Assistive Visual Content Creation Tools via Multimodal Correlation Analysis

James W. Hennessey

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Engineering

of

University College London.

Department of Computer Science

University College London

March 31, 2018

Dedicated to my parents.

Declaration

I, James W. Hennessey, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

James W. Hennessey

Abstract

Visual imagery is ubiquitous in society and can take various formats: from 2D sketches and photographs to photorealistic 3D renderings and animations. The creation processes for each of these mediums have their own unique challenges and methodologies that artists need to overcome and master. For example, for an artist to depict a 3D scene in a 2D drawing they need to understand foreshortening effects to position and scale objects accurately on the page; or, when modeling 3D scenes, artists need to understand how light interacts with objects and materials, to achieve a desired appearance.

Many of these tasks can be complex, time-consuming, and repetitive for content creators. The goal of this thesis is to develop tools to alleviate artists from some of these issues and to assist them in the creation process. The key hypothesis is that understanding the relationships between multiple signals present in the scene being created enables such assistive tools.

This thesis proposes three assistive tools. First, we present an image degradation model for depth-augmented image editing to help evaluate the quality of the image manipulation. Second, we address the problem of teaching novices to draw objects accurately by automatically generating easy-to-follow sketching tutorials for arbitrary 3D objects. Finally, we propose a method to automatically transfer 2D parametric user edits made to rendered 3D scenes to global variations of the original scene.

Acknowledgements

I'd like to sincerely thank my advisor Niloy Mitra for his mentorship during my masters and doctoral studies. Over the last six years I have learnt countless things from Niloy. While the many technical skills I've learnt will be very useful, it will be the lessons learned on choosing research topics, problem solving and collaborating with others, that will be most valuable. I really thank him for his optimism, encouragement to aim high and ongoing support.

I'd like to extend my gratitude to Holger Winnemöeller, Mira Dontcheva, Wilmot Li, Bryan Russell and Eli Shechtman who mentored me during two internships at Adobe Research. Having the opportunity to collaborate which such a varied group of respected researchers, helped with my own researcher development. They all made significant contributions to various parts of the research presented in this thesis. I'd particularly like to thank Bryan for his career and research advice beyond my internship. I'd like to express my thanks to Han Liu who collaborated on the How2Sketch project.

I'd like to show my appreciation to past and present members of the Smart Geometry Group for their advice and encouragement: Melinos, Bongjin, Nicolas, Martin, Chi-Han, Moos, Aron, Paul, Tom, Dan, Carlo, Robin, Tuanfeng and Yu-Shiang. I'd particularly like to thank Paul Guerrero for his help with user studies and useful discussions when working late at night. I'd also like to thank the VECG group, along with my friends and colleagues at Adobe Research and Disney Research. All of them parted various bits of wisdom to me over the years and I've really enjoyed the great discussions on research and current affairs. I'd like to thank the 'VEIV Fun Group' - Kelvin, Drew, Lucy, Sebastian, Jacob, David, Maria and

Theo - for always making me laugh and seeing the funnier side of the EngD process.

I'd like to thank my friends - Thom, Joe, Kieran, Nicola, Jamie, Sam, Fred, Hettie, Walkies, Josh, Thomas, Campo, Will, Alex, James, Clarissa, Georgina and the extended Southend-on-Sea gang - for always reminding me that there are more important things to life than SIGGRAPH deadlines.

Most importantly I'd like to acknowledge my family: my siblings Tom and Kate, and my parents Mary and John. I'd like to thank them for giving me a loving upbringing and always supporting me with anything I've chosen to do in life. They have all been a pillars of support throughout the last five years, particularly during all of the paper rejections, without them I wouldn't have started or finished this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

As humans we inherently communicate visually, from our body language to creating detailed architectural drawings. The reason why we prefer to communicate visually has various theories. Some art theorists believe it is down to children learning to see before they can speak [1]. While molecular biologists argue, it is due to more of our brains being dedicated to visual processing than any of our other senses combined [2]. Regardless of why we prefer communicating visually, it is humans have a long history of creating imagery to convey ideas and emotions.

Our interest in visual communication dates back over 40,000 years starting with primitive cave paintings of animals. The cave paintings were created using tools to engrave into rock or by using natural pigments such as charcoal or clay [3]. These primitive drawings predate the invention of written language introduced by the Egyptians around 2000 BC [3], perhaps another indication that communicating visually is more natural to humans. The variety and quality of visual medium to communicate has increased over the years. We now communicate using a wide variety of techniques such as high-fidelity 3D models, manipulated photographs and architectural designs. Driving the development of these medium have been the advances in the tools available to artists and designers.

Visual content creation tools have seen incremental improvements, as well as radical changes. From the various tools for traditional painting and drawing to

the domain specific software tools used today for 3D modelling or video editing. The progression of technology means the tools available to artists have become increasingly more powerful. However, they have also become increasingly more complex and there are a number of challenges artists face.

Some of the challenges that artists face are more theoretical, such as drawing from observation skills. Developing such skills is difficult, but once mastered can be transferred to different applications that require an understanding of how the 3D world should be depicted on a 2D surface. Other challenges are tool specific but with reoccurring themes. For example, commonly an artist will perform the same task multiple times and getting the exact same effect every time can be challenging. Both these theoretical and more applied challenges give an opportunity for researchers to explore more intelligent tools to help artists.

1.2 Assistive Tools

To address some of the challenges outlined in Section 1.1 this thesis focuses on creating assistive tools for artists and designers. We consider tools to be assistive when designed to help visual artists achieve a specific task, such as drawing an object more accurately or performing a type of image edit. To better define the scope of assistive tools and how to evaluate their success, we outline some goals and objectives below.

Goals. The goal of an assistive tool is to help artists more easily achieve a task that is otherwise complex, tedious and/or time consuming. We recognise some artistic endeavours are not about how efficiently artists can work or how accurate their images are, however, this thesis is specifically interested in these qualities of assistive tools. At the same time efficiency and accuracy needs to be balanced with having the artist being able to achieve their vision. Therefore we want our assistive tools to augment the artist's creativity without taking away control, fitting naturally in their workflow.

Objectives. To measures how well the proposed tools achieve these goals we have specified a number of objectives that can be measured by qualitative or quantitative analysis. For each of the tools we propose our objectives are to:

- Demonstrate that the proposed tool is more time efficient at achieving a given task compared to existing methods.
- Demonstrate that the proposed tool is more accurate at achieving a given task compared to existing methods.
- Get feedback from artists and designers to validate the usefulness of a the tool in their workflow.

For each of the proposed tools we will measure their success against these objectives.

1.3 Multimodal Correlation Analysis

An important concept throughout this thesis is the idea of a reference scene. A reference scene is an input representation of the scene that the artist would like to create or manipulate. For example, it could be an input photograph, 3D model or rendering. If a reference scene is available to the artist we argue that assistive tools can be enabled by creating different representations of the reference scene and performing multimodal correlation analysis on these auxiliary representations. Understanding how different modalities correlate can provide the intelligence that an assistive tool requires.

There are many different way that multiple representations or modalities of the reference scene can be created. For instance a 3D object is normally represented as a polygon mesh, but after some geometry processing, can be represented as segmented primitives with inter-primitive relations. Another example of multiple representations is how a 3D rendering can be represented as a single image (commonly known as a beauty pass) or as a number of light-path expressions [4] that

composite together to create the beauty pass. Leveraging multiple representations we demonstrate is useful in creating a variety of assistive tool as it allows for a higher-level of understanding of the scene.

The final common step in creating assistive tools is understanding how different signals correlate and using them to power an assistive tool. We use the term correlation in a broad sense, simply using it to mean having an understanding how two or more variables relate to each other. We use different techniques to find correlation, as the best method to find correlation depends on what information is useful in achieving a given task and how it can be utilised within an assistive tool.

1.4 Contributions

In our work we make three key technical contributions for creating assistive tools for visual content creation. These technical contributions and much of the text in this dissertation come from three peer-reviewed publications. These have been published as follows:

- James W. Hennessey and Niloy J. Mitra. 2015. An Image Degradation Model for Depth-augmented Image Editing. Computer Graphics Forum 34, 5 (August 2015), 191-199. [5]

- James W. Hennessey, Han Liu, Holger Winnemller, Mira Dontcheva, and Niloy J. Mitra. 2017. How2Sketch: generating easy-to-follow tutorials for sketching 3D objects. In Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D '17), Article 8 (March 2017), 11 pages. [6]

- James W. Hennessey, Wilmot Li, Bryan Russell, Eli Shechtman, and Niloy J. Mitra. 2017. Transferring image-based edits for multi-channel compositing. ACM Transactions on Graphics. 36, 6, Article 179 (November 2017), 16 pages. [7]

The contributions made in these publications are now breifly summarised below.

- In [5] the novel contribution is an image degradation model that predicts how well an image edit can be performed in presence of coarse depth information. The image degradation model can then be used to help guide artists when performing an image manipulation.

- In [6] we propose an algorithm for automatically generating easy-to-follow sketching tutorials for a user specified 3D model and viewpoint. These tutorials allow novice artists to learn structured techniques for accurately drawing objects of their choice.

- In [7] the core contribution is an algorithm for transferring parametric image-based edits to rendered scenes to global variations of the original scene. In this paper we also introduced a new editing workflow to make the creation of the parametric image-based edits easier. These tools allow for artists to more easily make image-based edits, while, also being able to continue altering the 3D scene they are creating.

1.5 Organisation

The remainder of this dissertation is organised as follows:

Chapter 2, Related Work. This chapter provides an overview of the related research literature from the human-computer interaction and computer graphics research communities. Specifically, it discusses the history and development of of *image manipulation*, *sketching and sketch-based interfaces*, and *edit and style transfer*.

Chapter 3, An Image Degradation Model for Depth-augmented Image Editing. In this chapter we investigate how even coarse depth information can be exploited to address some of the fundamental challenges in image editing namely producing

correct perspective, handling occlusion, and obtaining segmentation. To this end, we propose a novel image degradation model that predicts how well an image edit can be performed in presence of coarse depth information.

Chapter 4, Generating Easy-To-Follow Tutorials for Sketching 3D Objects. Accurately drawing 3D objects is difficult for untrained individuals, as it requires an understanding of perspective and its effects on geometry and proportions. Step-by-step tutorials break the complex task of sketching an entire object down into easy-to-follow steps that even a novice can follow. In this chapter we address this problem by proposing an method for automatically generating easy-to-follow tutorials for arbitrary 3D objects. We demonstrate that our easy-to-follow tutorials result in more accurate drawings compared to baseline tutorials.

Chapter 5, Transferring Image-based Edits for Multi-Channel Compositing. In this chapter we propose a method to automatically transfer parametric image-based edits made to rendered scenes across variations of object geometry, illumination, and viewpoint. This transfer problem is challenging since many edits may be visually plausible but non-physical, with a successful transfer dependent on an unknown set of scene attributes that may include both photometric and non-photometric features. We compare our edit-transfer method to existing state-of-the-art methods demonstrating that our method results in more accurate edit transfers. This chapter also introduces a new editing workflow that allows artists to be more efficient in creating image-based edits to rendered scenes.

