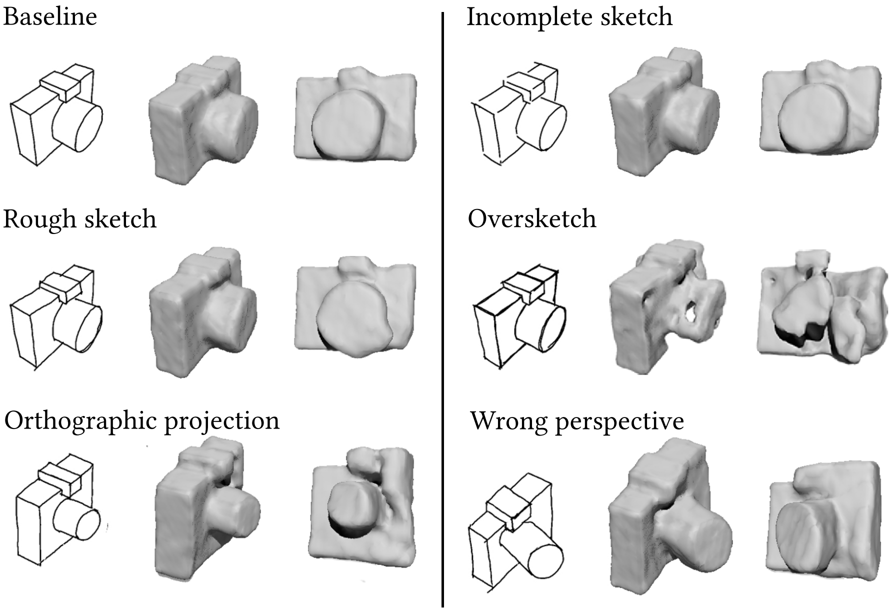


Figure 4.2: Behavior of the single-view network on various sources of noise. While the network trained on clean drawing tolerates some amount of sketchiness, overshoot and incompleteness, it is sensitive to over-sketching that produces thicker lines than the ones in the training set. Drawing with a very different or wrong perspective yields distorted shapes.

We implemented three different types of noise usually made by humans, shown in Figure 4.3:

Noisy lines This simulates the shaking of the pen. We sample each line with a constant density and displace each point along the normal to the line. We move each point in a coherent way with its neighbors so that the noise smoothly changes along the line.

Imprecise endpoints This simulates errors in the location of endpoints of lines which usually results in gaps or crossing near corners. We randomly displace endpoints along the tangent of the line.

Oversketching This simulates line repetitions that occurs when the user draw several times the same line. We repeat a random amount of time (between 1 and 2) each line with a different noise. However, the different lines are not clearly visible in our results because the noise is locally small.

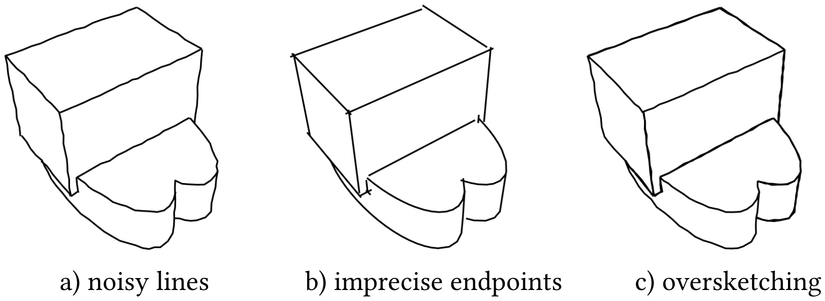


Figure 4.3: We simulate three types of noise that we observe in human drawings: a) noisy lines, b) imprecise endpoints, c) oversketching.

We also allows the thickness of each line to vary around the initial value. Finally, we generate our dataset by randomly sampling the amount of noise for each drawing. A few samples of generated drawings are shown in Figure 4.4.

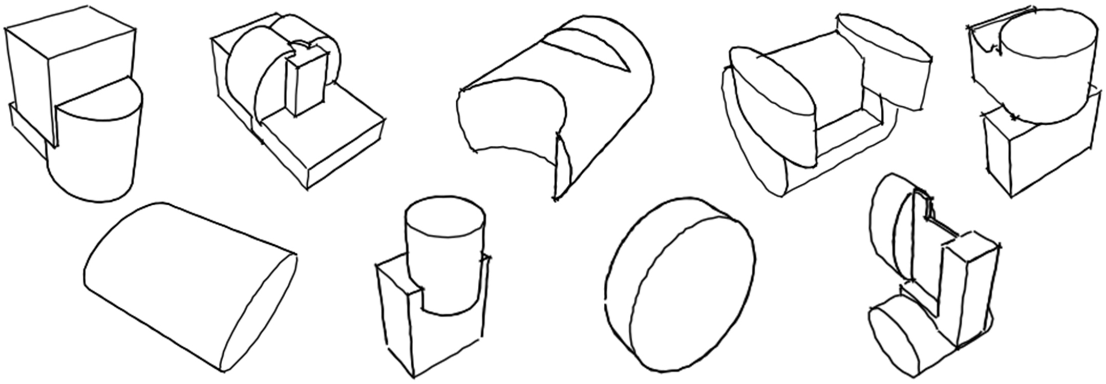


Figure 4.4: Examples of noisy drawings generated by our procedure. The amount of noise increases from left to right.

4.4 Results and evaluation

In the first part of this evaluation, we use the image-space contours described in Section 4.3.2 along with the volumetric single view network presented in Chapter 2 to evaluate the performances on several objects databases. In the second part, we evaluate how the robustness evolve when using the noisy contours described in Section 4.3.3 for the normal prediction network used in previous chapter.

4.4.1 Datasets

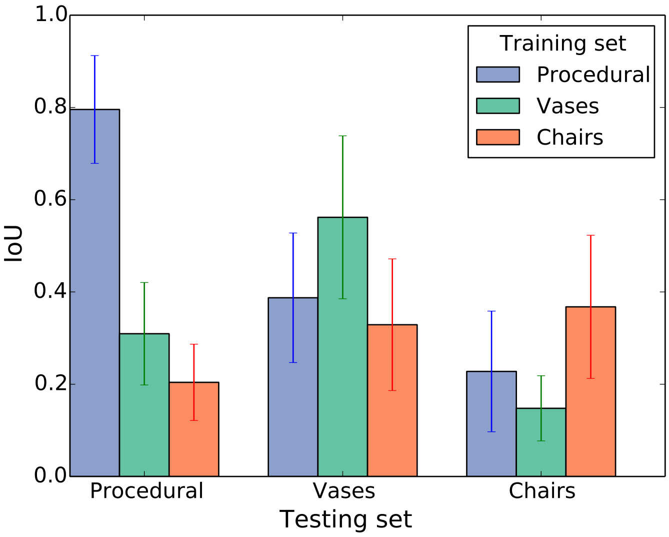


Figure 4.5: We trained the single-view network with three different datasets (depicted with different colors), and evaluated each version on drawings from the three testing sets (distributed on the x-axis). The network trained on abstract procedural shapes captures the overall shape of objects from other categories, while the networks trained on chairs and vases generalize poorly. Each network performs best on the shapes for which it has been trained.

One of the motivations for our deep-learning-based approach is to allow adaptation to different classes of objects. Figure 4.5 provides a quantitative evaluation of this ability for the single-view network. This figure plots reconstruction quality over the three testing datasets, using all three training datasets. As expected, the network trained on a given dataset performs best on this dataset, showing its specialization to a specific class. For instance, only the network trained on vases succeeds to create hollow shape from an ambiguous drawing (fourth row, third and fourth column). Interestingly, the network trained on abstract procedural shapes is second best on the other datasets, suggesting a higher potential for generalization. Figure 4.6 shows representative results for each condition. While the networks trained on chairs and vases manage to reconstruct objects from these classes, they fail to generalize to other shapes. In contrast, the network

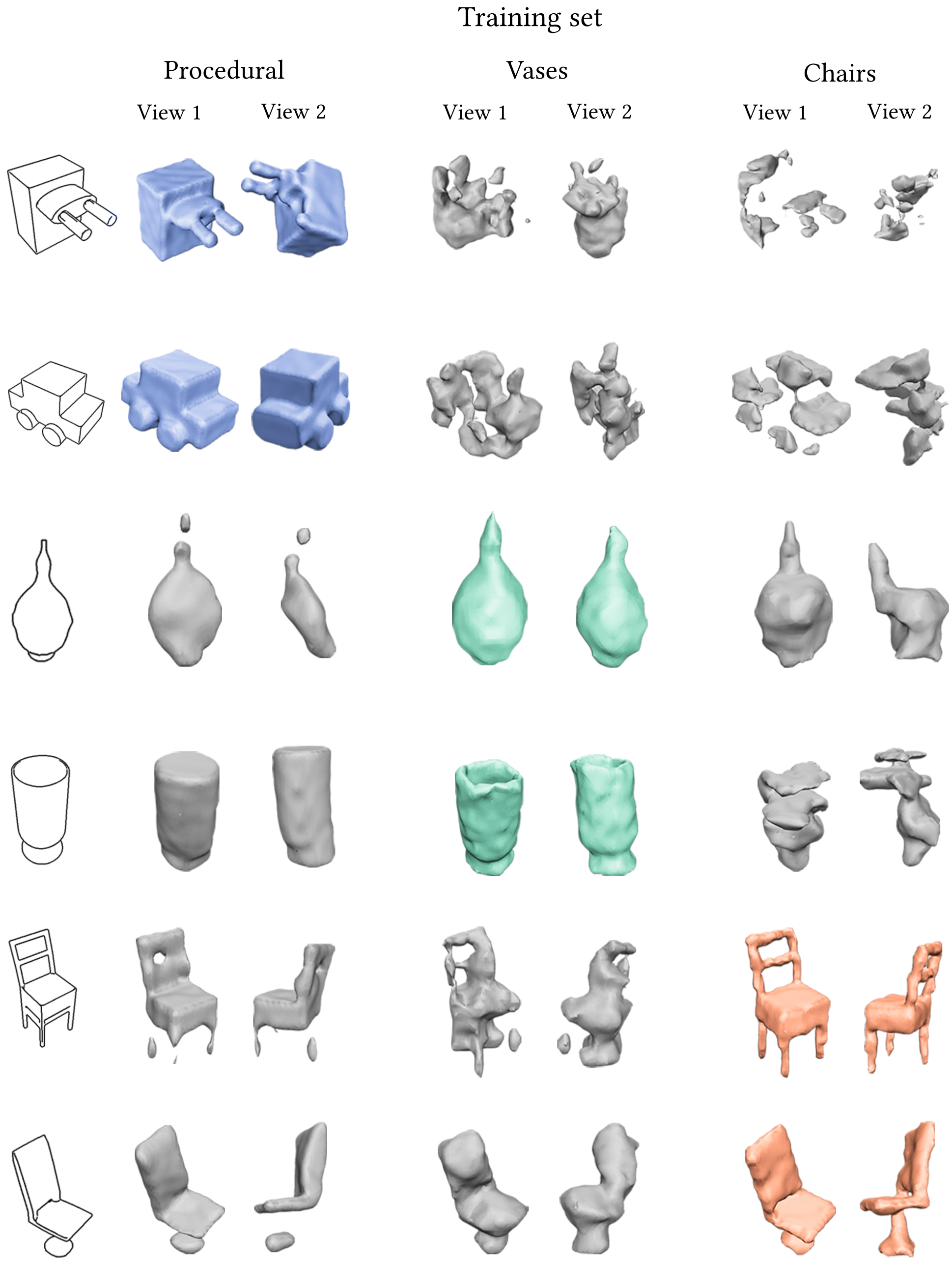


Figure 4.6: We trained the single-view network with three different datasets, and evaluated each version on the three testing sets. Each version performs best on the testing set for which it was trained, showing that the network learns to specialize to specific object categories. The network performs better on abstract procedural shapes than on chairs and vases, which contain more thin structures and for which the training set is smaller.

trained on abstract shapes captures the overall shape of chairs and vases, although it misses some of the details. This superiority of the procedural dataset may be due to its larger size and variability.

4.4.2 Influence of noisy lines

To evaluate how the addition of noise impacts the results of the network, we trained a normal prediction network (presented in the previous chapter) with three different variants of contours: the image-space contours presented in Section 4.3.2, the object-space contours presented in Section 4.3.3, and the object-space contours with the addition of noise (referred to noisy contours in the rest of this section).

We first evaluate the performance of each trained network over the different datasets by measuring the mean angular error over 800 synthetic sketches. These results are summarized in Figure 4.7. As expected, all networks performs better on clean contours (image-space or object-space) than on noisy ones, as shown by the three right-most bars in the plot. Moreover, as the two types of contours we extract are slightly different, the network trained with object-space contours performs better than the others on a testing set composed of image-space contours. The same applied for object-space contours.

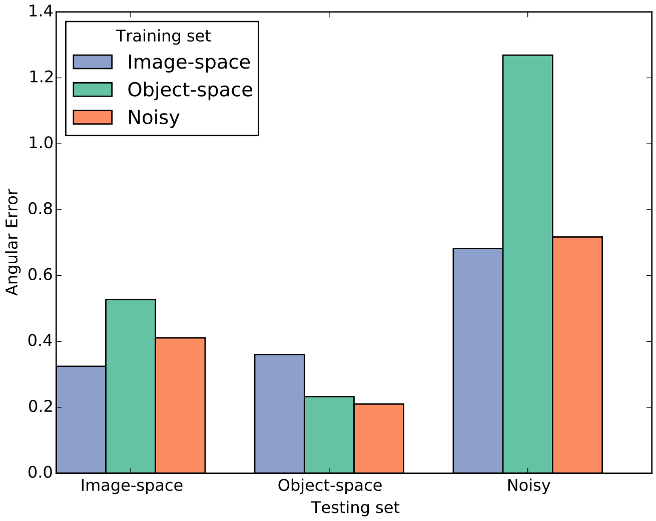


Figure 4.7: We trained a normal prediction network with the three different types of contours (image-space, object-space and noisy object-space) and test them against the same three categories. While all networks give better performances when tested on clean lines, the object-space contours give poor results when tested against a noisy dataset.