Chapter 6, Conclusion. In the final chapter we summarise the proposed tools and discuss future work in this field.

Chapter 2

Related Work

Assistive tools for visual content creation have a very broad body of related literature. In this chapter the most relevant research into digital tools that aide artists and designers to create imagery are discussed, these predominantly come from the computer graphics and human-computer interaction research communities. The chapter starts by looking at research into *image manipulation* (see Section 2.1), which, is most relevant to the image editing tools proposed in Chapters 3 and 5. The research related to Chapter 4 is then discussed in Section 2.2, which, covers *sketching and sketch-based interfaces*. Finally, Section 2.3 includes discussion of *edit and style transfer* research related to Chapter 5.

2.1 Image Manipulation

In this section the literature most relevant to our image degradation model (see Chapter 3) is discussed. This covers *traditional image editing* techniques, *depth-aware image editing* and methods for *rendering 3D models into images*.

Traditional image editing. Given the popularity and ubiquity of images, significant research has been devoted over the last decades in developing image editing algorithms. Many of them are commonly available as standard options in image editing packages like Gimp, Photoshop, etc. The central challenge we are most interested in is to plausibly account for the lack of depth in input images. This

Figure 2.1: Many other works have looked at advanced image editing techniques, such as Zheng et. al [8], whom use geometric proxies to approximate the geometry of objects in the scene. The allows artists to easily manipulate the shape of objects in the image.

makes it difficult to correctly handle perspective and/or occlusion effects. Even advanced methods like PatchMatch [9] fail when scenes are cluttered or a texture changes with the perspective of the object. In the context of segmentation, the commonly used GrabCut segmentation algorithm [10] cannot satisfactorily segment objects from different depth with the same appearance. In Chapter 3 we improve the results of these algorithms by the use of geometric planar primitives. It is worth noting research conduced concurrently and subsequently to ours also overcome the limitations of PatchMatch [9] and GrabCut [10]. The state-of-the-art techniques in semantic segmentation [11, 12, 13, 14, 15, 16] make use of advanced deep learning convolutional neural network (CNN) architectures, such a fully-connected networks, region based CNNs and encoder-decoder networks. Similarly advances in image completion make use of deep learning approaches, particularly with generative adversarial neural network (GAN) architectures [17, 18].

Depth-aware image editing. Many depth-aware solutions have been proposed to tackle specific image editing use cases and problems. For example RepFinder [19] use repeated objects in a scene to assist with image completion and depth ordering, while, [20] use vanishing points for artistically changing image perspective. We approach the problem in Chapter 3 by using noisy depth information and want to deal with these challenges for more general scenes.

A recent approach for editing man-made objects in 3D is to allow the user to create 3D proxies for objects in the scene. One example approximates objects with cuboids [8] and another generalized cylinders [21]. The results for both are im-

pressive but require substantial and specialized user-interaction [22]. Our approach in Chapter 3 also uses geometric primitives for editing objects in 3D, however, we demonstrate that it is not necessary to accurately parameterize the objects for a range of interactions.

Our application of parallax photography in Chapter 3 shares motivation with viewpoint changes from a single image demonstrated by [23]. Their application again has significant interaction to assign a depth to each segment. Tour into the picture [24] allow users to make animations from a single image by changing viewpoints, but do not complete occluded regions or achieve a parallax effect.

Rendering 3D Models in Images. Recent applications combine images with 3D models with impressive results in either editing the scene [25] or realistically compositing objects in the scene [26]. These require 3D models, that match objects in the scene, to be available for edits to be made. This setup is the same as the one used in Chapter 5, however, we focus on creating and transferring image-based edits to these scenes. In Chapter 3 we only use information from the original input photograph and aim to use the depth information as go between the image and the 3D model. RGB-D images have become increasingly popular and many methods have been proposed to address the common challenges of segmentation and depth map completion [27, 28, 29].

Our Chapter 3 application of parallax photography can be compared with [30] who create Parallax Photographs from LightField images. Their results are impressive but as a Lightfield is sampling the ray space the input is a lot richer (and heavy-weight). Instead RGB-D images are much easier to acquire. However, the simplicity introduces a number of new problems such as segmentation, occlusion and image completion.

2.2 Sketching and Sketch-based Interfaces

This section discusses the literature around sketching and sketch-based interfaces relevant to our tool in Chapter 4. The section covers *assisted drawing* tools, systems

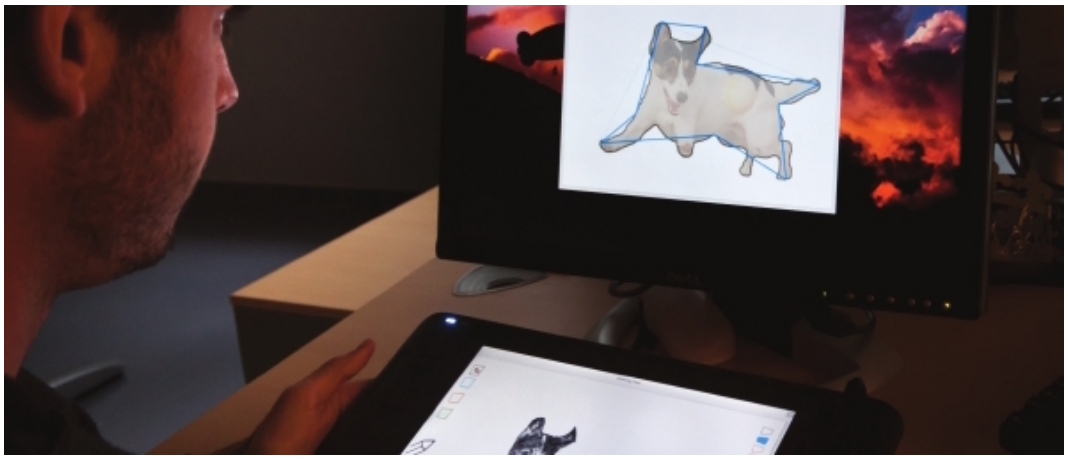


Figure 2.2: An excellent example of an assistive tool for learning to draw is The Drawing Assistant [31]. The tool generates automatic guidance over an input photograph and interactive feedback to help novices learns drawing-from-observation skills.

for creating *tutorials*, theoretical research into *drawing expertise* and methods for generating or utilising *line drawings*.

Assisted drawing. Various assistive tools have been proposed to assist a user in sketching: Correcting user input based on geometric analysis of the users input strokes [32, 33, 34]; relying on an underlying image to guide the user [31, 35, 36, 37]; or using crowdsourced data (e.g., many sketches) to improve the users drawing [38, 39, 40, 41, 42] at a local stroke level. Our work in Chapter 4 focuses on suggesting a meaningful drawing order and easy-to-construct guides for accurate depiction of perspective and proportions. The mentioned related work on Stroke correction or beautification is orthogonal to our main contribution and may be used to complement the contour drawing phase of our sketching tutorials. Other assisted sketching systems take as input 2D sketches and interpret them as 3D curve networks [43]. More advanced methods in the Sketch-based interfaces literature [44, 45, 46] use 2D input to infer 3D geometry or surface normals for complex shading. We focus on the automatic generation of sketching tutorials, rather than automatic inference based on the sketched curves.

Tutorials. A good tutorial greatly facilitates understanding. Many attempts have

been made to automatically generate high-quality tutorials for different applications. A digital drawing tutorial system was proposed by [47] that allows an expert to create tutorials for novices. Tutorial generation systems [48, 49] for specific sketching tasks have also been proposed, for example drawing a single scene with pre-defined objects, or 'eyes'. Grabler et al. [50] developed a tutorial system for photo manipulation tasks. In contrast, the focus of our method in Chapter 4 is on generating tutorials for sketching 3D models of man-made objects.

Drawing expertise. Tchalenko [51] found that novices and professional artists have comparable accuracy when performing basic line drawing tasks (straight lines and simple curves). However, in a follow-up study [52], he showed that when copying complex artworks, novices made significantly more errors than artists. The main difference in drawing strategy was that experts divided complex lines into easy-to-draw short segments. Schmidt et al. [53] found that experts made qualitatively similar errors to non-artists, indicating that perspective drawing is hard, even for trained users. Particularly for off-axis viewing angles, drawing error increased significantly. In an observational study, Grimm [54] found that artists commonly used a coarse-to-fine strategy starting with blocking shapes and finishing by drawing detailed items at the end. Our tool in Chapter 4 assists the user by breaking the drawing process up into basic steps that are easy to execute and by explicitly indicating vanishing line directions.

Line drawings. Many methods for generating stylized artistic renderings of objects have been proposed (see [55] for a survey). We leverage stylization in Chapter 4 to visually distinguish the various line types of our tutorials (perspective lines, guides, contours, etc). Other researchers investigated which features artists typically draw to convey 3D shape [56, 57, 58, 59]. Finally, [60] and [61] infer plausible contour ordering from 2D and 3D inputs, respectively. While the derived sequences are plausible, they are not tailored for tutorials and do not provide specific guidelines to make them easy to follow.

Figure 2.3: A seminal work in the style transfer literature is Image Analogies [62] proposed by Hertzmann et al. The framework allows the user to provide an input image (A), a stylised version of that image (A') and a target image (B). The algorithm then automatically generates an image (B') such that its relationship to B is analogous to the relationship between A and A'.

2.3 Edit and Style Transfer

This section explores related work for our edit transfer tool in Chapter 5. It discusses *3D appearance editing* as a form of edit transfer, *2D edit transfer* algorithms and *parametric edit transfer* techniques.

3D Appearance Editing. One approach to editing a 2D image of a rendered 3D scene, then transferring the edit to global variations on the 3D scene, would be to infer the 3D appearance edit. Having the appearance changed in 3D means the edit would automatically be transferred to any 3D scene changes. Subsequently, there is a significant body of work on manipulating the appearance of rendered 3D objects. In particular, many of these methods help users adjust the output of physically-based rendering techniques via "artistic" controls, such as scribble based material appearance transfer [63], relighting a scene using a lighting paint brush [64], exploting image-space repetitions to transfer edits [65], or using voice to interactively edit image edits [66]. These are summarized in a recent survey by Schmidt et al. [67]. While such controls are designed to facilitate the editing process, making specific adjustments to visual elements of a rendered scene is often still quite challenging given the complex interactions between light, materials, and geometry within most scenes. Moreover, many edits that artists want to make are either non-physical in nature (e.g., boosting and muting various highlights on an object) or much easier to specify in image space (e.g., emphasizing rim lighting at specific object contours). Finally, in some cases, the artist who creates the final, composited product image

may simply have much more familiarity with 2D image editing tools than 3D software. As a result, in Chapter 5 we focus on image-based retouching workflows rather than 3D appearance editing.

Another type of 3D appearance editing that is typically used for visual effects in computer-generated movies and animation is node-based compositing. For example, Nuke [68] is a popular commercial tool that supports this type of compositing. In such tools, masks are defined based on object or material ids, and a user-specified set of parameter adjustments are applied to the entire object or material based on these masks in each rendered frame. In contrast, our goal is to represent and transfer edits that are localized to specific parts of an object or material.

2D Edit Transfer. Previous work proposes a wide variety of techniques that facilitate image editing operations [69, 70, 9, 65, 8, 71, 72, 5]. The most relevant to our work are methods for transferring image edits across different images. Some approaches leverage inter-image correspondences to transfer edits to different viewpoints of the same scene or people [73, 74, 75, 76, 77, 78]. However, even with access to perfect correspondences, such methods are not sufficient for our problem in Chapter 5, since many retouching edits relate to lighting-dependent features (see Figure 5.9). An alternative approach is to use the editing history from the user interface [50, 79] or history inferred from the exemplar edit [80] to transfer edits to new images. Our edit transfer method is agnostic to the sequence of editing operations and only requires the final edited exemplar image. Our method builds on patch-based synthesis approaches [81, 82, 83, 84] to transfer edits. More specifically, our contribution is a synthesis method adaptive to the user's edits.

We can also view image-based style transfer techniques as a form of edit transfer [62, 85, 86]. However, such methods also have drawbacks. The neural network-based style transfer of Gatys et al. [86] can be difficult to control precisely. Image analogies [62, 85] provides a different formulation, but determining the appropriate guidance channels to successfully transfer edits is non-trivial, as noted

already in Section 5.1 and in Figure 5.2. In addition, all of the aforementioned image-based techniques aim to transfer or synthesize the edit itself in the target image. In contrast, our goal is to transfer spatially localized parameterized edits that can be further refined by the artist.

Parametric Edit Transfers. Finally, some previous work has proposed techniques for transferring parameterized edits in the domains of 3D modeling and 2D vector graphics [87, 88, 89, 90]. These methods demonstrate the utility of parameterized edit transfer for various content creation tasks. In Chapter 5, we present strategies for supporting a related type of edit transfer in the context of image-based edits to rendered content.

The following chapter is the first of our assistive tools. The tool focuses on helping artists make 3D image manipulations to 2D photographs.

Chapter 3

An Image Degradation Model for Depth-augmented Image Editing

Photographs remain the most popular medium to capture our surroundings. Although significant advances have been made in developing image editing tools, some edits remain difficult. One of the key challenges when performing image edits is to intelligently account for unknown scene geometry. To overcome this challenge we explore an assistive tool for the specific task of generating parallax photographs. Our tool takes a colour photograph as the input reference scene and a coarse depth map as a second axillary scene representation. To create our assistive tool, we investigate how multimodal correlation analysis of the coarse depth information and the photograph can be used to address some of the fundamental challenges in image editing namely producing correct perspective, handling occlusion, and obtaining segmentation. To this end, we propose a novel *image degradation model* that predicts how well an image edit can be performed in presence of coarse depth information. Technically, we create proxy geometry to summarize available depth information, and use it to predict occlusions and ordering between image patches, complete occluded regions, and anticipate image-level changes under camera movement. We evaluate the proposed image degradation model in the context of parallax photography from single depth images.

3.1 Introduction

Images remain the most dominant and ubiquitous of visual mediums. A vast selection of tools exists supporting various image editing and manipulation tasks. Typically, users are interested in manipulating scene content or camera pose in order to mimic being in the original 3D scene. However, the lack of actual geometry and depth information makes such edits theoretically impossible to perform correctly.

Beyond full scale 3D acquisition, one can alternatively capture a lightfield of the scene to accurately support many advanced manipulations (e.g., change in camera pose, simulate depth of field effects, etc.). This, however, comes at the cost of specialized and costly imaging setup. In this paper, we show that even very coarse and incomplete depth information can vastly simplify many image processing tasks. This is particularly relevant given the growing ubiquity of depth sensors that capture high resolution RGB information with loosely synchronized noisy depth information. We demonstrate such depth information can be used to plausibly handle occlusion, perspective, and completion effects — all from single view inputs.

More interestingly, we propose an *image degradation model* that predicts the success likelihood of a proposed manipulation. The motivation behind the degradation model comes from two observations: (i) image completion algorithms are limited and can leave undesirable artefacts and (ii) by introducing depth information to an image it enables the control of occlusions by changing camera viewpoint. The objective of the image degradation model is to identify poorly completed regions and prevent them from being revealed. Technically, we use the rough depth information to help create a planar proxy based abstraction of the input image. We propose an iterative algorithm to segment the input, while estimating the corresponding planar proxies to act as billboards for the respective segments. The information is then used to create a layered set of planar proxies with infilled and clipped textures per layer. Finally, we estimate a degradation score for new camera poses to predict the plausibility of the synthesized image composition.

We evaluate our framework in the context of creating parallax videos from single images. This application demonstrates a number of the challenges including

Figure 3.1: Example RGB-D input: Point cloud rendered from original camera pose (left) showing large regions of missing depth information labeled in red. When rendered from a different camera pose (right) the point cloud reveals many points have mislabeled depth values. The coarseness of the data emphasises the need for decomposing the scene into geometric proxies.

segmentation, image completion, perspective, occlusion and depth ordering in one use-case. We evaluate the proposed method on a variety of scenarios with both planar and non-planar objects, with and without texture.

3.2 Overview

Our application takes as input a single registered RGB-D image captured using a camera and calibrated consumer depth sensor. The RGB-D inputs have high density RGB measurements, but poor and incomplete depth information (see Figure 3.1). The goal is to utilize the available information to create an *image degradation* model to predict how successful typical image manipulations will be. In other words, the degradation model characterizes edits as simple, or difficult and likely to show artifacts.

In order to build such a model, we analyze an input RGB-D image to create an intermediate representation. Specifically, we segment the input (either automatically or semi-automatically), billboard-approximate them using planar proxies, obtain the relative ordering of the respective planes, and infill the occluded regions for each segment. We then build an image degradation model that captures the plausibility of the infilled pixels. The effectiveness of the proposed image degradation model is evaluated by using it to find suitable camera paths to create aesthetically

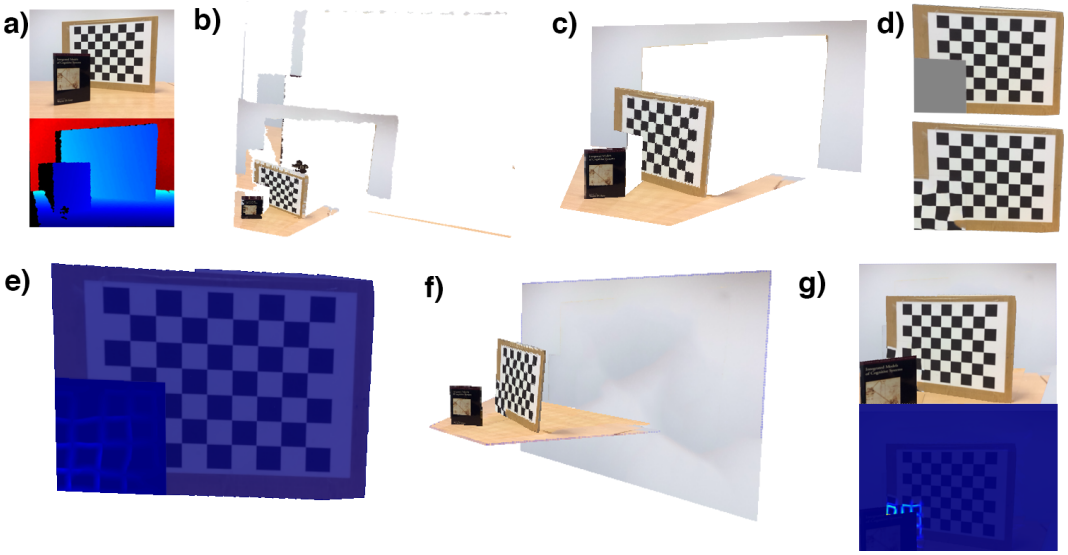


Figure 3.2: Method overview: (a) Input RGB + depth image; (b) incomplete point cloud with noisy data (note the misalignment of RGB and depth); (c) segmentation, primitive fitting, and depth completion; (d) occlusions identified and infilled using the primitives (shown for one segment); (e) degradation model built for infilled pixels; (f) decomposed and completed layered scene; and, (e) new view synthesized from user defined camera pose is flagged by the degradation model as undesirable.

pleasing parallax photographs from single images.

Our pipeline (see Figure 3.2) has three stages: (i) *Scene decomposition and completion* wherein we propose an iterative approach for image segmentation, depth map completion, and planar proxy fitting. These primitives are then used to determine occlusions and improve image completion. (ii) *An image degradation model* is then created consisting of a degradation score for each of the occluded pixels in each segment, completed in the previous step, representing the plausibility of the completed pixel. The degradation model consists of a spatial term and texture term. The intuitive idea behind these terms is that pixels close to known pixel values and in a low texture region should receive a low degradation score, versus those far from known pixels with a large amount of texture variations. Finally, we use the decomposed and completed scene with the computed degradation model in the (iii) *camera path generation* for a parallax photograph. Starting from user specified camera key frames artists can utilize the degradation model to find a good camera path between them.

Contributions. Our key contribution is an approach to use coarse depth to simplify image manipulation tasks. Central to this is a novel image degradation model that captures the quality of synthetic regions of images. Additionally we propose a method for creating proxy geometry to summarize coarse depth information and exploit these proxies when dealing with common challenges such as segmentation, occlusions, and perspective changes.

3.3 Method

All of our RGB-D images are captured using an Apple iPad and Occipital Structure-Sensor. The StructureSensor has a range of 0.4m to 3.5m+ with precision 0.5mm at 40cm, 30mm at 3m. In practice the upper bound is further but precision degrades, which, our pipeline mitigates. We use Occipital's calibration app to register the color and depth channels.

The RGB-D images are input into our system (see Figure 3.2 for pipeline). We convert the data to a point cloud and estimate pointwise normals using local PCA fitting. Note that the data is largely incomplete (marked in black in the depth channel) and also there is misalignment between color and depth channels as seen in the point cloud. The scene is then segmented using color and depth information (automatically or with user guidance), abstracted with planar primitives, and completed using guidance from the obtained primitives.

3.3.1 Scene Decomposition and Completion

The main goal of this step is to simultaneously performs segmentation, planar primitive fitting and pixel assignment. We couple these three steps in an iterative approach that reassigns pixels to primitive to improve the segmentation and obtain improved primitive fits. Figure 3.3 shows an example.