However, the behaviour of the three networks when tested with noisy contours is less intuitive: while the network trained on noisy contour gives better results than the one trained with clean object-space contours, both are surpassed by the one trained on image-space contours. Interestingly, the network trained with image-space contours seems to handle our noise much better than the two networks trained with object-space contours (clean or noisy).

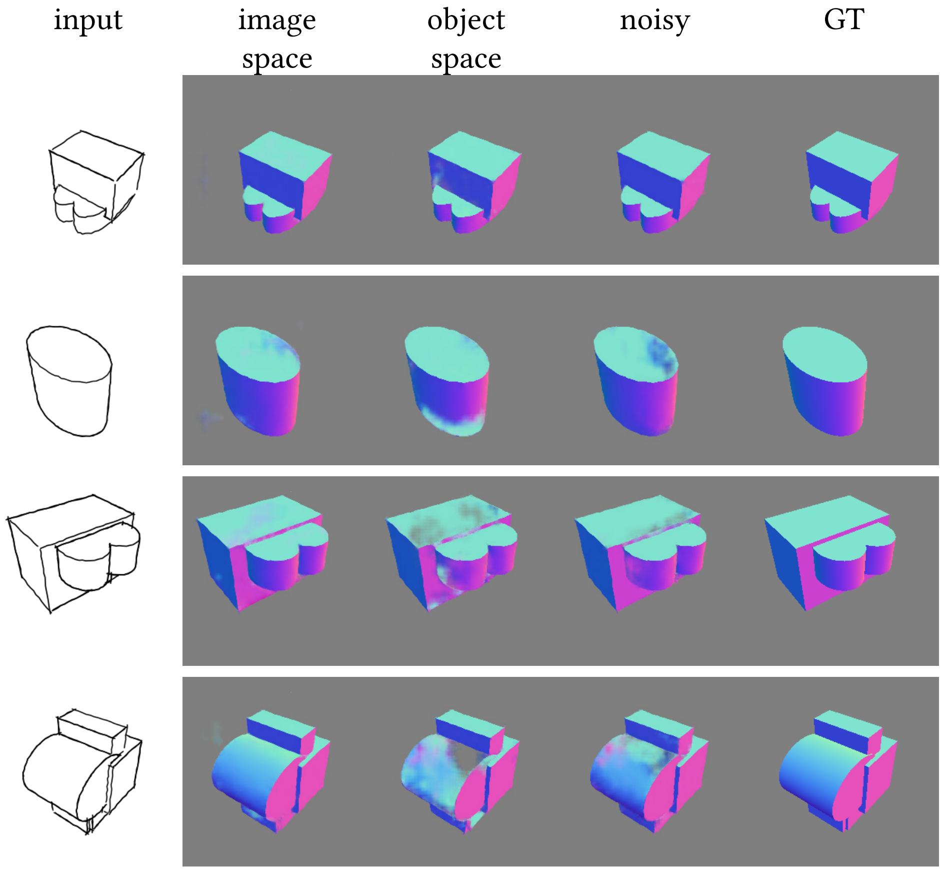


Figure 4.8: We fed synthetic noisy images to the three networks trained with different datasets. The network trained with image-space contours gives correct results while the ones trained with object-space contours produce strong artifacts.

To understand better these results, we show a few representative examples of noisy synthetic drawings evaluated by the three versions of the network in Figure 4.8. While the network trained on image-space contours gives satisfying results with only minor errors, the ones trained with object-space contours produce important errors by predicting some background normals (set to 0) in the interior of the shape. It also misses some large surfaces like the cylinder part of the last object. The network trained with noisy data gives slightly better results but still produces big artifacts.

The same comparison on real data gives similar results. In Figure 4.9, we feed the three networks with data drawn by humans. The first line shows the camera from Figure 4.2 with exaggerated noise while the other lines show sketches drawn in the application from Chapter 2 The observations are the same than with synthetic data: the networks trained with object-space contours produce strong artifacts, creating big holes in the resulting normal maps.

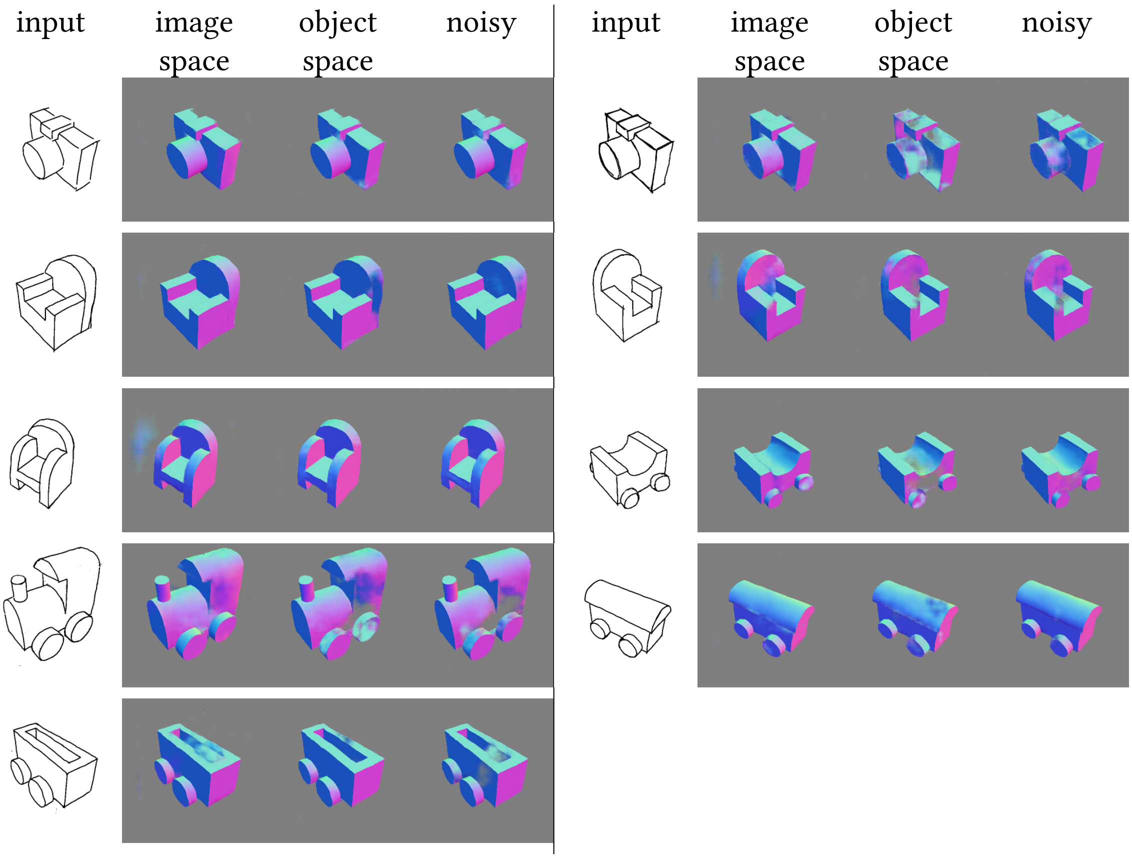
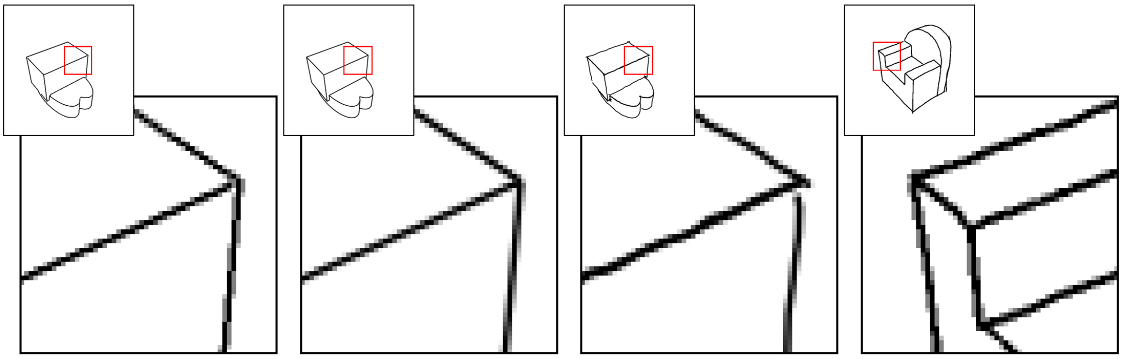


Figure 4.9: We fed real noisy images to the three networks trained with different datasets: the camera on the first line has been drawn with exaggerated noise on purpose, other drawings come from the application. The network trained with image-space contours (first column) gives correct results while the ones trained with object-space contours produce strong artifacts (second and third column).

One hypothesis that we have to explain this behaviour is the difference in anti-aliasing of the lines. Indeed, image-space and object-space contours look very similar at a large scale. But if we zoom in on the lines, we can observe that the anti-aliasing is different between the three versions of synthetic contours and the sketches from the application, as shown in Figure 4.10. Our image-space contours are computed at high resolution (1024^2) and downsample to the input size of the network. As a result the anti-aliasing is not perfectly smooth. This effect is particularly visible on the almost vertical line shown in the example. On the opposite, object-space contours produce perfectly smooth anti-aliased lines although a bit less regular with the incorporation of noise. Noisy contours also exhibits slight variations on the line thickness. More importantly, the lines drawn in the application (such as the ones shown in Figure 4.9) do not have a regular anti-aliasing.


a) image-space contours

b) object-space contours

c) object-space contours with noise

d) sketch drawn in the application

Figure 4.10: Extreme zoom on the lines shows different anti-aliasing effects on the different types of lines.

The lines drawn by users are thus similar to the one extracted with image-space contours. We think that this could explain the difference of performance on real data. Moreover, object-space contours always have very clean and smooth anti-aliased lines. This could potentially make the network trained with these contours very sensible to any irregular patterns in the lines and explain its bad performances on noisy inputs. This suggests that we need to be careful to match the training data with the real inputs of the system, not only for objects and viewpoints, but also for line rendering. More experiments need to be done to verify this hypothesis and understand better these differences.

Moreover, the normal prediction network is naturally robust to the type of local noise that we mimic in our noise procedure. In order to get more meaningful results, it will be necessary to simulate larger scale noise (such as translating or rotating each stroke randomly) and to analyze the results on the volumetric network, that seems to be less robust to large amount of noise.

4.5 Conclusion

In this chapter, we demonstrated that the use of abstract procedural shapes as a training database allows to reconstruct a large variety of man made objects. We also showed that the network trained with this dataset generalizes better than networks trained on a class specific dataset.

We extract standard contours from these objects and mimic some errors made by humans. Further investigation is needed to conclude on the usefulness of noise in training data.

Chapter 5

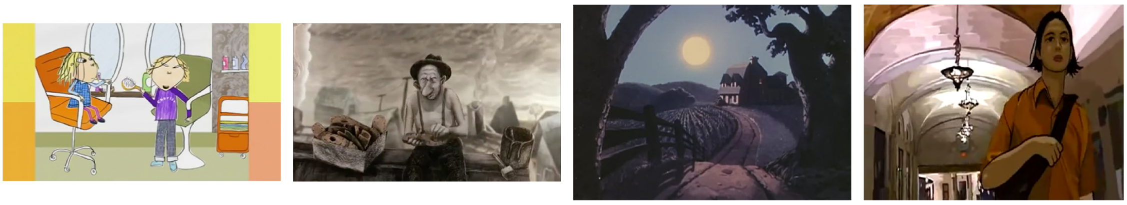
Image-space motion rigidification for video stylization

This work was done in collaboration with Aaron Hertzmann (Adobe Research). It will be presented at *Expressive 2019*.

5.1 Introduction

The goal of *video stylization* is to give a video the look of having been created with an artistic medium, such as oil painting or watercolor. Past research in non-photorealistic animation has worked hard to ensure "temporal coherence", generally taken to mean avoiding flickering artifacts, while also following optical flow [HE04, Lit97, HP00, BCK+13, BNTS07, OH12, SED16, RDB18]. We believe that some of the most recent methods have become *too successful* at it: too much temporal coherence creates the uncanny and unappealing effect of a 3D world covered in paint, rather than of a painting of a 3D world (*e.g.* [SED16, RDB18]). Some previous works have injected noise into animation in the quest for a more hand-made look [FLJ+14, KP11, FJS+17]. Our work explores a different avenue – inspired by traditional cut-out and multi-plane animation – to create *motion* that looks hand-drawn, rather than being *too faithful* to the input.

We complement existing methods by introducing *motion rigidification*, which consists of deforming a video so that its motion becomes as piecewise-rigid as possible in image space. When advected along our modified optical flow, style elements undergo 2D rigid transforms and uniform scaling rather than tracking 3D trajectories. We enforce similarity (rotation, translation, and scaling) rather than strict rigidity, since scaling is necessary to model objects that move away or toward the observer. The resulting stylized videos exhibit a very "2D look;" this look is reminiscent of traditional cut-out animations like *Charlie and Lola* and *Village of Idiots* (Figure 5.1) where objects are animated by moving their parts rigidly from frame to frame, and by replacing the parts when they deform



© Charlie and Lola © Village of Idiots © Disney © Waking Life

Figure 5.1: Our method takes inspiration from traditional cut-out and multi-plane animations. See our video for animated versions.

significantly.