Scene decomposition. First, we segment the RGB image into SLIC SuperPixels [91] (Figure 3.3c) and fit planar primitives to the different segments. We encode the fitted primitives in the normal-intercept form as $\mathbf{n} \cdot \mathbf{p} + d = 0$. We cluster the

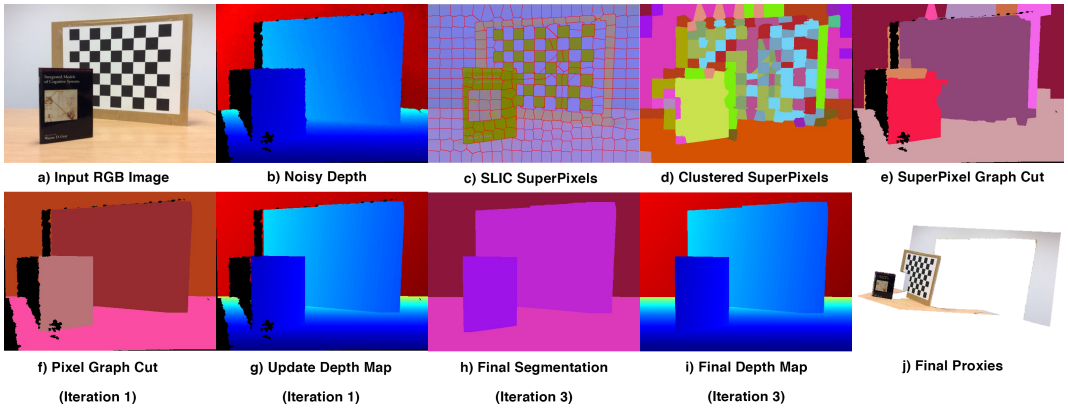


Figure 3.3: Scene decomposition: simultaneously performs segmentation, planar primitive fitting, and pixel depth assignment a) Input RGB image b) Input depth map; note incomplete and missing regions c) SLIC Superpixels computed and planer primitives fitting d) Clustering of SuperPixels plane primitives e) Alpha-expansion graph-cut at SuperPixel level f) First iteration of alpha-expansion graph-cut on pixel level g) First iteration of planar primitive fitting and depth map completion h) Final image segmentation i) Final primitive fitting and depth map completion j) Final segmentation and 3D proxies

superpixel plane primitives using k-means in the $\mathbb{R}^4$ space. Empirically, we found $k = 20$ provided good results (Figure 3.3d). We again fit planar proxies to the superpixel clusters to obtain a rough initial segmentation. Essentially, the clustering step links superpixels sharing similar fitted planes. This allows non-locally linking superpixels, for example walls are identified to be coming from the same plane even under occlusion.

An alpha-expansion graph cut [92] is used to improve upon the initial superpixel segmentation. The graph cut allows $(k+1)$ possible labels that a superpixel could be assigned, representing the current segments and their primitives from the superpixel clustering, and an additional possible assignment of a plane at large distance away. We use a unary cost for each label that encourage the average distance between the label's plane primitive and all points in a superpixel to be small and the average angle between point normals and plane normal to be similarly small.

$$E_u(i) := \frac{1}{N}\sum_{i=1}^{N}(|\mathbf{p}_i \cdot \mathbf{n}_{prim} + d_{prim}|) + \lambda \exp(-|\mathbf{n}_{sp} \cdot \mathbf{n}_{prim}|).$$

In our tests we set $\lambda = 1000$. Note that for pixels with no assigned depth value (i.e.,

missing depth) we skip the unary term. If a whole superpixel has no depth data then it is given a uniform cost for all primitives.

The pairwise cost for the graph encourages the neighbouring primitives to have similar color and depth as:

$$E_p(i,j) := \alpha \exp(-|\mathbf{c}_i - \mathbf{c}_j|) + \beta \exp(-|d_i - d_j|) \tag{3.1}$$

where, $\mathbf{c}_x$ and d_x respectively denote the mean color and depth values assigned to the current segmentation primitives and normalized between 0 and 1. We used $\alpha = 1000$ and $\beta = 200$ in our tests. Again, we exclude the depth term here if one of the pixels in a superpixels have no associated depth value. Finally, we refit planar segments to the updated segmentation results. The SuperPixel level graph cut can be seen in Figure 3.3e.

We then iteratively refine the segmentation and depth map but now working at the pixel level. Each of the three iterations consists of performing a pixel level alpha expansion, updating the primitives, and updating the depth map. Specifically, the alpha expansion uses the same terms as previously. However, as we are working at the pixel level the point to primitive distance is no longer averaged, nor is the color or depth term in the pairwise cost. We update the depth map by setting each point's position as the ray-plane intersection for the assigned primitive. In the first iteration, we only reassign the pixels with already known depth to correct flying pixels. Subsequently, we visit the remaining pixels to also fill in regions with missing depth.

User assistance. In complex scenes, the above approach can fail to detect small objects, or very similar objects (in depth and color) can be wrongly merged. This is particularly a challenge for mid to far objects, where the corresponding depth precision is particularly poor. In such cases, we allow the user to scribble objects as specific segments. From the scribble marked regions, we compute the region's mean color, $\mathbf{c}_\mu$, point normal, $\mathbf{n}_\mu$, and depth value, d_μ. Using region growing, we append neighboring candidate pixels p to the current region if the following

conditions hold:

$$|\mathbf{c}_\mu - \mathbf{c}_p| < \lambda_c \text{ AND } d_\mu - d_p < \lambda_d \text{ AND } \mathbf{n}_\mu \cdot \mathbf{n}_p < \lambda_n$$

where, $\lambda_c = 65$, $\lambda_d = 0.3(d_{max} - d_{min})$ and $\lambda_n = 25^\circ$. This rough segmentation is then used instead of the output of the SuperPixel level alpha-expansion, and we continue with the iterative segmentation, primitive fitting and depth map completion at the pixel level as previously described.

Billboarding. Note that we do not require the segmented regions to be planar. While it is possible to work directly with the 3D pointcloud segments, we demonstrate that *billboarding* the pointset is a much simpler and sufficient for many of the target applications (cf., [93]). This drastically simplifies subsequent processing steps while we can still plausibly handle perspective and occlusion effects. However, some segments are not well approximated by a plane and in some cases result in a plane with poor orientation. Hence, we identify the non-planer segments based on the corresponding fitting residue, and 'billboard' them fronto-parallel. Specifically, we assign the points to a plane with a normal facing the camera. This avoids inaccurate planes being fitted to an object, causing issues later in our pipeline. We found for scenes with a large range setting the residue threshold to 2000 best, scenes with medium range 1000 and small range scenes 300.

Occlusion map. Next, we identify which regions on the primitives are occluded by foreground objects. For each pixel we find the 3D point on each primitive using ray-plane intersection. If the point's depth is greater than the associated value in the completed depth map and the pixel for this segment has no color information (i.e., is not visible in the input image), we mark it as occluded. After searching over all the points in a layer, we test if the marked occluded regions are connected to a region on the primitive that is visible. This removes false positive occlusions when the object has actually ended, but this is not known at the primitive level. We further clip the primitives by extending the visible edges into the occluded regions.The

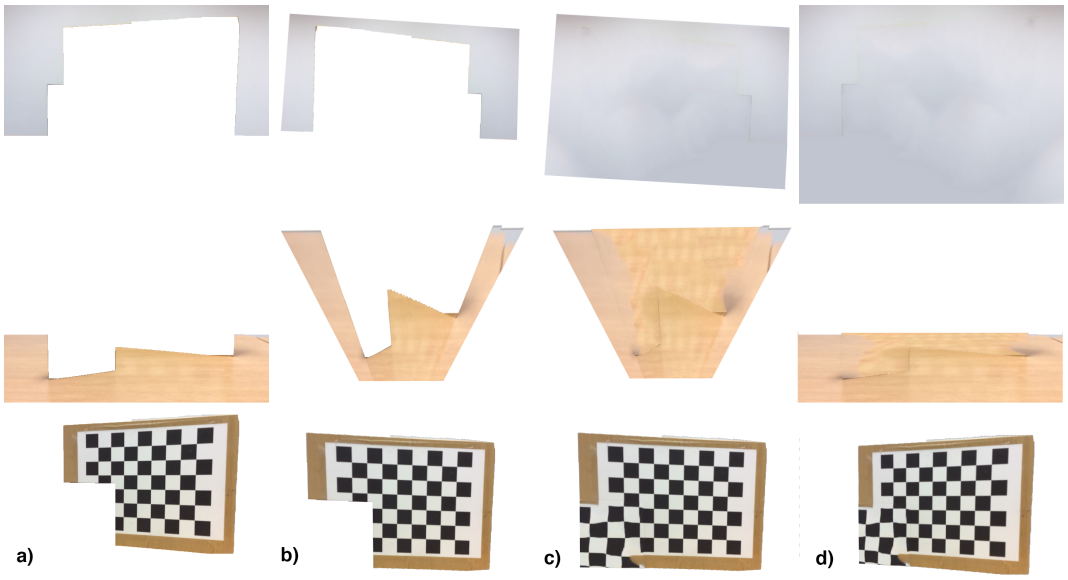


Figure 3.4: Image Completion: (a) Segments in their original position from input image (b) Segments made front-parallel to the camera using 2D homography (c) Occluded regions determined and infilled using PatchMatch (d) Infilled segments returned back to original pose.

result for each layer are pixels marked as being occluded that need to be infilled.

Fronto-parallel image completion. The final step is to complete the occluded regions. As has been observed by [94] image completion works better with planer surfaces. As our scene is positioned around the depth sensors optical center, we transform each primitive so it is fronto-parallel with the camera by finding the rotation between the primitive's normal vector and the vector pointing down the negative z-axis. We apply this transformation to the points in 3D, and find the corresponding 2D homography and apply it to the image (see Figure 3.4). Thus we exploit the planar proxies to obtain fast and light-weight image warping.

To deal with shadows before performing image completion, we grow the depth-occluded pixels slightly to include some visible pixels, removing any shadowing artefacts. We used the Photoshop implementation of PatchMatch [9] for image completion. We warp the new completed image back to the original pose by applying the inverse rotation.

3.3.2 Image Degradation Model

We can use the created scene abstraction to propose a simple image degradation model that predicts plausibility of image manipulations. In other words, it provides a confidence score for the quality of the infilled pixels from the previous step and penalize bad ones if they are revealed by proposed image manipulations. The overall degradation of the image is then the sum of pixel-level degradation scores visible in the image. For example, in the case of parallax photography from a single image, the degradation score is zero when the camera is at the origin, and increases as we move further from the original camera pose, however, *not* uniformly in all directions. Hence, the score can inform the user which directions to pursue, and more importantly which ones to avoid.

Our proposed per-pixel degradation score consists of two terms: a *spatial* term measuring proximity to known pixels and a *texture* term measuring the plausibility of infilled textures.

The spatial term captures the intuition that deeper inside occluded regions, our guesses will have access to less local information for clues. We compute it by using a breadth-first region growing approach starting at the boundary of known and unknown pixels. The boundary grows by adding any of the occluded pixels in the eight-connected neighborhood of the current boundary. We repeat the process until all unknown pixels have been visited. The degradation score is set to 1 for the

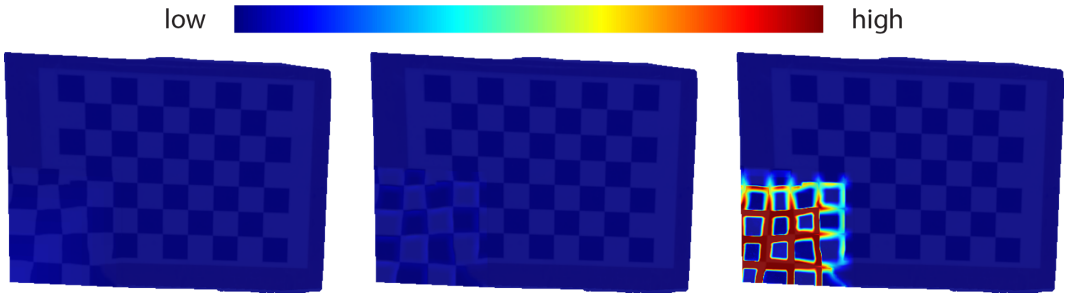


Figure 3.5: Degradation Model: The heat-maps visualize the degradation model for one segment. The spatial term (left) gently degrades the further known pixel values. The texture terms (middle) shows greater degradation around the sharp texture boundaries of the checkerboard pattern but low degradation in the centre of squares. The combined final term (right) shows how high textured regions close to known values will receive a moderate score; such regions further from known pixels receive a high score.

first layer and increments on each iterations.