Our approach is inspired by the work of Breslav et al. [BSM+07], who hypothesize that, since style elements are traditionally drawn in 2D, they should move in 2D to preserve their hand-drawn appearance. However, their approach was limited to a very specific type of texture-mapped 3D rendering. In addition, while their method changes the motion of the stylization texture, it keeps the underlying object unchanged, which yields motion discrepancies at silhouettes. We build on their approach and generalize it to arbitrary videos and to any stylization algorithm that takes an optical flow as input.

Our solution includes three main components:

- A motion segmentation algorithm that decomposes a video into near-rigid pieces. Users can control the segmentation with scribbles, for instance to capture motions that are subtle yet contribute to the intended story.
- A motion optimization algorithm that warps pixel trajectories such that they form as-rigid-as-possible segments while deviating as-little-as-possible from the original trajectories.
- A video re-rendering algorithm that synthesizes a video whose motion conforms with prescribed pixel trajectories. The output of our method is thus a new video aligned with its rigidified optical flow, which can be used as input to any video stylization method.

We demonstrate the effectiveness of our method by rigidifying videos with complex motions (animals, humans, natural scenes), which we subsequently stylize with a recent by-example style transfer algorithm [GEB16, RDB18].

5.2 Related work

Video stylization has been an active topic in Non-Photorealistic Rendering for more than two decades [Lit97, Mei96], as surveyed by Bénard et al. [BBT11], Kyprianidis et al. [KCWI13], and Rosin and Collomosse [RC13]. We first discuss the main approaches to stylize videos, before discussing related methods on motion estimation and processing.

Video stylization. The earliest methods for stylized animation targeted oil painting, where individual brush strokes are clearly visible [Mei96, Lit97]. The most common strategy to produce such a style consists in distributing brush strokes to cover the first frame of the animation, moving the strokes to the next frame using optical flow, and removing or adding strokes to avoid overlaps and gaps [Lit97, HP00, HE04]. These approaches have later been extended to styles like watercolor by advecting [BNTS07] or filtering [KP11] a stylization texture from frame-to-frame. Recent methods employ by-example texture synthesis to handle an even wider range of styles [BCK$^+$13, BBRF14, SED16, FJS$^+$17, RDB18]. These methods cast the synthesis as a global optimization that strives to reproduce the appearance of an exemplar while maintaining temporal coherence along optical flow. However, enforcing temporal coherence too strictly results in rather artificial results, which motivated Fišer et al. [FLJ$^+$14, FJS$^+$17] to inject randomness in the synthesis to mimic the temporal noise of traditional animations. Our work is largely complementary to all these methods, since our goal is not to improve how the stylization follows the video motion, but rather to modify that motion to look more hand-drawn.

Our approach follows the idea of Breslav et al. [BSM$^+$07], who stylizes 3D animations using 2D patterns that approximate object motion with similarity transforms. Their main idea and results are highly inspirational, but the specific approach they took has numerous limitations. In particular, their method only applies to textured 3D models, with predefined texture segmentation. In addition, their method does not modify the motion of the underlying 3D objects, resulting in visible sliding of the patterns along silhouettes where the input and modified motion differ significantly. Our approach addresses these limitations to produce rigidified videos that are compatible with a large body of existing video stylization methods.

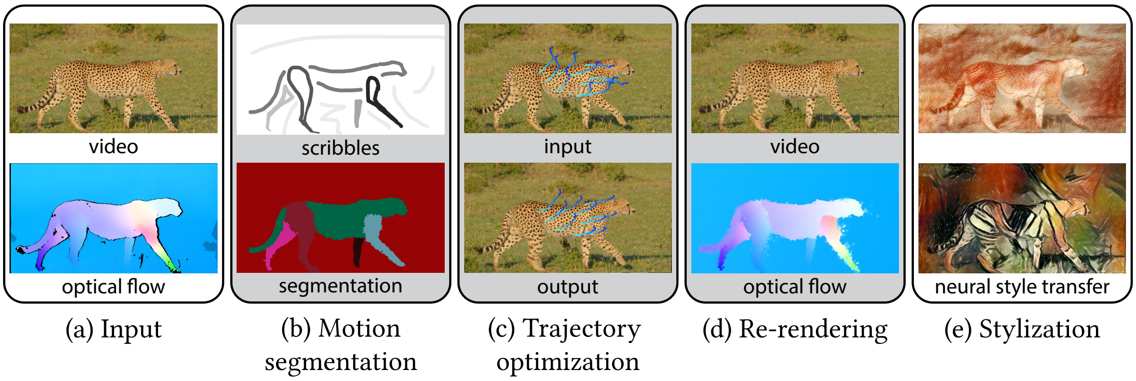
Our approach is inspired by the 2D motion produced by traditional animation techniques, such as paper cut-out. Barnes et al. [BJS+08] described an animation system dedicated to this technique, where users animate characters made of one or several rigid parts. Each part is rendered with a constant texture, which mimics traditional animations where the same piece of paper is moved from frame to frame. In contrast, we designed our method to best preserve the original appearance of the input video, including temporal variations of texture and shading within each rigid piece. We leave the choice of abstracting away such variations to the subsequent stylization algorithms that can be applied on our output.

Motion estimation and processing. While we aim at simplifying motion in a video, several methods aim at *magnifying* motion [LTF+05, WRS+12, WRDF13]. In particular, our method follows the main processing steps of Liu et al. [LTF+05] – motion segmentation, motion modification, and video re-rendering. However, our implementation of the two first steps differs. For motion segmentation, Liu et al. group correlated trajectories, while we group pixels that follow the same similarity transforms. For motion modification, Liu et al. simply apply a scaling factor on the motion vectors, while we optimize for new trajectories that are as-rigid-as-possible while staying close to the input. Closer to our application domain, Collomosse et al. [CRH05], Lee et al. [LYKL12] and Wang et al. [WDAC06] magnify motion in videos to reproduce the classical “squash and stretch” effects of cartoon animations, while Dvorožňák et al. [DLKS18a] transfer these effects from a hand-drawn exemplar. Finally, local rigidity has been used as a regularizer in optical and scene flow computation [VSR13, YL15], which is complementary to our goal of modifying the video to achieve rigid motion.

5.3 Overview

Given a video and its optical flow, our goal is to generate a new video and optical flow such that

- The new video is composed of large segments that follow similarity transforms from frame to frame,
- The pixel trajectories in the new video are close to the pixel trajectories in the original video,
- The new video and its optical flow are well aligned.



(a) Input (b) Motion segmentation (c) Trajectory optimization (d) Re-rendering (e) Stylization

Figure 5.2: Overview of our method. Given an input video and its optical flow (a), we first employ interactive segmentation to decompose the video into parts that approximately move rigidly (b). We then optimize pixel trajectories such that they remain close to the input trajectories, while being as rigid as possible (c, magnified for visualization). The optimized trajectories are then used to re-render the video and its optical flow (d), which can serve as input to any existing stylization algorithm (e).

We address these objectives as three separate computational steps — motion segmentation, trajectory optimization, and video re-rendering (Figure 5.2).

We first cast motion segmentation as a labeling problem, where pixels of a frame receive the same label if their optical flow is approximated well by the same similarity transforms (Section 5.4). Our formulation includes spatial and temporal smoothness terms to favor the emergence of large segments that move coherently during multiple frames. Since the number and shape of the segments greatly impact the outcome of our method, we provide artistic control on this step by means of user scribbles.

However, while the similarity transforms found for each segment only introduce subtle deviations from the original optical flow, accumulating these deviations over multiple frames would yield significant drift of the video content. We address this issue in a second step, where we track pixels along extended sequences and optimize the resulting trajectories to best satisfy the local rigid motion while minimizing global drift (Section 5.5).

Our last step consists in warping the video according to the displacement of the optimized trajectories. Note that, unlike conventional methods that estimate optical flow *from* a given video, this step entails generating a new video that follows the given flow. Once re-rendered, our rigidified video is ready to be processed by any existing video stylization algorithm.

5.4 Motion segmentation

The first step of our method takes as input a video and its optical flow and segments it into parts such that, for each frame of the video, the optical flow within each part is well approximated by a similarity transform. A similarity transform $\mathbf{S}$ is composed of a rotation matrix $\mathbf{R}$, a translation vector $\mathbf{t}$, and a uniform scaling s. We formulate this segmentation as a labeling problem, where each label $\ell \in \mathcal{L}$ is associated with a series of similarity transforms over all frames, $\{\mathbf{S}^t_\ell = (\mathbf{R}^t_\ell, \mathbf{t}^t_\ell, s^t_\ell)\}_{t\in(1\ldots T)}$. The output of this step is a spatio-temporal label map, which assigns each pixel of each frame to one of the labels, each of which has an associated similarity transform (Figure 5.2b). We use a fixed number of labels, specified by the user scribbles (Section 5.4.2).

Since the optical flow of real-world videos is often inaccurate at objects boundaries, we achieve more precise segmentations by complementing the per-frame motion models with a color model for each segment. We use a Gaussian Mixture Model (GMM) with 5 Gaussians to represent the color distribution of a label over all frames, with scale, mean and variance parameters denoted as $(\alpha^g_\ell, \mu^g_\ell, \sigma^g_\ell)_{g\in(1\ldots5)}$.

We first discuss how to evaluate the quality of a given configuration of unknowns, before explaining how we find high-quality configurations using an optimization algorithm that iterates between assigning pixels to labels, and updating the motion and color parameters of each label given their assigned pixels. While we describe our algorithm in terms of pixels, we detail at the end of Section 5.4.3 how we accelerate the optimization by working on superpixels.

5.4.1 Energy formulation

In what follows, we denote $I^{t\in T}$ the input video frames and $\mathbf{f}^t$ the optical flow from frame t to the next. Each pixel i in frame t has an initial position $\mathbf{u}^t_i$, such that $\mathbf{u}^{t+1}_i = \mathbf{u}^t_i + \mathbf{f}^t_i$.

We define the quality of a given labeling L with an energy composed of two terms. The first term measures how well the similarity transforms and Gaussian Mixture Model of a label approximate the optical flow and color of pixels assigned to that label. For a given pixel i, frame t and label ℓ, we express the term as

$$\begin{aligned} E_{\text{fit}}(\mathbf{u}_i^t, \ell) &= \mathbf{n}(\mathbf{u}_i^t + \mathbf{f}_i^t) - (\mathbf{R}_\ell^t s_\ell^t \mathbf{u}_i^t + \mathbf{t}_\ell^t) \\ &\quad - w_{\text{color}} \log \big(\sum_g \alpha_\ell^g G(I^t(\mathbf{u}_i), \mu_\ell^g, \sigma_\ell^g) \big), \end{aligned} \tag{5.1}$$

where G denotes the normal distribution and w_{color} balances the contribution of the motion and color models.

The second term encourages large, uniform segments by penalizing the assignment of different labels to neighboring pixels that share similar colors and motion

$$\begin{aligned} E_{\text{smooth}}(\mathbf{u}_i, \mathbf{u}_j, \ell_i, \ell_j) &= \\ \delta(\ell_i \neq \ell_j) &\left[\exp\left(-\frac{D_c(i,j)}{2\beta_c} \right) + \exp\left(-\frac{D_f(i,j)}{2\beta_f} \right) \right], \end{aligned} \tag{5.2}$$

where $D_c(i,j) = \mathbf{n}I(\mathbf{u}_i) - I(\mathbf{u}_j)$ measure the color difference and $D_f(i,j) = \mathbf{n}\mathbf{f}_i - \mathbf{f}_j$ the optical flow difference between pixels i and j. The indicator function $\delta(\ell_i \neq \ell_i)$ equals 1 when the labels ℓ_i and ℓ_j differ, 0 otherwise, and the exponential decreases quickly as the color and motion differences increase. We follow Rother et al. [RKB04] to compute the weights β_c and β_f as the average color and optical flow differences of the video, computed over all pixel neighborhoods. In practice, we evaluate E_{smooth} on a spatio-temporal neighborhood to also encourage temporal coherence of the segmentation, as detailed in Section 4.

We balance these two terms with dedicated weights to obtain the energy of a given labeling L over the entire video sequence

$$\begin{aligned} E_{\text{segment}}(L) &= \sum_i \sum_t w_{\text{fit}} E_{\text{fit}}(\mathbf{u}_i^t, \ell_i^t) \\ &\quad + \sum_{j \in \mathcal{N}_i^t} w_{\text{smooth}} E_{\text{smooth}}(\mathbf{u}_i^t, \mathbf{u}_j, \ell_i^t, \ell_j), \end{aligned} \tag{5.3}$$

where ℓ_i^t denotes the label assigned to pixel i in frame t, and $\mathcal{N}_i^t$ denotes its spatio-temporal neighborhood.

5.4.2 User guidance

The energy formulation outlined above solely measures the quality of a segmentation based on geometric criteria (fitness and smoothness). However, the quality of a segmentation also often depends on artistic goals. For example, users may want to approximate background objects with a single segment, yet decompose a foreground object in several pieces to better capture subtle motions. Similarly, users may choose to segment each leg of an animal separately to prevent one of the legs to appear "fixed" to the body, even if that leg only moves slightly. In addition, the segmentation algorithm can be sensitive to errors in the optical flow or to low-contrast object boundaries. We enable user control and correction by incorporating scribbles in our segmentation algorithm. Each scribble is assigned a color that represents a label, such that pixels scribbled with the same color should end up in the same segment, while pixels scribbled with a different color should be in separate segments. We achieve this behavior by over-writing the fitting term on scribbled pixels

$$E_{\text{fit scribble}}(\mathbf{u}_i^t, \ell) = w_{\text{scribble}} \delta(\ell \neq \ell_s) \tag{5.4}$$

with ℓ_s the label of the scribble. The weight w_{scribble} balances the strength of the user annotations against the other terms of the optimization.