The texture term captures the intuition that uniform (or structured) regions are more likely to be plausibly infilled. We compute it using a similar region growing approach. As the boundary region grows the degradation score is the average sum of absolute difference between each pixel and its $(2k+1) \times (2k+1)$ neighbourhood of visited or visible (i.e., known) pixels as:

$$texture(i, j) := \frac{1}{N} \sum_{x=-k}^{k} \sum_{y=-k}^{k} |I(i+x, j+y) - I(i, j)|.$$

where, N is the number of pixels in the neighbourhood that have been visited and the neighbourhood width $k = 10$. We estimate the final degradation score for a pixel simply as product of the two terms. Figure 3.5 shows an example.

3.3.3 Novel view synthesis

We can now use the layered texture-infilled planar proxies to generate novel view images, and also score the plausibility of the synthesized view using the proposed degradation model.

In the context of parallax photography, we have to generate a new image for each camera view along a path. The path is defined by the user who selects two key frames parameterized by camera location and rotation; the remaining poses on the path are linearly interpolated. Changing the camera pose is equivalent to applying the same transformation to all of the points in the scene, so our camera actually stays in one place and the scene is moved. To generate the new image equivalent to moving the camera position we warp the planar proxies to a new pose using homographies. To find the homographies we simply transform all the 3D points by the transformation from the original camera pose to the new pose, and project them onto the image plane. We store the points' original positions in the image plane and their new positions. Then, we estimate homographies to map the two sets of points using a RANSAC-based approach from the OpenCV library. Finally, we transform the points back to their original positions. We create the final image composite using the painters algorithm, iterating over the layers, and updating the

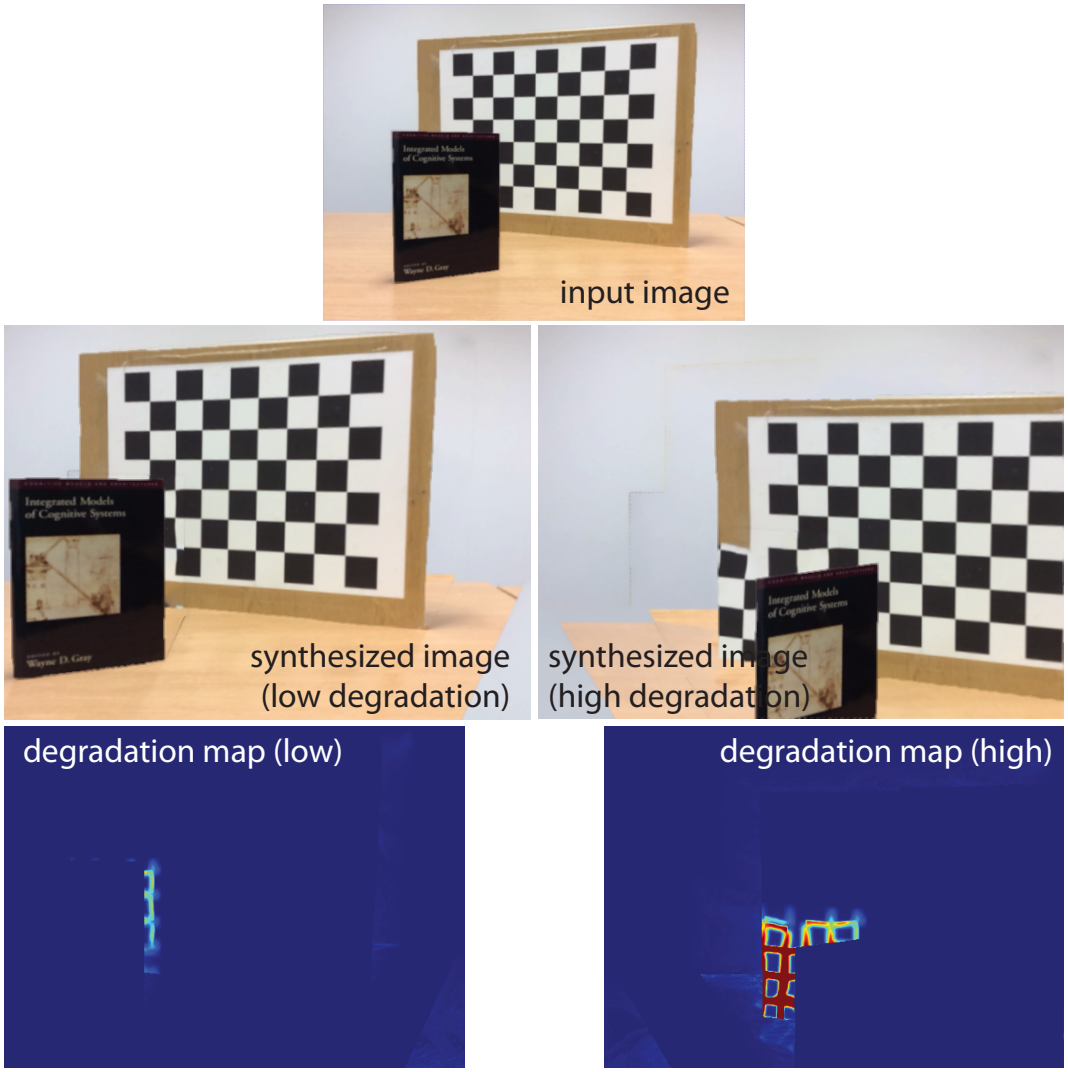


Figure 3.6: Novel View Synthesis: Input image (top) is used to create two novels views (middle) with degradation models (bottom). The left example has a low degradation score as the revealed region only has moderate texture and is close to known pixels. The right example has a high degradation score as it reveals a high texture region and far from known pixels.

output image pixel if a lower depth value (closer to camera) is found. Figure 3.6 shows some example novel views and their degradation models.

3.4 Results

We evaluate our framework for creating parallax photographs with the degradation providing feedback on the quality of the results. Figure 3.7 shows both of these in action in a variety of settings; video results can be found on our project webpage[1].

[1] http://geometry.cs.ucl.ac.uk/projects/2015/degradation_model/

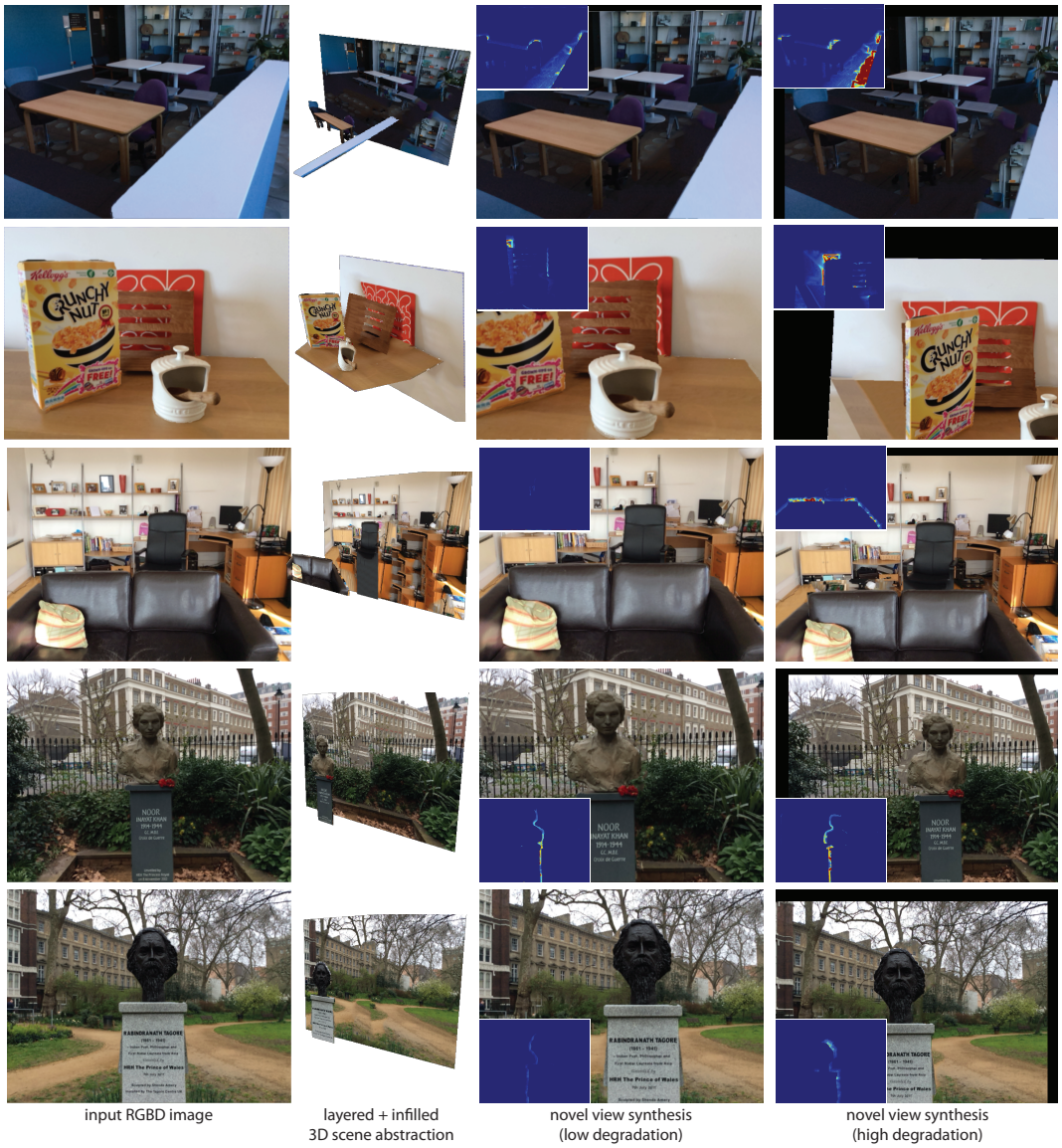


Figure 3.7: From top to bottom scenes: office, kitchen, living room, park1, park2. In each row, we show the input RGBD image, the abstracted layered scene, novel view synthesis with low degradation (inset showing degradation map), novel view synthesis with high degradation (inset showing degradation map), respectively.

All the scenes are captured using a StructureSensor. Please refer to the accompanying video for full sequences.

Figure 3.7-kitchen shows how occluded regions can be determined and completed effectively. The book stand, which is partly occluded by the cereal box is reliably infilled (due to fronto-parallel rectification) and still ensures that the background remains visible.

We demonstrate how we can deal with large regions of missing depth, due to

range limitation of consumer depth sensors, by allowing segments be approximated by a far away plane in Figure 3.7-park1 and Figure 3.7-park2 . In these scenes, we are still able to have a parallax effect with only two proxies in the scene.

The two statues in scenes park1 and park2, the pot in scene kitchen, and the table a chairs in scene office show how non-planar objects can be approximated by a plane primitive. By using the plane fitting error we can identify such objects and set their normal facing the camera and using the segments centroid. This does, however, lead to inaccurate perspective scalings in scene-office. Note that for non-planar objects, we can also add a degradation term for views deviating from fronto-parallel projection.

For each scene, we give examples of synthesized views with low and high degradation scores. Qualitatively, the degradation models captures image blemishes reasonably. For example, in the scene-office, moving the camera too far into the scene reveals a poorly infilled region that gets flagged by a high degradation score. In scene-kitchen, panning right and forward reveals a much smaller segment on high-texture infill, compared to panning right. Similarly with the scene-living room the poorly infilled floor is also flagged by the degradation model.

Some of the scenes required user-interaction for the segmentation step. Figure 3.7-office required the table legs to be highlighted to ensure the legs were segmented with the table top; Figure 3.7-kitchen required user interaction to ensure the orange tray was assigned to the back wall, not the wooden stand; and Figure 3.7-living-room required the sofa and chair to be tagged as separate objects.

Depth of Field. The primitive abstraction and depth can be used in creating a depth of field effect, see Figure 3.8. The user can control the camera's depth of field by setting a focus depth value and range: pixels within the depth of field remain the same but those outside are blurred with a Gaussian kernel. To get the complete depth of field effect the variance parameter for the Gaussian Filter is made dependent on the difference in each pixel's depth with the depth of field range.

Figure 3.8: Depth of Field: A DoF effect can be created using the primitives. The left image shows the first frame of parallax photograph with the book currently in focus. The right image shows the final frame with the checkerboard in focus. Throughout the sequence the camera's depth of field remains the same but as the camera moves forward the object in focus changes.

Limitations. Our approach works best when there are only a handful of intersecting primitives. In scenes such as Figure 3.9 where there are too many intersecting primitives in close depth proximity and appearance, we are unable to segment and fit primitives correctly. The problem is complicated as the noise level in the depth measurement is higher than the depth separation of the scene planes. The initial synthesis looks plausible for small view changes, but when the user makes bigger view change it reveals glaring artefacts breaking the illusion.

We only used planes as proxy geometry in our implementation. While we demonstrated that planes can solve many of the challenges, more complex primitives are likely to provide more interesting results. For example, cylinders, where appropriate, would provide more accurate occlusions and perspective changes as the camera moves.

3.5 Closing Remarks

We have presented an assistive tool that can aide artists in creating parallax photographs from single depth images. Central to creating this tool is understanding the signals present in both an input photograph and depth image. Understanding these multimodal signals allows us to predict occlusions and ordering between im-

Figure 3.9: Failure Case: For this input scene (left) where there are many intersecting objects we are unable to accurately fit primitives and determine occlusions (right). Without accurate proxies we are unable to correctly complete the scene or synthesise new views.

age patches, complete occluded regions, and anticipate image-level changes under camera movement. We argue that empirically we demonstrate that artists can more easily generate accurate parallax photographs from a single depth image by simply selecting two key frames, rather man manually creating each frame in the sequence. We additionally qualitatively evaluate the accuracy of the image degradation model. We argue that the tool meets the objectives set out for assistive tools, however, more vigorous evaluation via a user study could support this more strongly.

Subsequent Work. Since this work was published there have been several advances in semantic segmentation [11, 12, 13, 14, 15, 16] and image completion [17, 18]. In future work we would like to evaluate the image degradation model using these more advanced algorithms in our pipeline. Potentially, the improved segmentation could help with more accurate primitive fitting. Moreover, with improved data-driven image completion, it would be interesting to further evaluate the accuracy and usefulness of our image degradation model.

Next, in Chapter 4 we explore a very different type of assistive tool that generates sketching tutorials for novice artists. To enable this tool we use a reference scene of a 3D model and through some geometry processing generate alternative representations that can be used to enable our assistive tool.

Chapter 4

Generating Easy-To-Follow Tutorials for Sketching 3D Objects

Accurately drawing 3D objects is difficult for novices, as it requires an understanding of perspective and its effects on geometry and proportions. Step-by-step tutorials break the complex task of sketching an entire object down into easy-to-follow steps that even a novice can follow. However, creating such tutorials requires expert knowledge and is time-consuming. As a result, the availability of tutorials for a given object or viewpoint is limited. In this chapter we present an assistive tool How2Sketch (H2S) to address this problem. H2S automatically generates easy-to-follow sketching tutorials for arbitrary 3D objects. Given a segmented 3D model and a camera viewpoint as an input reference scene. H2S computes a sequence of steps for constructing a drawing scaffold comprised of geometric primitives, which helps the user draw the final contours in correct perspective and proportion. To enable this we use multiple representations of the input reference scene, a 3D model. Specifically, we generate different possible candidate variations of scaffolding primitives that approximate the segmented 3D model. The algorithm analyses the candidate scaffolding primitives and solves for an ordering among them that is easy-to-follow. Specifically, H2S explicitly wants to allow small geometric modifications to the size and location of the object parts to simplify relative positioning. Technically, we formulate this scaffold construction as a single selection problem that *simultaneously* solves for the ordering and geometric changes of the primitives.

4.1 Introduction

The ability to draw real-world objects is a useful and important skill across many disciplines. Product designers draw daily as they generate and refine product ideas, fine artists may spend hours in figure drawing classes learning how to replicate a shape from the real world, while hobbyists use sketches for visual expression. Still, sketching requires skill and practice. One of the major challenges in drawing real-world objects is learning to draw *what you see* rather than *what you know* [95]. A simple cylinder, for example, is *known* to have a circular cross-section with equal widths at the top and bottom. However, when we actually *see* a cylinder, it is subject to perspective distortion: circles become ellipses while projected radii diminish with distance from the viewer.

Tutorials are commonly employed to teach novices how to draw a specific object using correct drawing practices. Manual authoring such tutorials requires significant expertise and time commitment even for trained artists. Consequently, objects and viewpoints in existing tutorials tend to be limited and are chosen by the expert, rather than the users of the tutorials. To address these issues, we present an approach for *automatically generating easy-to-follow tutorials for drawing part-segmented 3D models from user specified viewpoints.* Figure 4.1 shows parts of a

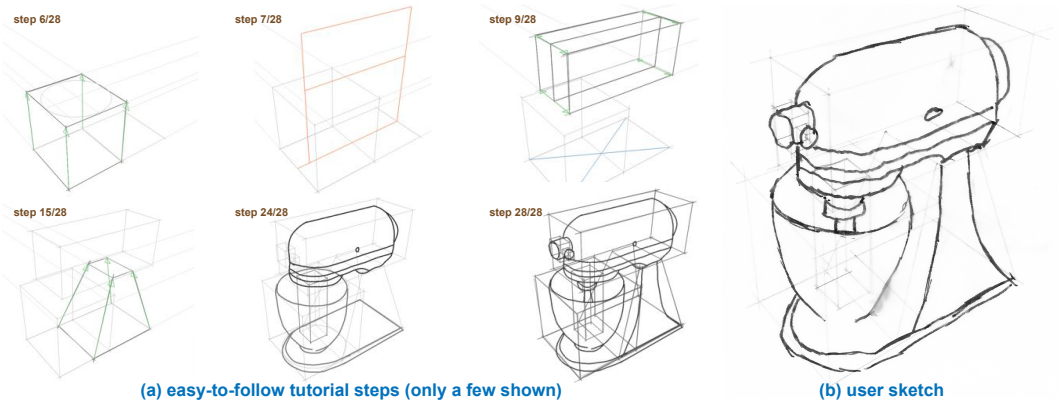


Figure 4.1: (a) We present How2Sketch, a system that automatically generates easy-to-follow tutorials for drawing 3D models. Each generated tutorial presents a list of steps for drawing scaffolding primitives that help the user draw the object in correct perspective. To help the user draw the scaffolding, the tutorial shows how to construct guidelines that anchor object parts relative to one another. User study feedback on the tutorials indicates that users feel they are able to create more accurate drawings (b).

tutorial generated by our system and the drawing by one of our study participants based on that tutorial. Our algorithm targets man-made objects where part relations and proportions tend to be inherently meaningful and crucial for accurate depiction.

Inspired by instructional books and online tutorials, we take explicit steps to make a sketching tutorial *easy-to-follow*:

i. Focus on accurate inter-part proportions and *relations* via a drawing scaffold, followed by detailing of the object contour;

ii. Proceed in a *coarse-to-fine* fashion, where object parts are abstracted as primitives (e.g., cuboids, cylinders) over several levels of detail to build up said scaffold;

iii. Propose a particular drawing *order* among the scaffolding primitives such that those sketched later can be easily anchored (i.e., drawn with guidance) off already drawn primitives; and

iv. Provide explicit steps for the construction of *guidelines* to accurately anchor the scaffolding primitives.

Our key observation is that in easy-to-follow tutorials the dimensions and arrangements of object-parts tend to have ratios that are easy to construct. For example, it is easier to construct the center line of a rectangular face compared to its one-fifth line. Tutorial authors choose to construct with such 'easy ratios' to simplify the drawing process and to focus on the procedure, rather than incidental and arbitrary measurements (see Figure 4.2). To apply this technique to existing objects, How2Sketch proposes small geometric changes while keeping overall deviations from the source model minimal. Since in each step new primitives and guidelines are anchored with respect to those drawn in the previous steps, the ordering of steps significantly affects the simplicity of ratios that can be employed, and the incurred geometric approximations. This tight interdependence between ordering of primitives and their geometric changes makes the problem non-trivial. A further challenge is to preserve the original inter-part relationships of objects, even

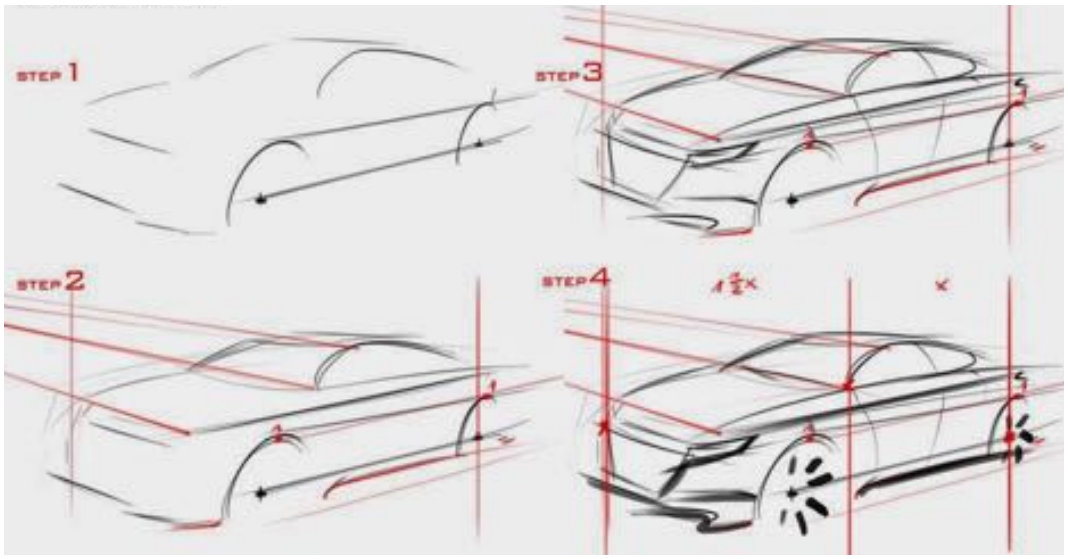


Figure 4.2: A step-by-step sketching tutorial for drawing a car, ©Czajkowski. The task is made simpler by breaking it into steps and by providing guidance about part proportions and alignments.

under geometric perturbations. For example, in Figure 4.1 the coaxial relationship between the mixer bowl and mixer blade is preserved.