The different scribble colors implicitly define the set of labels $\mathcal{L}$ considered by the optimization. We also experimented with automatic segmentation and a variable number of labels, using a so-called *label cost* to encourage the use of as few labels as possible [DOIB12]. However, we achieved our best results with user guidance. In practice, we only require users to provide scribbles in a few keyframes of their choice, and we propagate these scribbles over the entire video by tracking the scribbled pixels until they get occluded.

5.4.3 Optimization

The energy we defined depends on two sets of variables – the assignment of pixels to labels, ℓ_i^t, and the similarity transforms and Gaussian Mixture Models associated to each label, parameterized by $\mathbf{S}_\ell^t$ and $(\alpha_\ell^g, \mu_\ell^g, \sigma_\ell^g)$ respectively. We solve for values of these variables that approximately minimize $E_{\text{segment}}(L)$ using the PEARL algorithm, which is a general optimization method for multi-model fitting [IB12, DOIB12]. In a nutshell, the algorithm alternates between assigning observations to labels using a fixed set of models, and updating the model parameters of each label to best fit the observations assigned to it (Algorithm 5.1). At each iteration, the assignment of labels is performed using the α-expansion algorithm[1]. We performed 3 such iterations for all results, which

[1] Code for multi-label segmentation available at https://vision.cs.uwaterloo.ca/code

was sufficient to converge in our experiments. The main challenge in applying PEARL in our context is to properly initialize and update the model parameters to capture the complex motion of real-world objects over multiple frames.

Algorithm 5.1: PEARL algorithm [DOIB12] applied to our motion segmentation problem.

```
1 Initialize similarity transforms and GMMs for the set of label candidates $\mathcal{L}$ ;
2 Run $\alpha$-expansion to compute the optimal labeling $L$ according to $E_{\text{segment}}(L)$
    (Equation 5.3), using fixed label candidates $\mathcal{L}$ ;
3 Update the similarity transforms and GMMs of the label candidates $\mathcal{L}$ to best
    fit the optical flow and color distribution within each segment of $L$ ;
4 Goto 2;
```

Initializing the motion and color models. Since each scribble color corresponds to a unique label, we initialize the Gaussian Mixture Model and similarity transforms of each label from its scribbled pixels tracked along the video. Given a set of such scribbled trajectories, we use the least-squares formulation described by Breslav et al. [BSM+07] to fit a similarity transform on the optical flow displacements within each frame, and use the *OpenCV* [Bra00] implementation of Gaussian Mixture Models to fit a color distribution on the colors gathered from all frames. Finally, when the trajectories of the scribbled pixels start after the first frame of the video, or end before the last frame, we initialize the similarity transforms of the missing frames with the transforms obtained at the closest frames.

Updating the motion and color models. Each labeling iteration of the PEARL algorithm forms segments by assigning pixels to labels. Our goal is then to use the optical flow and color values of each segment to update the motion and color models of the corresponding label. However, a given label may only occur in a subset of the video frames; while each label needs a color and motion model in every frame to be used as candidates for the next labeling iteration. Our solution is to extend the segment to other frames by tracking each of its pixel along the forward and backward optical flows. We then update the model parameters using the same least-squares and GMM fitting as for the initialization. Finally, in the event where all pixel trajectories of a segment end before reaching some of the frames, we update the similarity transforms of such frames with the transforms obtained at the closest frames.

Implementation details. In practice, we accelerate the evaluation of $E_{\text{segment}}(L)$ by computing the labeling on a graph of superpixels rather than on the pixel grid. As a downside, working with superpixels reduces temporal coherence of the segmentation since superpixels are computed in each frame independently. We use the average color and optical flow values over superpixels to compute the color and motion difference terms D_c and D_f, and consider two superpixels to be spatial neighbors if they share a boundary, and temporal neighbors if they are connected by at least one optical flow vector. We also introduce a weight on the term E_{smooth} for temporal neighbors according to the number of optical flow connections they share, $w_{\text{temporal}}(i,j) = \frac{\sum_{p\in\mathcal{S}_i}\sum_{q\in\mathcal{S}_j}\delta_{\mathbf{f}}(p,q)}{\min(|\mathcal{S}_i|,|\mathcal{S}_j|)}$, with $\mathcal{S}_i$ and $\mathcal{S}_j$ neighboring superpixels and $\delta_{\mathbf{f}}(p,q)$ equals 1 when pixel p and q are connected by the optical flow, 0 otherwise. Finally, we consider that a superpixel is covered by a scribble if 25% of its pixels are covered by that scribble.

Our implementation is based on the Flownet 2.0 optical flow algorithm [IMS+17] and on SEEDS superpixels [VdBBR+12]. We detect occlusions by checking the consistency of the forward and the backward flow, as described by Sundaram et al. [SBK10] (Equation 5 in their paper).

5.5 Trajectory optimization

Our segmentation algorithm recovers one similarity transform per segment, per frame. However, applying these transforms in sequence results in significant drift, as approximation errors accumulate from frame to frame. The second step of our approach is to optimize pixel trajectories over the video to best reproduce the similarity transforms found at each frame, while keeping pixels close to their original trajectories. As an additional benefit, balancing rigidity of the output with fidelity to the input offers a continuum of solutions, ranging from the original video all the way to a highly rigidified video.

Our approach starts by tracking pixels along the video optical flow to create their trajectories. We adopt a greedy scheme where we start a trajectory for every pixel of the first frame, and then for every pixel of subsequent frames that is not traversed by any existing trajectory. We repeat this process in reverse order, starting from the last frame and progressing towards the first. These two passes over the video provides us with a large set of, sometimes redundant, trajectories. We then order the trajectories by length and select them one by one until all pixels of the video are traversed by at least one trajectory. We end up with N trajectories $\mathbf{U}_{i=1\cdots N}$, each tracking a pixel $\mathbf{u}_i$ over a continuous subset of the frames, *i.e.* $\mathbf{U}_i = (\mathbf{u}_i^{t-m}, ..., \mathbf{u}_i^t, ..., \mathbf{u}_i^{t+n})$. We next optimize for new trajectories $\hat{\mathbf{U}}_i$ according to two energy terms.

The first term measures the deviation of each trajectory from the similarity transforms of the segments it traverses, similar in spirit to as-rigid-as-possible energies used for image and surface deformation [SMW06, SA07]

$$E_{\text{rigid}}(\hat{\mathbf{U}}_i) = \sum_t \mathbf{n}\hat{\mathbf{u}}_i^{t+1} - (\mathbf{R}_\ell^t s_\ell^t \hat{\mathbf{u}}_i^t + \mathbf{t}_\ell^t) \tag{5.5}$$

where the sum runs over all frames of the trajectory, and we use the shorthand $\ell = \ell_i^t$ for clarity.

The second term measures the deviation of each trajectory from its original position

$$E_{\text{anchor}}(\hat{\mathbf{U}}_i) = \sum_t \mathbf{n}\hat{\mathbf{u}}_i^t - \mathbf{u}_i^t. \tag{5.6}$$

Combining the two terms gives an energy over all trajectories

$$\begin{aligned} E_{\text{trajectories}}(\hat{\mathbf{U}}) &= \sum_{i=1\cdots N} w_{\text{rigid}} E_{\text{rigid}}(\hat{\mathbf{U}}_i) \\ &+ w_{\text{anchor}} E_{\text{anchor}}(\hat{\mathbf{U}}_i). \end{aligned} \tag{5.7}$$

After optimization, we generate the output optical flow $\hat{\mathbf{f}}_i^t$ by splatting the vector $(\hat{\mathbf{u}}_i^{t+1} - \hat{\mathbf{u}}_i^t)$ for each pixel along each optimized trajectory. Similarly, we generate a warping field $\mathbf{w}_i^t$ by splatting the vector $(\mathbf{u}_i^t - \hat{\mathbf{u}}_i^t)$, which we will use to render a new video aligned with the optimized optical flow (Section 5.6). Since the optimized trajectories may not traverse all pixels of the output, we diffuse the splatted values to empty pixels.

Implementation details. Equation 5.7 corresponds to a linear least-squares energy, which we minimize by solving the corresponding sparse linear system using Eigen [GJ+10]. Like with the segmentation, we speed-up computation and improve robustness to noise by performing the above optimization over superpixels rather than pixels, where we select trajectories such that each superpixel is traversed by at least one trajectory. However, since this strategy results in a much sparser set of trajectories, diffusing the splatted optical flow and warp vectors produces blurry vector fields. We address this issue by first generating a new segmentation $\hat{L}$, where we assign to each superpixel the most frequent label among the trajectories traversing that superpixel. We then use this segmentation to stop the diffusion at borders between superpixels of different labels. Note that $\hat{L}$ is only a proxy for the segmentation of the output video, since the superpixels are computed on the input rather than on the unknown output. Nevertheless, we found that this approximation improves results compared to using the input segmentation, or no segmentation at all.

5.6 Rendering

We are now equipped with a rigidified optical flow $\hat{\mathbf{f}}_i^t$, along with a warping field $\mathbf{w}_i^t$ that indicates how to distort the input frames to align them with the new flow. Specifically, we render each new frame $\hat{I}^t$ by looking up, for each pixel $\hat{\mathbf{u}}_i^t$, the color of pixel $\hat{\mathbf{u}}_i^t + \mathbf{w}_i^t$ in the original frame I^t.

5.7 Results

We applied our approach on videos with varied motion, including deformable animals and characters (walking cheetah, talking man, walking woman, dancing girl), fluids (waves), and out-of-plane motion (camera rotating around a mountain or following a street). Figure 5.3 shows one frame for each of these sequences, along with user scribbles and the resulting motion segmentation.

Our results demonstrate several different effects, which were produced as a function of the input video and the user scribbles that we provided. For example, we assigned the jaw of the talking man to a different segment than the remaining of his face, which results in a cut-out motion similar to how Canadians are animated in *South Park*. We also purposely separated the legs of the cheetah from its body to achieve a puppet-like animation, or the head of the dancing girl from her torso for a similar effect. Our method also applies to non-articulated objects, such the depth layers of the mountain sequence, or the ground and walls of the walking sequence. In such cases, our method approximates rigid out-of-plane motions by 2D translations and scaling, as is traditionally done in multi-plane cell

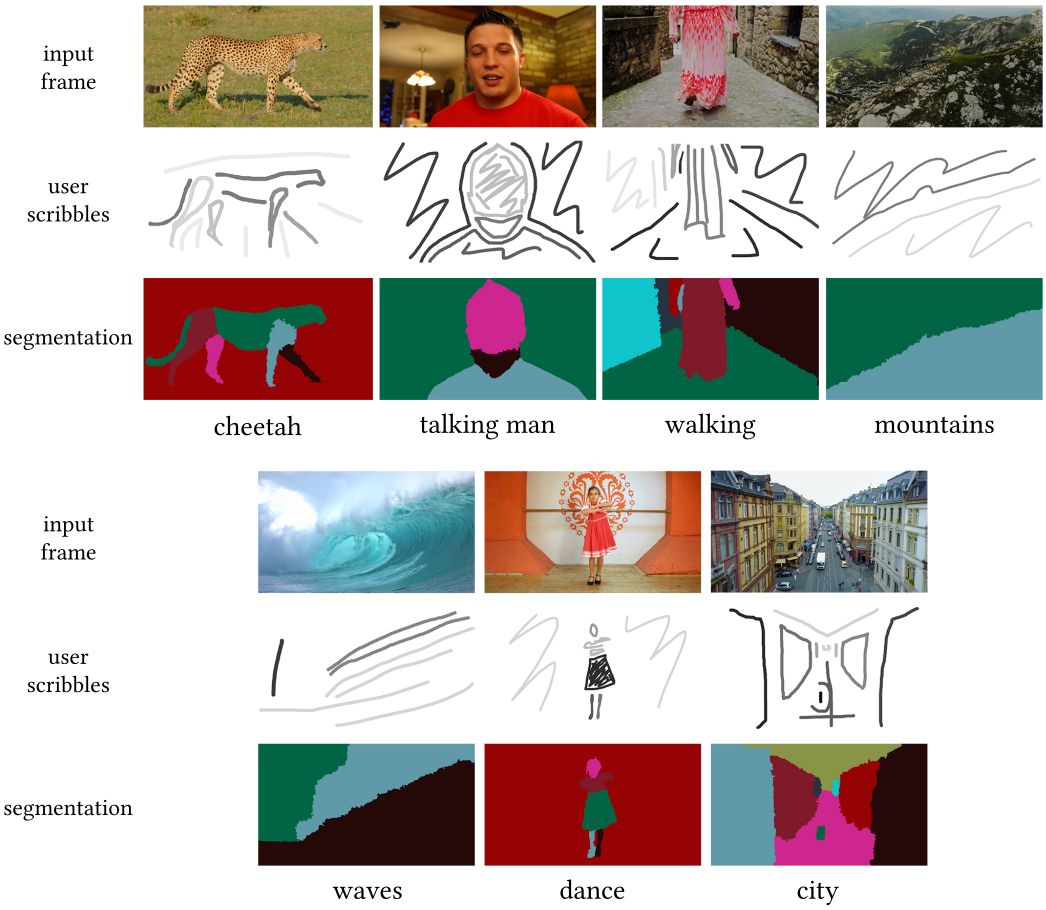


Figure 5.3: Example scribbles and motion segmentation for each sequence in our results.

animation. Finally, the wave sequence illustrates an extreme case of non-rigid motion. Our method approximates the complex motion as a series of simple ones, which results in visible discontinuities at motion borders. These discontinuities can be attenuated by reducing the weight w_{rigid} in Equation 5.7.

Figure 5.4 visualizes the rigidity of an output video by showing how a checkerboard texture evolves as it is advected along its optical flow. Note how the squares of the checkerboard retain their shape in successive frames, while they quickly distort when advected along the original video, revealing the 3D shape of the underlying objects. The only distortions that remain visible in our results occur at disocclusions, where our implementation of texture advection stretches the texture to cover the gaps.

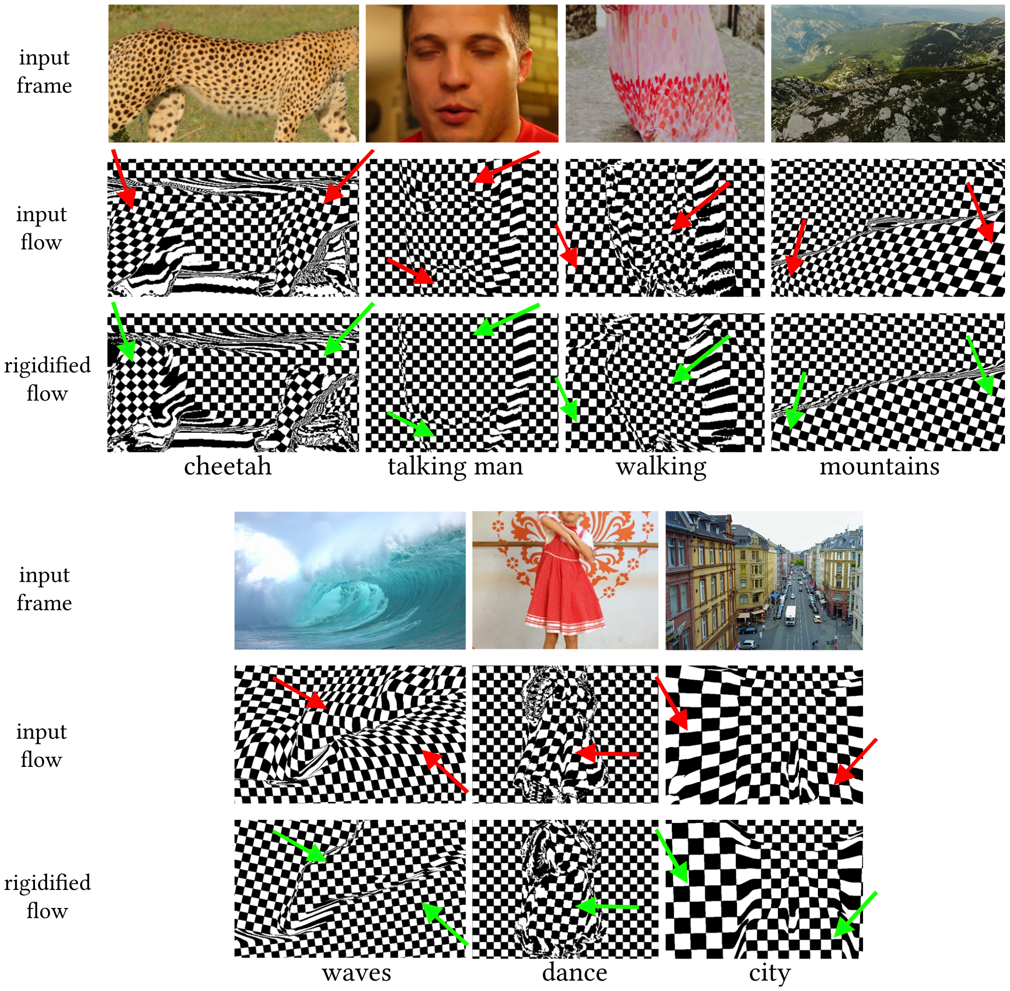


Figure 5.4: Advecting a checkerboard texture along the video quickly reveals distortions due to non-rigid motion (middle row). Our method better preserves the shape of the checkerboard pattern (bottom row). See our supplemental materials for animated versions of this visualization.

We strongly encourage readers to look at our video [2] to judge the effect of our method during animation. In particular, while we provide the intermediate warped videos as supplemental material, the benefit of our approach is best appreciated on side-by-side comparisons between stylizations of original sequences and stylizations of our rigidified versions. For stylization, we use the method by Ruder et al.[3] [RDB18], which incorporates temporal coherence constraints to the successful neural style transfer algorithm of

[2] https://youtu.be/t2Hu58J0gH8

[3] https://github.com/manuelruder/artistic-videos

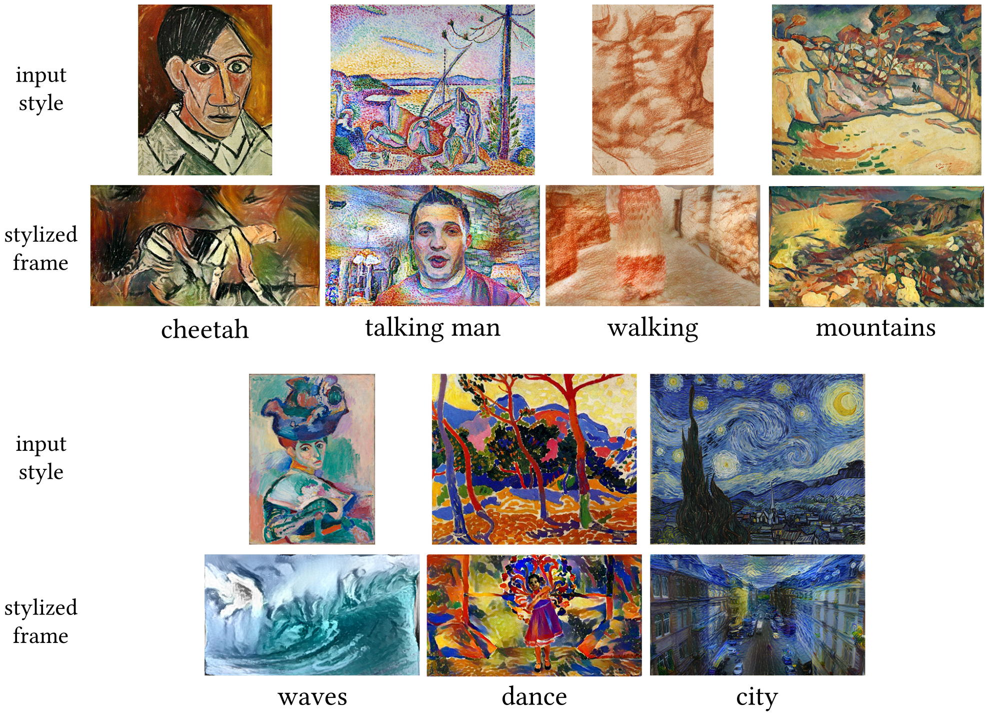


Figure 5.5: We use neural style transfer [GEB16, RDB18] to render videos in various styles. We only transferred the luminance for the *waves* sequence.

Gatys et al. [GEB16]. We used this approach to transfer the style of famous painters, as illustrated in Figure 5.5.

Limitations. Our focus in this work is on the style of motion, rather than on automated video analysis. Our method is sensitive to errors in the input optical flow, which can impact the motion segmentation and optimization. We used extra scribbles to both indicate the desired style and to correct such errors. The optical flow and segmentation algorithms also produce ragged object boundaries, which creates artifacts in the warped video. However, these artifacts are largely hidden in the final stylized result. Given the dizzying pace of advances in computer vision at present, we believe that it should be easy to considerably improve these aspects of our method.

The term E_{anchor} of our trajectory optimization typically results in a warping field of small magnitude, which makes our simple image warp sufficient in most cases. Nevertheless, stretching or fold-over artifacts sometimes occur in areas where two segments move in opposite directions by several pixels, as shown in Figure 5.6. A potential so-

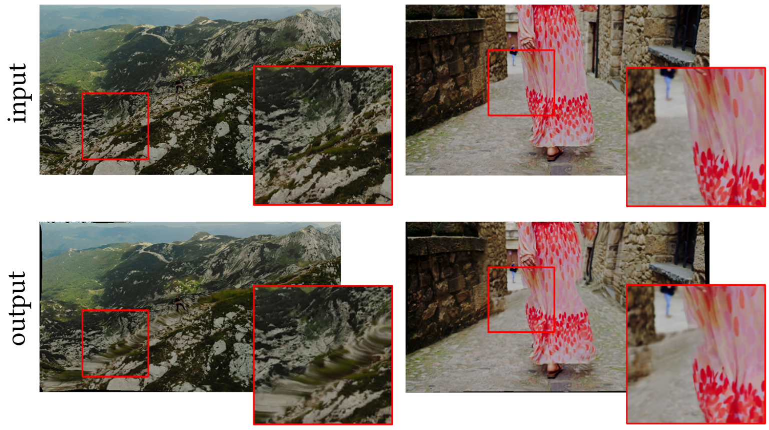


Figure 5.6: Our simple image warp can produce stretching (left) or fold-over artifacts (right) in the presence of strong displacement between neighboring segments. User-provided depth ordering and in-painting would be needed to handle such cases.

lution to this limitation would be to assign a depth order to each segment and in-paint holes that appear between segments or along image borders using texture synthesis, as done by Liu et al. for motion magnification [LTF+05].

Parameter settings and timings. All our videos have a resolution of around 800×450 pixels, which we segmented into 5600 superpixels, each covering around 60 pixels. We kept all parameters fixed for our tests. In particular, we used $w_{\text{color}} = 0.01$ to balance the color and optical flow terms of the segmentation, $w_{\text{fit}} = 80$, $w_{\text{smooth}} = 15$, $w_{\text{scribble}} = 10000$ to treat scribbles as hard constraints, $w_{\text{anchor}} = 1e-6$ and $w_{\text{rigid}} = 1$ to achieve a near rigid output. We also experimented with smaller values of w_{rigid}, but the resulting effects were too small to be noticeable.

Table 5.7 details the time spent for each step of our method, for each of our results on a desktop computer equipped with an Intel Xeon E5-2630 CPU (20 cores) and 48GB of memory . The most expensive part is the motion segmentation, which takes around 4 seconds per frame on average. Significant time is also spent on diffusing the optical flow and warp vectors to all pixels (around 2 seconds par frame), which could be greatly accelerated by using a GPU solver.

Sequence	# frames	# scribbled keyframes	# labels	segmentation (s)	optimization (s)	diffusion (s)
cheetah	230	6	6	1050	57	529
talking	280	7	4	842	45	375
walk	250	4	8	1375	108	866
mountains	200	1	2	423	26	319
dance	200	10	6	937	33	649
waves	180	4	3	589	40	549
street	220	5	9	1438	38	361

Table 5.1: Timings for some of our results. The computational cost of the segmentation is roughly linear with the number of frames and labels, while the remaining steps are linear with the number of frames. Motion segmentation dominates the cost, followed by the diffusion of optical flow and warping vectors.

5.8 Conclusion

In this chapter, we presented a method to simplify the motion of a video, inspired by traditional cut-out animation. We first segment the video into large rigid regions, guided by user scribbles. We then optimize for the position of each pixel so that it follows the rigid motion of its segment while deviating as little as possible from the initial positions. Finally, we warp the input video so that the frames match with the piecewise rigid flow.

The new video and its optical flow can then be fed to any stylization algorithm. The resulting stylized video has a look reminiscent of traditional 2D animation.

Chapter 6

Conclusion

In this last chapter, I begin by summarizing the contributions presented in this thesis and present ideas to complement them, for both sketch based modeling and video stylization. In a third section, I discuss the challenges of designing tools for artists and elaborate on opportunities for future research.

6.1 Sketch based modeling

Research in sketch-based modeling has long been driven by the need for a flexible method capable of reconstructing a large variety of shapes from drawings with minimal user indications. In this thesis we explored the use of deep learning to reach this goal and proposed a simple modeling interface that allows users to seamlessly sketch and visualize shapes in 3D. Our approach is modular and we see multiple directions of future research to improve it.

6.1.1 Predicting shapes with deep volumetric networks

We proposed an architecture capable of predicting 3D volumes from a single drawing, as well as fusing information from multiple drawings via iterative updates. Our system can handle an arbitrary number of views in arbitrary order.

Finer reconstructions may be obtained by training our CNNs with different loss functions. In particular, *adversarial networks* have recently shown an impressive ability to hallucinate fine details in synthesis tasks by combining a generator network with a discriminator that learns to identify if an output is real or synthesized [IZZE17]. Specific network architecture could also be used, such as the ones inspired from octrees, in order to reach a higher resolution output.

Although our iterative process is easy to train, it has the major disadvantage of being sensible to the order of the drawings. In particular, the first drawing has a major impact on the final output shape. A single network that takes all sketches as input and treat them all at once would be more satisfactory. Inspired by SurfaceNet [JGZ+17], we conducted preliminary experiments on a network that takes a single volume as input, carved from all input sketches. However, our early results suggest that 3D convolutions slow down and complexify training. More experiments should thus be done in this direction. Recently, Aittala et al. [AD18] presented a network that takes as input an arbitrary number of aligned images to deblur them. Each image is fed into a network and features of all image are aggregated via a max pooling operation before going through a last series of layers. In our scenario, we could extract 3D features from each sketch, align these feature maps by rotating them to the same viewpoint and aggregate these to get the final shape.

6.1.2 Training with synthetic data

To avoid the collection of 3D objects and drawings, we trained our system with fully synthetic data by generating procedural shapes and rendering them in contours.

Despite its simplicity, our abstract shape grammar proved sufficient to train our system to reconstruct a variety of man-made objects. We hope that this new application will motivate further research in the design of advanced shape grammars that capture the statistics of real-world objects.