Technically, we map the geometric adjustment and ordering of parts to a single selection problem. We first generate a set of potential candidate primitives by enumerating different anchoring possibilities. Since such anchoring requires drawing guidelines, and some guidelines are easier to construct than others, the algorithm prefers anchoring possibilities that rely on easy-to-construct guidelines, such as the top edge, bottom edge, center line, etc., of existing primitives. Our key insight is that the problem of geometric adjustment and ordering of parts can be simultaneously solved by *selecting* an appropriate subset from the candidate primitives, in order to balance between geometric changes and ease of constructing necessary guidelines.

We test our algorithm on a range of examples and evaluate our algorithmically generated easy-to-follow tutorials with a user study, which finds that H2S tutorials can help both with objective as well as perceived accuracy of sketches, and are easier to follow.

4.2 Learning How to Sketch

To inform the design of How2Sketch we studied several drawing books [95, 96, 97], consulted various sketching websites (e.g., Sketch-A-Day [98], Draw-A-Box [99]), carried out an expert interview with a professional artist, and participated in a drawing course.

Through this process we found that effective tutorials for drawing 3D objects typically include the following:

- *Parts are approximated by geometric primitives:* Plane, cubes and cylinders are heavily used to approximate shapes. They are easy to construct and verify visually.

- *Steps are coarse-to-fine:* First, the overall object is scaffolded with approximate shapes, followed by finer contour details. Primitives are drawn sequentially, in optimized order.

- *Anchor shapes to each other:* Shapes are drawn with respect to previously drawn shapes, to aid with correct placement and proportions. Instructions for positing shapes relative to each other use simple measurements (e.g., draw box *half way* down the side, draw circle in the *center* of the rectangle), etc.

- *Vanishing lines for perspective:* Vanishing points are explicitly indicated to aid the user to draw correctly.

How2Sketch supports the above tutorial features as follows:

(a) Scaffolding primitives. How2Sketch utilizes scaffolding primitives to geometrically approximate each segmented object part. The system supports planes, cuboids, cylinders, and truncated pyramids, as they allow for planar guidelines to be used, which are simple to construct, and cover a wide range of shapes. (Note that in our visualization, cylinders are shown as axis-aligned bounding boxes, since the box faces are used for providing guidance for drawing ellipses for cylinder caps.) In addition to scaffolding, we guide users in drawing ellipses to better approximate

some shapes.

(b) Ordering. Our algorithm provides the relative ordering of the scaffolding primitives. Further, How2Sketch offers detailed, sequenced instructions for constructing primitives.

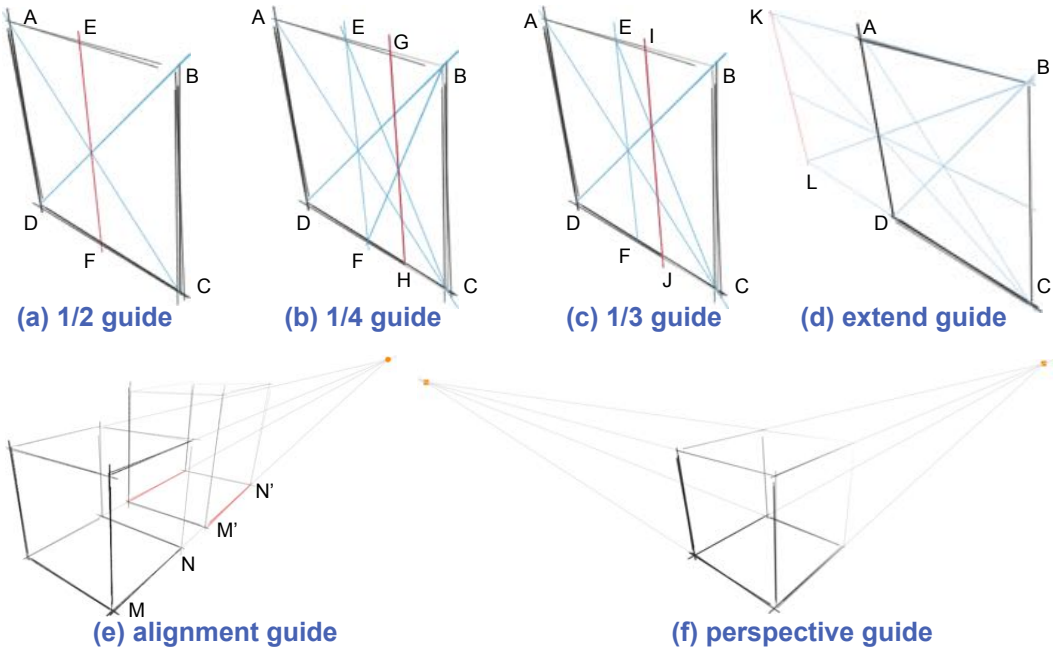


Figure 4.3: Our system supports different forms of guidelines for drawing coplanar proportions (a-d), for anchoring alignments (e), and for previewing 2-point perspectives (f). Please refer to Sec. 4.2.

(c) Placement, alignment, and proportions. We support a set of coplanar guidelines (see Figure 4.3). Given a face $ABCD$, its diagonals help construct the ½ line EF (Figure 4.3a). Two levels of ½ lines produce a ¼ line GH (Figure 4.3b); while intersecting a diagonal BD with line CE produces a ⅓ line IJ (Figure 4.3c). Similarly, we support extrusion towards a vanishing point as in Figure 4.3d where $ABCD$ is extended by reflection to form $BCLK$ such that $AB = AK$. Finally, we also support alignment, as in Figure 4.3e, $M'N'$ is aligned with MN.

(d) Perspective. To provide perspective information, we show the vanishing points (if within the drawing area) and also show the vanishing lines leading to them (Fig-

ure 4.3f). How2Sketch supports sketching in 2-point and 3-point perspective.

4.3 Generating Sketch Sequences

Given a 3D object (S) segmented into parts and a desired viewpoint, our goal is to establish an easy-to-follow sequence for drawing the object, starting with the scaffolding and progressing to the contour details. We make it easier to draw the scaffold by actively making small part-level geometric changes to facilitate relative anchoring using a set of guidelines.

As described in Section 4.2, we have adopted simple procedures to accurately draw guidelines at easy-to-construct ratios (½, ⅓, ¼, 1×, 2×, etc). Object part positions and sizes in the original models, however, rarely conform to such ratios. Hence, we propose to modify object parts, so that they end up with part relationships that are easy to draw. We motivate this choice twofold: (i) Scaffolding primitives in tutorials like those generated by H2S are already approximations of real geometry and thus contain a measure of error. Some of this error can actually be compensated by adjusting the fit of contours within the scaffold. (ii) Accurate estimation of lengths and ratios is difficult, even for experts, so errors are almost unavoidable. By enforcing that parts relate via simple ratios for which reasonable geometric constructs can guide the user H2S can minimize per-part error and make better global decisions about how to distribute the overall error.

Our algorithm proceeds in three main stages (see Figure 4.4): (i) generating part-level primitives and encoding inter-primitive relations; (ii) creating primitive

Table 4.1: Notation table.

symbol	denotes
S	input part-segmented model
P_i	primitive corresponding to the i-th part of S
$R_{i,j}$	relation between primitive pairs (P_i, P_j)
C_j	Parent candidate that can be used for anchoring
$C^k_{j\to i}$	candidate for the i-th part primitive with (anchoring) parent from the j-th part primitive, where k denotes the k-th such instance
$\mathscr{C}_{*i}$	set of all the candidate primitives generated for part primitive P_i
$\chi(X)$	indicator variable corresponding to the selection of X
Λ	assignment of indicator variables denoting a set of selected candidates

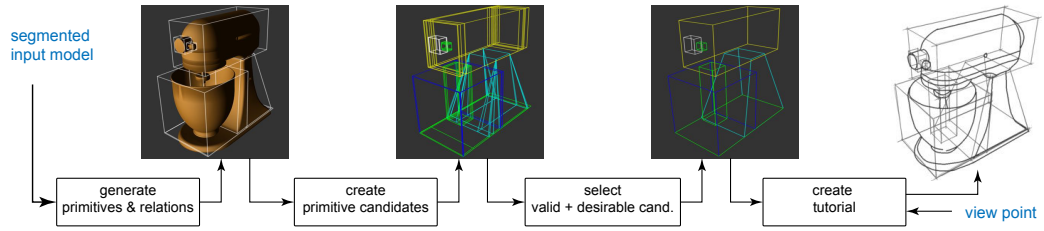


Figure 4.4: System Overview. Starting from a segmented input model and a user-specified viewpoint, How2Sketch generates easy-to-follow step-by-step tutorials. The system automatically makes subtle geometric modifications to simplify the tutorial steps.

candidates based on various inter-primitive anchorings strategies; and (iii) selecting a valid and desirable set of primitives among the candidate selections. The result implicitly encodes how to geometrically modify each part (both their dimension and placement), and in which order to draw them. Intuitively, our algorithm produces an easy-to-follow primitive drawing sequence at the cost of deviating from the original geometry in a controlled fashion. We now elaborate each step. Please refer to Table 4.1 for symbols used in the following.

4.3.1 Generating Primitives and Inter-part Relations

H2S takes as input a pre-segmented 3D model and abstracts the model parts with primitive shapes. In our implementation we support planes, cuboids, cylinders, and truncated pyramids (see Figure 4.5a). For each part of the input model S, we use least-squares to fit different axis-aligned primitive types and take the one with the smallest residual. In case of ties, we prefer the simpler primitive. We denote the primitive for the i-th part as P_i (the type of primitive is not explicitly indicated in this notation).