We demonstrated the potential of our system by training it with simple contour drawings. Artists often use other visual cues to depict shape in their drawings, such as hatching to convey shading [HZ00], cross-sections to convey curvature directions [SBSS12], scaffolds and vanishing lines to lay down perspective and bounding volumes [SKSK09]. An exciting direction of research would be to train our system to generalize to all these drawing techniques. However, achieving this goal may require the design of new non-photorealistic rendering algorithms that formalize and reproduce such techniques [GSV+17]. Going further, style transfer algorithms [KNBH12b] may even enable the synthesis of user-specific training data.

Moreover, we use in our system only the final line drawing as a bitmap image, but the temporal sequence of lines could also be used to better infer the underlying shape. For example, construction lines and scaffolds provide important clue on the final shape at the very beginning of the sketching process.

Finally, we trained our system with noisy lines to improve its robustness but could not conclude on the effect on the final results. A direction of future research would be to design noise that corresponds better to the errors made by humans and study how it can be used to improve the results. In this thesis, we focused on local noise made when tracing non perfectly straight line, but human drawings usually contains perspective errors at a more global scale. Our experiment also suggests that care should be taken at matching low level features (such as anti-aliasing) of the input lines in the training data.

6.1.3 Fusing different modalities to increase quality

Recent work on sketch-based modeling using deep learning relied either on volumetric or image-based representations of 3D shapes. In this thesis, we showed that we can combine these two representations by using the voxel grid as a support for normal maps fusion. This allows to combine the ability to recover hidden parts via volumetric prediction while the normal maps capture finer details.

Other image-based modalities could be used to improve further the quality of the prediction, for example depth maps that would allow to conserve depth discontinuities. Recent works have also shown that networks can predict a confidence score on their own prediction [LPL+18]. This score could be used in the fusion step to weight the different inputs.

Fundamentally, we see deep learning and hand-crafted optimizations as complementary. We could thus further optimize the resulting shape to enforce regularities such as parallelism, orthogonality and symmetry [LWC+11, XCS+14].

6.2 Video stylization

Despite decades of research in non-photorealistic rendering, motion stylization has received significantly less attention than appearance stylization. As a result, while stylization algorithms can now make individual frames look much like paintings, stylized videos often move too realistically compared to traditional hand-drawn animations. Often, they make the world appear covered in paint. Unfortunately, there are very few reference points in traditional animation to inspire innovative algorithms. Most hand-painted and hand-drawn animations use very simple strategies for creating motion, such as redrawing every frame, or moving paper cut-outs. Hence, to some extent, every non-photorealistic animation algorithm creates a new style of motion.

In this spirit, we explore in this thesis a new style of motion, based on identifying problems with existing motion styles, and taking inspiration from the traditional cut-out animation style as well as from the seminal work by Breslav et al. [BSM$^+$07]. Our output videos produce a strong sense of 2D motion, as if individual parts were moved around in the image plane. Since our method outputs a new video and its optical flow, it is compatible with any existing stylization algorithm.

Our method employs motion segmentation to decompose the video into near-rigid parts, and we found that different segmentations of the same video can result in very different stylizations, which motivated us to provide user control on this step. In the future, we believe that a combination of better computer vision algorithms and new design principles inspired by real-world animations could help automate motion simplification. For example, the Cartoon Animation Filter [WDAC06] aims at mimicking cartoon specific features such as squash and stretch but other styles of motions could be considered.

Moreover, we think that motion and appearance could be stylized jointly to get a more convincing result. In our case, we could use the segmentation and the associated rigid motions to guide the stylization step, for example stylizing each segment individually to give a cartoon effect [WXSC04] or using the rigid motion to guide hatching or brush directions.

6.3 Future work on digital tools for artists

My explorations during this thesis have led me to distinguish four main criteria that digital creative tools should fulfill:

Meaningful task The scope of the task needs to be well defined relatively to the workflow of the artist: placed at the right moment and with the right inputs and outputs. Moreover, the gain for the artists needs to match the investment of using the tool (learning to use it, adapt their way of working). This also means that the task should be fulfilled with a good enough quality so that the artists doesn't have to manually correct it.

Ease of use The tools should be intuitive to use for artists so that they can easily and quickly experiment with it. The tools should thus adapt to the way artists work more than the opposite.

Room for creativity The artistic intent should always come from the artist and not from the tool. Fully automatic tools are thus not necessarily a good solution, as the artists should be able to control the process. The tools should facilitate parts that can be automated but leave the control to the user each time it is needed so that they can express their creativity. For professional artists, I would even argue that they rarely need tools to replace their final gesture, but more to help them in the preparatory steps where it is crucial to give them a large amount of freedom to explore.

Pleasant result The resulting creation should be similar to the result the artist would obtain manually.

I believe that satisfying these criteria requires to collaborate with artists all along the process.

A tight collaboration at the beginning of the project is necessary to analyze the needs of artists and understand how they are working. This helps to think about what is the task in which they need help, how the artists will use the tool and what is the creative part that should be conserved. It is also important to keep artist's inputs coming all along the project up till the end of it in order to evaluate how the project fulfills these criteria.

This opens interesting challenges I would like to tackle. In particular, I believe that creating intuitive tools requires a tight integration between user interfaces and computer graphics. As stated in the introduction of this thesis, computer vision and computer graphics are essential to **interpret** the intent of the user correctly and **generate** visual content that corresponds to it. But work is also needed on user interaction to allow artists to **express** their intent intuitively. The design of both the interface and the computer graphic tools should thus be done jointly so that they form a consistent system and lean on each other.

In the case of our sketch based modeling system, we designed the interface so that it helps the system to interpret user's sketches. The interface guide the user to draw in accordance to the training data so that we get better results.

Playful Palette [SLD17] is also a good example of such co-design. The goal was to bring a new way of managing and choosing colors when painting digitally, inspired by traditional physical palette. Although this project emphasizes on the interface design, computer graphics were necessary to model the digital palette as a mix of color blobs and thus to generate the palette interface. This paper is a perfect mix between interaction and computer graphics problem with an artist-centered approach. The authors first carefully study how artists use a physical palette to extract design principles for their digital version of it and then had the final system tested by artists as an evaluation.

As a conclusion, the creation of digital tools for artists is a broad field that can be applied to various artistic domains: design, 2D animation, 3D modeling, painting, photography or video manipulations. Each of these domains brings new challenges and infinite possibilities of helping the creation process of artists. While a large variety of methods can be used to solve these problems - the recent progress of deep learning techniques bringing interesting new ways of solving them -, it is crucial to always work in collaboration with artists and put them at the center of these research problems.

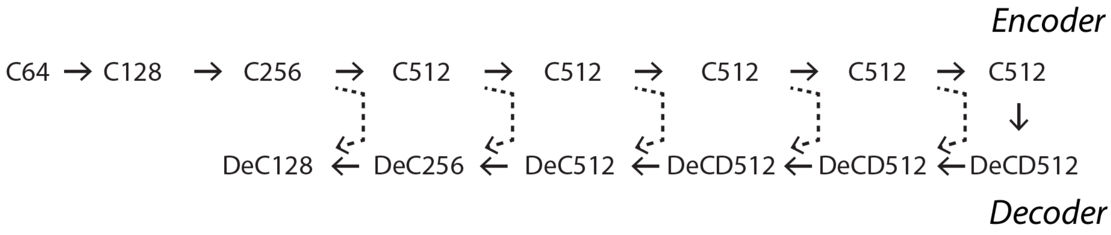
Appendix A

Architecture of the volumetric network

We adapt our architecture from pix2pix [IZZE17] by reducing the number of layers of the decoder part.

Let $(De)Ck$ denote a (De)Convolution-BatchNorm-ReLU layer with k filters (outputs a layer with k channels). $(De)CDk$ denotes a (De)Convolution-BatchNorm- Dropout-ReLU layer with a dropout rate of 50%. All convolutions are 4×4 spatial filters applied with stride 2. Convolutions in the encoder downsample by a factor of 2, whereas deconvolutions in the decoder upsample by a factor of 2.

The encoder-decoder architecture consists of the following layers:



After the last layer of the decoder, a SoftMax is applied followed by a classification loss. We then keep only one channel over two (the one containing the probability of occupancy) to get a voxel grid of dimension 64.

As an exception to the above notation, Batch- Norm is not applied to the first C64 layer in the encoder. All ReLUs in the encoder are leaky, with slope 0.2, while ReLUs in the decoder are not leaky.

The skip connections (shown as dashed arrows) consist in concatenating the output of a convolution in the encoder to a deconvolution of the same size in the decoder.

Bibliography

[AD18] Miika Aittala and Frédo Durand. Burst image deblurring using permutation invariant convolutional neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 731–747, 2018.

[ADN+17] Rahul Arora, Ishan Darolia, Vinay P Namboodiri, Karan Singh, and Adrien Bousseau. Sketchsoup: Exploratory ideation using design sketches. In *Computer Graphics Forum*, volume 36, pages 302–312. Wiley Online Library, 2017.

[BBL+17] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[BBRF14] Mark Browning, Connelly Barnes, Samantha Ritter, and Adam Finkelstein. Stylized keyframe animation of fluid simulations. In *Symposium on Non-Photorealistic Animation and Rendering - NPAR '14*, 2014.

[BBS08] Seok-Hyung Bae, Ravin Balakrishnan, and Karan Singh. Ilovesketch: as-natural-as-possible sketching system for creating 3d curve models. In *ACM symposium on User Interface Software and Technology (UIST)*, pages 151–160, 2008.

[BBT11] Pierre Bénard, Adrien Bousseau, and Joëlle Thollot. State-of-the-art report on temporal coherence for stylized animations. *Computer Graphics Forum*, 30(8):2367–2386, December 2011.

[BCK+13] Pierre Bénard, Forrester Cole, Michael Kass, Igor Mordatch, James Hegarty, Martin Sebastian Senn, Kurt Fleischer, Davide Pesare, and Katherine Breeden. Stylizing Animation By Example. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 32(4):119:1–119:12, July 2013.

[BH18] Pierre Bénard and Aaron Hertzmann. Line drawings from 3d models. *arXiv preprint arXiv:1810.01175*, 2018.

[BJS+08] Connelly Barnes, David E. Jacobs, Jason Sanders, Dan B Goldman, Szymon Rusinkiewicz, Adam Finkelstein, and Maneesh Agrawala. Video puppetry: A performative interface for cutout animation. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*, 27(5):124:1–124:9, December 2008.

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

[Bra00] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[BSM+07] Simon Breslav, Karol Szerszen, Lee Markosian, Pascal Barla, and Joëlle Thollot. Dynamic 2D Patterns for Shading 3D Scenes. *ACM Transaction on Graphics (Proc. SIGGRAPH)*, 26(3), 2007.

[BT81] H. Barrow and J. Tenenbaum. Interpreting line drawings as three-dimensional surfaces. *Artificial Intelligence*, 17, 1981.

[CAFM16] Joao Carreira, Pulkit Agrawal, Katerina Fragkiadaki, and Jitendra Malik. Human pose estimation with iterative error feedback. 2016.

[CFG+15] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical report, 2015.

[CFGL16] David Coeurjolly, Marion Foare, Pierre Gueth, and Jacques-Olivier Lachaud. Piecewise smooth reconstruction of normal vector field on digital data. In *Computer Graphics Forum*, volume 35, pages 157–167. Wiley Online Library, 2016.

[CGL+08] Forrester Cole, Aleksey Golovinskiy, Alex Limpaecher, Heather Stoddart Barros, Adam Finkelstein, Thomas Funkhouser, and Szymon Rusinkiewicz. Where do people draw lines? In *ACM Transactions on Graphics (Proc. SIGGRAPH)*, volume 27, page 88. ACM, 2008.

[CGL17] David Coeurjolly, Pierre Gueth, and Jacques-Olivier Lachaud. Digital surface regularization by normal vector field alignment. In *International Conference on Discrete Geometry for Computer Imagery*, pages 197–209. Springer, 2017.

[CGS16] Loïc Ciccone, Martin Guay, and Robert Sumner. Flow curves: an intuitive interface for coherent scene deformation. In *Computer Graphics Forum*, volume 35, pages 247–256. Wiley Online Library, 2016.

[CIF$^+$12] Forrester Cole, Phillip Isola, William T Freeman, Frédo Durand, and Edward H Adelson. Shapecollage: occlusion-aware, example-based shape interpretation. In *European Conference on Computer Vision (ECCV)*, pages 665–678. Springer, 2012.

[CK17] Qifeng Chen and Vladlen Koltun. Photographic image synthesis with cascaded refinement networks. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[CRH05] J. P. Collomosse, D. Rowntree, and P. M. Hall. Rendering cartoon-style motion cues in post-production video. *Graphical Models*, 67(6):549–564, 2005.

[CSGC16] Frederic Cordier, Karan Singh, Yotam Gingold, and Marie-Paule Cani. Sketch-based modeling. In *SIGGRAPH ASIA Courses*, pages 18:1–18:222. ACM, 2016.

[CSMS13] Frederic Cordier, Hyewon Seo, Mahmoud Melkemi, and Nickolas S. Sapidis. Inferring mirror symmetric 3d shapes from sketches. *Computer Aided Design*, 45(2):301–311, February 2013.

[CXG$^+$16] Christopher Bongsoo Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *European Conference on Computer Vision (ECCV)*, pages 628–644, 2016.

[CZS$^+$13] Tao Chen, Zhe Zhu, Ariel Shamir, Shi-Min Hu, and Daniel Cohen-Or. 3-sweep: Extracting editable objects from a single photo. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 32(6), 2013.

[DAD$^+$18] Valentin Deschaintre, Miika Aittala, Fredo Durand, George Drettakis, and Adrien Bousseau. Single-image svbrdf capture with a rendering-aware deep network. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 37(4):128, 2018.

[DAI$^+$18] Johanna Delanoy, Mathieu Aubry, Phillip Isola, Alexei A Efros, and Adrien Bousseau. 3d sketching using multi-view deep volumetric prediction. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 1(1):21, 2018.

[DBB$^+$17] Marek Dvorožňák, Pierre Bénard, Pascal Barla, Oliver Wang, and Daniel Sỳkora. Example-based expressive animation of 2d rigid bodies. *ACM Transactions on Graphics (TOG)*, 36(4):127, 2017.

[DLKS18a] Marek Dvorožňák, Wilmot Li, Vladimir G. Kim, and Daniel Sýkora. ToonSynth: Example-based synthesis of hand-colored cartoon animations. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 37(4), 2018.

[DLKS18b] Marek Dvorožnák, Wilmot Li, Vladimir G Kim, and Daniel Sỳkora. Toonsynth: example-based synthesis of hand-colored cartoon animations. *ACM Transactions on Graphics (TOG)*, 37(4):167, 2018.

[DOIB12] Andrew Delong, Anton Osokin, Hossam Isack, and Yuri Boykov. Fast Approximate Energy Minization with Label Costs. *International Journal of Computer Vision (IJCV)*, 96(1):1–27, 2012.

[DSTB16] A. Dosovitskiy, J. T. Springenberg, M. Tatarchenko, and T. Brox. Learning to generate chairs, tables and cars with convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2016.

[EF15] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2650–2658, 2015.

[EHA12] Mathias Eitz, James Hays, and Marc Alexa. How do humans sketch objects? *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 31(4):44:1–44:10, 2012.

[ERB$^+$12] Mathias Eitz, Ronald Richter, Tamy Boubekeur, Kristian Hildebrand, and Marc Alexa. Sketch-based shape retrieval. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 31(4):31:1–31:10, 2012.

[ES11] Koos Eissen and Roselien Steur. *Sketching: The Basics.* Bis Publishers, 2011.

[FJL$^+$16] Jakub Fišer, Ondřej Jamriška, Michal Lukáč, Eli Shechtman, Paul Asente, Jingwan Lu, and Daniel Sýkora. StyLit: Illumination-guided example-based stylization of 3d renderings. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 35(4), 2016.

[FJS$^+$17] Jakub Fišer, Ondřej Jamriška, David Simons, Eli Shechtman, Jingwan Lu, Paul Asente, Michal Lukáč, and Daniel Sýkora. Example-based synthesis of stylized facial animations. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 36(4), 2017.

[FLB16] Jean-Dominique Favreau, Florent Lafarge, and Adrien Bousseau. Fidelity vs. simplicity: a global approach to line drawing vectorization. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 2016.

[FLJ$^+$14] J. Fišer, M. Lukáč, O. Jamriška, M. Čadík, Y. Gingold, P. Asente, and D. Sýkora. Color me noisy: Example-based rendering of hand-colored animations with temporal noise control. *Computer Graphics Forum (Proc. EGSR)*, 33, 2014.

[FMK$^+$03a] Thomas Funkhouser, Patrick Min, Michael Kazhdan, Joyce Chen, Alex Halderman, David Dobkin, and David Jacobs. A search engine for 3d models. *ACM Transactions on Graphics (TOG)*, 22(1):83–105, 2003.

[FMK$^+$03b] Thomas Funkhouser, Patrick Min, Michael Kazhdan, Joyce Chen, Alex Halderman, David Dobkin, and David Jacobs. A search engine for 3d models. *ACM Transactions on Graphics*, 22(1):83–105, January 2003.

[FSG17] Haoqiang Fan, Hao Su, and Leonidas Guibas. A point set generation network for 3d object reconstruction from a single image. *IEEE Computer Vision and Pattern Recognition (CVPR)*, 2017.

[GCR13] Martin Guay, Marie-Paule Cani, and Rémi Ronfard. The line of action: an intuitive interface for expressive character posing. *ACM Transactions on Graphics (TOG)*, 32(6):205, 2013.

[GEB16] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[GFK$^+$18] Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. Atlasnet: A papier-mâché approach to learning 3d surface generation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 216–224, 2018.

[GIZ09] Yotam Gingold, Takeo Igarashi, and Denis Zorin. Structured annotations for 2D-to-3D modeling. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*, 28(5), 2009.

[GJ$^+$10] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[Goo16] Google. Tilt brush. https://www.tiltbrush.com/, 2016.

[GSV$^+$17] Giorgio Gori, Alla Sheffer, Nicholas Vining, Enrique Rosales, Nathan Carr, and Tao Ju. Flowrep: Descriptive curve networks for free-form design shapes. *ACM Transaction on Graphics (Proc. SIGGRAPH)*, 36(4), 2017.

[GTDS04] Stéphane Grabli, Emmanuel Turquin, Frédo Durand, and François X Sillion. Programmable style for npr line drawing. In *Rendering Techniques 2004 (Eurographics Symposium on Rendering)*. ACM Press, 2004.

[GTDS10] Stéphane Grabli, Emmanuel Turquin, Frédo Durand, and François X. Sillion. Programmable rendering of line drawing from 3d scenes. *ACM Transactions on Graphics*, 29(2):18:1–18:20, 2010.

[HE04] James Hays and Irfan A. Essa. Image and video based painterly animation. In *International Symposium on Nonphotorealistic Animation and Rendering (NPAR)*, 2004.

[HGY17] X. Han, C. Gao, and Y. Yu. Deepsketch2face: A deep learning based sketching system for 3d face and caricature modeling. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 36(4), July 2017.

[HKYM16] Haibin Huang, Evangelos Kalogerakis, Ersin Yumer, and Radomir Mech. Shape synthesis from sketches via procedural models and convolutional networks. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 22(10):1, 2016.

[HLW$^+$17] James W Hennessey, Han Liu, Holger Winnemöller, Mira Dontcheva, and Niloy J Mitra. How2sketch. In *Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games-I3D'17*. ACM Press, 2017.

[HMC$^+$15] Fabian Hahn, Frederik Mutzel, Stelian Coros, Bernhard Thomaszewski, Maurizio Nitti, Markus Gross, and Robert W Sumner. Sketch abstractions for character posing. In *Proceedings of the 14th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 185–191. ACM, 2015.

[HP00] Aaron Hertzmann and Ken Perlin. Painterly rendering for video and interaction. In *International Symposium on Nonphotorealistic Animation and Rendering (NPAR)*, 2000.

[HTM17] Christian Häne, Shubham Tulsiani, and Jitendra Malik. Hierarchical surface prediction for 3d object reconstruction. In *IEEE International Conference on 3D Vision (3DV)*, pages 412–420, 2017.

[HZ00] Aaron Hertzmann and Denis Zorin. Illustrating smooth surfaces. In *SIGGRAPH*, pages 517–526. ACM Press/Addison-Wesley Publishing Co., 2000.

[IB12] Hossam Isack and Yuri Boykov. Energy-based geometric multi-model fitting. *International Journal of Computer Vision*, 97:123–147, April 2012.

[IBB15] Emmanuel Iarussi, David Bommes, and Adrien Bousseau. Bendfields: Regularized curvature fields from rough concept sketches. *ACM Transactions on Graphics*, 2015.

[IBT13] Emmanuel Iarussi, Adrien Bousseau, and Theophanis Tsandilas. The drawing assistant: Automated drawing guidance and feedback from photographs. In *ACM Symposium on User Interface Software and Technology (UIST)*. ACM, 2013.

[IMS$^+$17] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[IMT99] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: A sketching interface for 3d freeform design. In *SIGGRAPH*, pages 409–416, 1999.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of The 32nd International Conference on Machine Learning*, pages 448–456, 2015.

[IZZE17] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *IEEE Computer Vision and Pattern Recognition (CVPR)*, 2017.

[JDD03] Thouis R. Jones, Frédo Durand, and Mathieu Desbrun. Non-iterative, feature-preserving mesh smoothing. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 2003.

[JGZ$^+$17] Mengqi Ji, Juergen Gall, Haitian Zheng, Yebin Liu, and Lu Fang. Surfacenet: An end-to-end 3d neural network for multiview stereopsis. In *IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[JHR$^+$15] Amaury Jung, Stefanie Hahmann, Damien Rohmer, Antoine Begault, Laurence Boissieux, and Marie-Paule Cani. Sketching Folds: Developable Surfaces from Non-Planar Silhouettes. *ACM Transactions on Graphics*, 2015.

[JSD$^+$14] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the*

22nd ACM international conference on Multimedia, pages 675–678. ACM, 2014.

[KB14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[KCWI13] J. E. Kyprianidis, J. Collomosse, Tinghuai Wang, and T. Isenberg. State of the art: A taxonomy of artistic stylization techniques for images and video. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 19(5):866–885, 2013.

[KG17] Alex Kendall and Yarin Gal. What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision? In *Advances in Neural Information Processing Systems 30 (NIPS)*, 2017.

[KGC$^+$17] Rubaiat Habib Kazi, Tovi Grossman, Hyunmin Cheong, Ali Hashemi, and George W Fitzmaurice. Dreamsketch: Early stage 3d design explorations with sketching and generative design. In *UIST*, pages 401–414, 2017.

[KNBH12a] Evangelos Kalogerakis, Derek Nowrouzezahrai, Simon Breslav, and Aaron Hertzmann. Learning Hatching for Pen-and-Ink Illustration of Surfaces. *ACM Transactions on Graphics*, 31(1), 2012.

[KNBH12b] Evangelos Kalogerakis, Derek Nowrouzezahrai, Simon Breslav, and Aaron Hertzmann. Learning Hatching for Pen-and-Ink Illustration of Surfaces. *ACM Transactions on Graphics*, 31(1), 2012.

[KP11] Michael Kass and Davide Pesare. Coherent noise for non-photorealistic rendering. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 30(4):30:1–30:6, July 2011.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems (NIPS)*, pages 1097–1105, 2012.

[LC87] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH*, 21(4):163–169, August 1987.

[LF08] Jeehyung Lee and Thomas Funkhouser. Sketch-based search and composition of 3D models. In *EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling (SBIM)*, June 2008.

[LGK$^+$17] Zhaoliang Lun, Matheus Gadelha, Evangelos Kalogerakis, Subhransu Maji, and Rui Wang. 3d shape reconstruction from sketches via multiview convolutional networks. In *International Conference on 3D Vision (3DV)*, 2017.

[Lit97] Peter Litwinowicz. Processing images and video for an impressionist effect. In *SIGGRAPH*, pages 407–414, 1997.

[LLW17] Chengze Li, Xueting Liu, and Tien-Tsin Wong. Deep extraction of manga structural lines. *ACM Transactions on Graphics (TOG)*, 36(4):117, 2017.

[LPL$^+$18] Changjian Li, Hao Pan, Yang Liu, Xin Tong, Alla Sheffer, and Wenping Wang. Robust flow-guided neural prediction for sketch-based freeform surface modeling. In *ACM Transaction on Graphics (Proc. SIGGRAPH Asia)*, page 238. ACM, 2018.

[LRS18] Chenxi Liu, Enrique Rosales, and Alla Sheffer. Strokeaggregator: consolidating raw sketches into artist-intended curve drawings. *ACM Transactions on Graphics (TOG)*, 37(4):97, 2018.

[LS96] H Lipson and M Shpitalni. Optimization-based reconstruction of a 3d object from a single freehand line drawing. *Computer-Aided Design*, 28(8):651–663, 1996.

[LS00] H. Lipson and M. Shpitalni. Conceptual design and analysis by sketching. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 14(5):391–401, November 2000.

[LSD15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *IEEE Computer Vision and Pattern Recognition (CVPR)*, 2015.

[LTF$^+$05] Ce Liu, Antonio Torralba, William T. Freeman, Frédo Durand, and Edward H. Adelson. Motion magnification. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 24(3), 2005.

[LWC+11] Yangyan Li, Xiaokun Wu, Yiorgos Chrysanthou, Andrei Sharf, Daniel Cohen-Or, and Niloy J. Mitra. Globfit: Consistently fitting primitives by discovering global relations. *ACM Transactions on Graphics (proc. SIGGRAPH)*, 30(4), 2011.

[LXC+17] Jun Li, Kai Xu, Siddhartha Chaudhuri, Ersin Yumer, Hao Zhang, and Leonidas Guibas. Grass: Generative recursive autoencoders for shape structures. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 36(4), 2017.

[LYF17] J. Liu, F. Yu, and T. Funkhouser. Interactive 3D Modeling with a Generative Adversarial Network. In *International Conference on 3D Vision (3DV)*. 2017.

[LYKL12] Sun-Young Lee, Jong-Chul Yoon, Ji-Yong Kwon, and In-Kwon Lee. Cartoonmodes: Cartoon stylization of video objects through modal analysis. *Graphical Models*, 74(2):51 – 60, 2012.

[LZC11] Yong Jae Lee, C Lawrence Zitnick, and Michael F Cohen. Shadowdraw: real-time user guidance for freehand drawing. In *ACM Transactions on Graphics (TOG)*, volume 30, page 27. ACM, 2011.

[MA83] Worthy N. Martin and J. K. Aggarwal. Volumetric descriptions of objects from multiple views. *IEEE Transactions on Pattern Analysis Machine Intelligence*, 5(2):150–158, February 1983.

[Mei96] Barbara J. Meier. Painterly rendering for animation. In *SIGGRAPH*, pages 477–484, 1996.

[Min16] Patrick Min. Binvox 3d mesh voxelizer. `http://www.google.com/search?q=binvox`, 2016. Accessed: 2016-11-01.

[MM89] J. Malik and D. Maydan. Recovering three-dimensional shape from a single image of curved objects. *IEEE Pattern Analysis and Machine Intelligence (PAMI)*, 11(6):555–566, 1989.

[MSS+18] Santiago E Montesdeoca, Hock Soon Seah, Amir Semmo, Pierre Bénard, Romain Vergne, Joëlle Thollot, and Davide Benvenuti. Mnpr: a framework for real-time expressive non-photorealistic rendering of 3d computer graphics. In *Proceedings of the Joint Symposium on Computational Aesthetics and Sketch-Based Interfaces and Modeling and Non-Photorealistic Animation and Rendering*, page 9. ACM, 2018.

[MVH$^+$17] Nicolas Mellado, David Vanderhaeghe, Charlotte Hoarau, Sidonie Christophe, Mathieu Brédif, and Loïc Barthe. Constrained palette-space exploration. *ACM Transactions on Graphics (TOG)*, 36(4):60, 2017.

[NDY$^+$16] A. Nguyen, A. Dosovitskiy, J. Yosinski, T. Brox, and J. Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[NGDGA$^+$16] Gen Nishida, Ignacio Garcia-Dorado, Daniel G. Aliaga, Bedrich Benes, and Adrien Bousseau. Interactive sketching of urban procedural models. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 2016.

[NISA07a] Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa. Fibermesh: designing freeform surfaces with 3d curves. In *ACM transactions on graphics (TOG)*, volume 26, page 41. ACM, 2007.

[NISA07b] Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa. Fibermesh: designing freeform surfaces with 3d curves. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 26, 2007.

[NT03] F. S. Nooruddin and G. Turk. Simplification and repair of polygonal models using volumetric techniques. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 9(2):191–205, April 2003.

[OH12] Peter O'Donovan and Aaron Hertzmann. Anipaint: Interactive painterly animation from video. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 18(3):475–487, 2012.

[OK12] Gunay Orbay and Levent Burak Kara. Sketch-based surface design using malleable curve networks. *Computers & Graphics*, 36(8):916–929, 2012.

[OSSJ09] L. Olsen, F.F. Samavati, M.C. Sousa, and J. Jorge. Sketch-based modeling: A survey. *Computers & Graphics*, 33, 2009.

[OWL15] Markus Oberweger, Paul Wohlhart, and Vincent Lepetit. Training a feedback loop for hand pose estimation. In *IEEE International Conference on Computer Vision (ICCV)*, pages 3316–3324, 2015.

[Pip07] Alan Pipes. *Drawing for Designers*. Laurence King, 2007.

[PYY$^+$17] Eunbyung Park, Jimei Yang, Ersin Yumer, Duygu Ceylan, and Alexander C. Berg. Transformation-grounded image generation network for novel 3d view synthesis. *IEEE Computer Vision and Pattern Recognition (CVPR)*, 2017.

[RC13] Paul Rosin and John Collomosse. *Image and Video-Based Artistic Stylisation*. Springer, 2013.

[RDB18] M. Ruder, A. Dosovitskiy, and T. Brox. Artistic style transfer for videos and spherical images. *International Journal of Computer Vision*, 2018.

[RDI10] Alec Rivers, Frédo Durand, and Takeo Igarashi. 3d modeling with silhouettes. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 29(4):109:1–109:8, 2010.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention - MICCAI*, pages 234–241, 2015.

[RKB04] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. "grabcut": Interactive foreground extraction using iterated graph cuts. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 23(3):309–314, 2004.

[RUBG17] Gernot Riegler, Ali Osman Ulusoy, Horst Bischof, and Andreas Geiger. Octnetfusion: Learning depth fusion from data. In *International Conference on 3D Vision (3DV)*. 2017.

[RVRK16] Stephan R Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In *European Conference on Computer Vision*, pages 102–118. Springer, 2016.

[SA07] Olga Sorkine and Marc Alexa. As-rigid-as-possible surface modeling. In *Proc. EUROGRAPHICS/ACM SIGGRAPH Symposium on Geometry Processing*, pages 109–116, 2007.

[SAG$^+$13] Alex Shtof, Alexander Agathos, Yotam Gingold, Ariel Shamir, and Daniel Cohen-Or. Geosemantic snapping for sketch-based modeling. *Computer Graphics Forum*, 32(2):245–253, 2013.

[SBHH16] Patsorn Sangkloy, Nathan Burnell, Cusuh Ham, and James Hays. The sketchy database: Learning to retrieve badly drawn bunnies. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 2016.

[SBK10] N. Sundaram, T. Brox, and K. Keutzer. Dense point trajectories by gpu-accelerated large displacement optical flow. In *European Conference on Computer Vision (ECCV)*, Lecture Notes in Computer Science. Springer, Sept. 2010.

[SBSS12] Cloud Shao, Adrien Bousseau, Alla Sheffer, and Karan Singh. Crossshade: shading concept sketches using cross-section curves. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 31(4), 2012.

[SDY$^+$18] Wanchao Su, Dong Du, Xin Yang, Shizhe Zhou, and Hongbo Fu. Interactive sketch-based normal map generation with deep neural networks. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 1(1), 2018.

[SED16] Ahmed Selim, Mohamed Elgharib, and Linda Doyle. Painting style transfer for head portraits using convolutional neural networks. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 35(4):129:1–129:18, July 2016.

[SIJ$^+$07] Ryan Schmidt, Tobias Isenberg, Pauline Jepp, Karan Singh, and Brian Wyvill. Sketching, scaffolding, and inking: a visual history for interactive 3d modeling. In *Proceedings of the 5th international symposium on Non-photorealistic animation and rendering*, pages 23–32. ACM, 2007.

[SKSK09] Ryan Schmidt, Azam Khan, Karan Singh, and Gord Kurtenbach. Analytic drawing of 3d scaffolds. In *ACM Transactions on Graphics (TOG)*, volume 28, page 149. ACM, 2009.

[SKv$^+$14] Daniel Sýkora, Ladislav Kavan, Martin Čadík, Ondřej Jamriška, Alec Jacobson, Brian Whited, Maryann Simmons, and Olga Sorkine-Hornung. Ink-and-ray: Bas-relief meshes for adding global illumination effects to hand-drawn characters. *ACM Transactions on Graphics*, 33, 2014.

[SLD17] Maria Shugrina, Jingwan Lu, and Stephen Diverdi. Playful palette: an interactive parametric color mixer for artists. *ACM Transactions on Graphics (TOG)*, 36(4):61, 2017.

[SLF$^+$17] Patsorn Sangkloy, Jingwan Lu, Chen Fang, FIsher Yu, and James Hays. Scribbler: Controlling deep image synthesis with sketch and color. *IEEE Computer Vision and Pattern Recognition (CVPR)*, 2017.

[SLKD16] Amir Semmo, Daniel Limberger, Jan Eric Kyprianidis, and Jürgen Döllner. Image stylization by interactive oil paint filtering. *Computers & Graphics*, 55:157–171, 2016.

[SLZ$^+$13] Tianjia Shao, Wilmot Li, Kun Zhou, Weiwei Xu, Baining Guo, and Niloy J Mitra. Interpreting concept sketches. *ACM Transactions on Graphics (TOG)*, 32(4):56, 2013.

[SMW06] Scott Schaefer, Travis McPhail, and Joe Warren. Image deformation using moving least squares. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 25(3):533, 2006.

[SQLG15] Hao Su, Charles R Qi, Yangyan Li, and Leonidas J Guibas. Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2686–2694, 2015.

[SSISI16] Edgar Simo-Serra, Satoshi Iizuka, Kazuma Sasaki, and Hiroshi Ishikawa. Learning to Simplify: Fully Convolutional Networks for Rough Sketch Cleanup. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 35(4), 2016.

[SSM$^+$16] Suraj Srinivas, Ravi Kiran Sarvadevabhatla, Konda Reddy Mopuri, Nikita Prabhu, Srinivas S. S. Kruthiventi, and R. Venkatesh Babu. A taxonomy of deep convolutional neural nets for computer vision. volume 2, page 36. 2016.

[ST90] Takafumi Saito and Tokiichiro Takahashi. Comprehensible rendering of 3-d shapes. *SIGGRAPH*, 24(4):197–206, 1990.

[Sut64] Ivan E Sutherland. Sketchpad a man-machine graphical communication system. *Simulation*, 2(5):R–3, 1964.

[Sys11] Dassault Systèmes. Catia natural sketch. https://www.3ds.com/fr/produits-et-services/catia/, 2011.

[TBvdP04] Matthew Thorne, David Burke, and Michiel van de Panne. Motion doodles: an interface for sketching character motion. In *ACM Transactions on Graphics (TOG)*, volume 23, pages 424–431. ACM, 2004.

[TDB16] M. Tatarchenko, A. Dosovitskiy, and T. Brox. Multi-view 3d models from single images with a convolutional network. In *European Conference on Computer Vision (ECCV)*, 2016.

[TDB17] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2088–2096, 2017.

[TEG18] Jianchao Tan, Jose Echevarria, and Yotam Gingold. Efficient palette-based decomposition and recoloring of images via rgbxy-space geometry. In *SIGGRAPH Asia 2018 Technical Papers*, page 262. ACM, 2018.

[VdBBR$^+$12] Michael Van den Bergh, Xavier Boix, Gemma Roig, Benjamin de Capitani, and Luc Van Gool. Seeds: Superpixels extracted via energy-driven sampling. In *European conference on computer vision*, pages 13–26. Springer, 2012.

[VSR13] C. Vogel, K. Schindler, and S. Roth. Piecewise rigid scene flow. In *IEEE International Conference on Computer Vision (ICCV)*, 2013.

[WAvK$^+$12] Yunhai Wang, Shmulik Asafi, Oliver van Kaick, Hao Zhang, Daniel Cohen-Or, and Baoquan Chen. Active co-analysis of a set of shapes. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*, 2012.

[WDAC06] Jue Wang, Steven M. Drucker, Maneesh Agrawala, and Michael F. Cohen. The cartoon animation filter. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 25(3):1169–1173, 2006.

[WFG15] Xiaolong Wang, David Fouhey, and Abhinav Gupta. Designing deep networks for surface normal estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 539–547, 2015.

[WKL15] Fang Wang, Le Kang, and Yi Li. Sketch-based 3d shape retrieval using convolutional neural networks. In *IEEE Computer Vision and Pattern Recognition (CVPR)*, pages 1875–1883, 2015.

[WRDF13] Neal Wadhwa, Michael Rubinstein, Frédo Durand, and William T. Freeman. Phase-based video motion processing. *ACM Transaction on Graphics (Proc. SIGGRAPH)*, 32(4), 2013.

[WRS+12] Hao-Yu Wu, Michael Rubinstein, Eugene Shih, John Guttag, Frédo Durand, and William T. Freeman. Eulerian video magnification for revealing subtle changes in the world. *ACM Transaction on Graphics (Proc. SIGGRAPH)*, 31(4), 2012.

[WSK+15] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shape modeling. *IEEE Computer Vision and Pattern Recognition (CVPR)*, 2015.

[WXSC04] Jue Wang, Yingqing Xu, Heung-Yeung Shum, and Michael F Cohen. Video tooning. In *ACM Transactions on Graphics (ToG)*, volume 23, pages 574–583. ACM, 2004.

[WZL+18] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2mesh: Generating 3d mesh models from single rgb images. In *European Conference on Computer Vision (ECCV)*, 2018.

[WZX+16] Jiajun Wu, Chengkai Zhang, Tianfan Xue, William T Freeman, and Joshua B Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Neural Information Processing Systems (NIPS)*, 2016.

[XCS$^+$14] Baoxuan Xu, William Chang, Alla Sheffer, Adrien Bousseau, James McCrae, and Karan Singh. True2form: 3d curve networks from 2d sketches via selective regularization. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 33(4), 2014.

[XGS15a] Qiuying Xu, Yotam Gingold, and Karan Singh. Inverse toon shading: Interactive normal field modeling with isophotes. In *Proceedings of Sketch-Based Interfaces and Modeling (SBIM)*, June 2015.

[XGS15b] Qiuying Xu, Yotam Gingold, and Karan Singh. Inverse toon shading: interactive normal field modeling with isophotes. In *Proceedings of the workshop on Sketch-Based Interfaces and Modeling*, pages 15–25. Eurographics Association, 2015.

[XXM$^+$13] Xiaohua Xie, Kai Xu, Niloy J Mitra, Daniel Cohen-Or, Wenyong Gong, Qi Su, and Baoquan Chen. Sketch-to-design: Context-based part assembly. In *Computer Graphics Forum*, volume 32, pages 233–245. Wiley Online Library, 2013.

[YL15] J. Yang and H. Li. Dense, accurate optical flow estimation with piecewise parametric model. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[YYSL16] Xinchen Yan, Jimei Yang, Kihyuk Sohn, and Honglak Lee. Attribute2image: Conditional image generation from visual attributes. In *European Conference on Computer Vision (ECCV)*, pages 776–791. Springer, 2016.

[ZHH96] Robert C. Zeleznik, Kenneth P. Herndon, and John F. Hughes. Sketch: An interface for sketching 3d scenes. In *SIGGRAPH*, pages 163–170. ACM, 1996.

[ZHP$^+$19] Long Zhao, Fangda Han, Xi Peng, Xun Zhang, Mubbasir Kapadia, Vladimir Pavlovic, and Dimitris N Metaxas. Cartoonish sketch-based face editing in videos using identity deformation transfer. *Computers & Graphics*, 2019.

[ZLDM16] Youyi Zheng, Han Liu, Julie Dorsey, and Niloy Mitra. Smart canvas : Context-inferred interpretation of sketches for preparatory design studies. *Computer Graphics Forum*, 35(2):37–48, 2016.

[ZSY+17] Yinda Zhang, Shuran Song, Ersin Yumer, Manolis Savva, Joon-Young Lee, Hailin Jin, and Thomas Funkhouser. Physically-based rendering for indoor scene understanding using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5287–5295, 2017.