Man-made objects often have dominant inter-part relations. We found it desirable to preserve such relations in the generated tutorials. Hence, we first detect such inter-part relations and later preserve them in the generated tutorials. We simply test (see [100]) each pair of primitives P_i and P_j for any (supported) relations. In our implementation, we handle coplanar, coaxial, and common bisector plane relations. In case of multiple relations between a pair of primitives, we prefer common bisector plane over coaxial over coplanar. We represent a relation using a binary variable

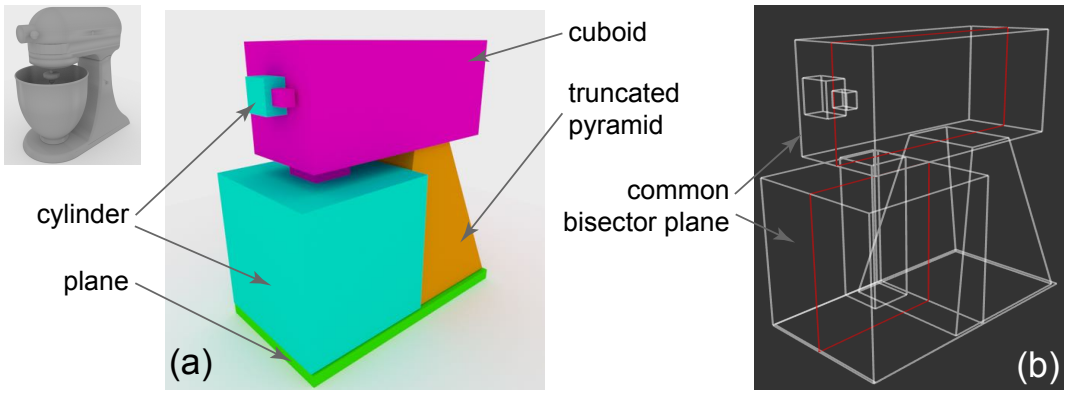


Figure 4.5: Given a part-segmented input model S (top-left inset), the system abstracts the parts as different primitives (a) and identifies inter-part relations. For example, here the mixer bowl and mixer head primitives share a common bisector plane.

$R_{i,j}$ where i and j respectively denote the primitives P_i and P_j (type of relation is not explicitly indicated in this notation). If a relation is present, we mark $R_{i,j} = 1$ and $R_{i,j} = 0$ otherwise. Figure 4.5 shows some examples.

4.3.2 Creating Candidate Primitives

We now describe the candidate primitive generation step that creates additional primitives based on possible anchoring strategies. We use $\mathscr{C}_{*i}$ to denote the set of all the candidate primitives generated corresponding to the primitive part P_i. Since the original primitives (e.g. P_1, P_2 and P_3 in Figure 4.6) are always candidates, we start with $\mathscr{C}_{*i} := \{P_i\}$. We generate candidate primitives in three stages:

(i) For each pair of primitives P_i and P_j, we generate candidates of the form $C^k_{j\to i}$, where $j \to i$ indicates that a candidate is generated for primitive part P_i and is anchored off P_j with k denoting different anchoring possibilities. For example, parts can be anchored based on different guidelines described in Section 4.2 for different face- or plane-based anchors. We append these candidates to the respective candidate sets as: $\mathscr{C}_{*i} \leftarrow \mathscr{C}_{*i} \cup \{C^1_{j\to i}, C^2_{j\to i}, \ldots\}$ (see Figure 4.6b).

(ii) The small part modifications introduced during anchoring in step (i) may violate some of the relations $R_{i,j}$. For each pair of primitives (P_i, P_j) sharing a relation, we add additional primitives to their candidate sets to restore the relations. Specifically, corresponding to a candidate of the form $C^k_{j\to i}$ (created in stage (i)), we create a new candidate of the form $C^{k'}_{i\to j}$ such that $C^k_{j\to i} \leftrightarrow C^{k'}_{i\to j}$ are similarly related

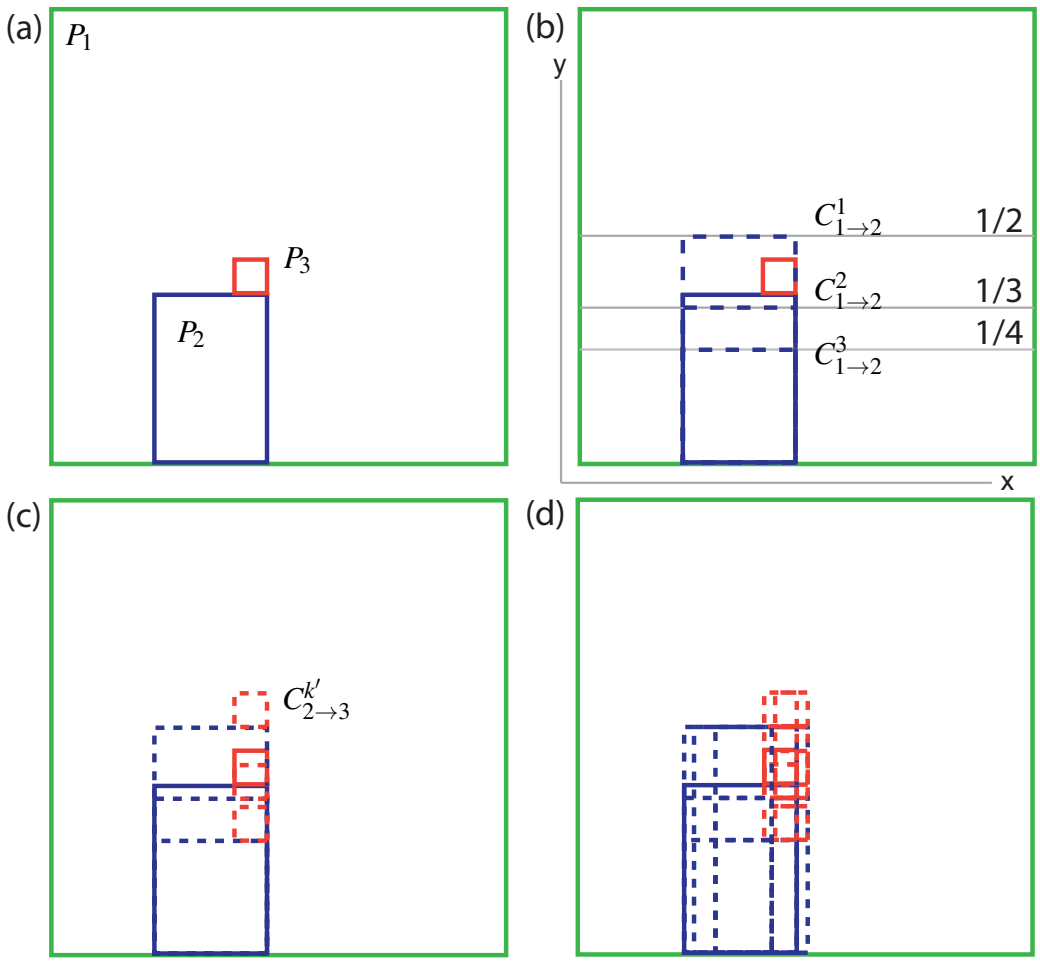


Figure 4.6: (a) Starting from initial primitives P_1, P_2, P_3, for each pair of primitives we generate several adjusted primitive candidates. Candidates are generated for each axis independently, for example, in (b) we show how using P_1 as a parent several P_2 candidates are created by aligning its top edge to the 1/2 ($C^1_{1\to2}$), 1/3 ($C^2_{1\to2}$) and 1/4 ($C^3_{1\to2}$) guides on the y-axis. (c) As P_2 and P_3 have a co-planar relation for each $C^k_{1\to2}$ candidate a new P_3 candidate ($C^{k'}_{2\to3}$) is generated restoring this relation. This process is repeated both in the other dimensions and to generate second-level candidates (see Section 4.3.2) to generate the full set of candidates (d). This is an illustrative figure in 2D with only some of candidate primitives shown.

as in $P_i \leftrightarrow P_j$. We append all such relation-based additional candidate primitives to the respective candidate sets, i.e., $\mathscr{C}_{*i} \leftarrow \mathscr{C}_{*i} \cup C^{k'}_{i\to j}$ (see Figure 4.6c).

Note that in the above a candidate is allowed to be anchored from one or multiple parents, as each axis can be independently anchored. Additionally a candidate can be partially unguided (e.g., the width and length of cuboid is guided but the

height is not) or completely unguided (e.g., it is simply the input primitive) (see Figure 4.6). We defer further details to the implementation section. (iii) We allow second-level candidates, i.e., candidate primitives as generated above are allowed to act as anchors for other primitives creating a hierarchy. To this end, we simply iterate one more time stage (i) and (ii) (e.g., in Figure 4.6d candidates $C^k_{1\to 2}$ create second-level candidates for P_3). Note that before starting this step, we remove the undesirable candidate primitives with large changes in geometry or relative placements (more details in the implementation section).

At the end of this stage, we have a set of candidates for each part P_i of the input model resulting in the super set of candidate primitives of the form $\{\mathscr{C}_{*i}\}$ (see Figure 4.7).

4.3.3 Selecting Candidate Primitives

Having generated multiple candidates, our remaining task is to select a set of valid and optimal candidates, as explained next.

Valid candidate sets. We first characterize the notion of valid selections. We use indicator variables $\chi(X)$ to denote if a candidate primitive X is selected (i.e., $\chi(X) = 1$) or not (i.e., $\chi(X) = 0$). We have $\chi(C^k_{j\to i}) \in \{0,1\}$ for each $C^k_{j\to i} \in \mathscr{C}_{*i}$. Let Λ denote a particular assignment for the indicator variables for *all* the candidate primitives.

Among the various possible selections, not all the subsets of candidates of the form Λ constitute *valid* selections. A valid selection of candidates should satisfy three conditions:

i. For each part of S, exactly *one* candidate primitive should be selected;

ii. If a selected candidate primitive is anchored off one or more parent (candidate) primitives, then its parent primitive(s) *must* also be selected;

iii. If any two primitives P_i and P_j share a relation, then their corresponding selected candidate primitives should also respect the same relation.

We now express the above conditions in terms of the indicator variables in Λ.

(a) We encode (1) as

$$\sum_{j,k} \chi(C^k_{j\to i}) = 1 \qquad \forall i. \tag{4.1}$$

(b) We encode (2) as a quadratic constraint involving the binary selection variables as

$$\chi(C^k_{j\to i})\chi(C_j) - \chi(C^k_{j\to i}) = 0 \tag{4.2}$$

for each dependent pair $C^k_{j\to i} \in \mathscr{C}_{*i}$ and its parent C_j. Note that this condition *disallows* $\chi(C^k_{j\to i}) = 1$ AND $\chi(C_j) = 0$, but allows any of the other three assignments involving $\chi(C^k_{j\to i})$ and $\chi(C_j)$.

(c) We now encode (3). Let two primitives P_i and P_j share a relation, i.e., $R_{i,j} = 1$. Let $\mathscr{C}_{*i} = \{C^1_{*i}, C^2_{*i}, \dots\}$ be all the generated candidates for primitive P_i and similarly $\mathscr{C}_{*j} = \{C^1_{*j}, C^2_{*j}, \dots\}$ for primitive P_j. Then for *each* pair of the form $C^k_{*i} \in \mathscr{C}_{*i}$ and $C^{k'}_{*j} \in \mathscr{C}_{*j}$ that does *not* share the relation $R_{i,j}$, we require

$$\chi(C^k_{*i})\chi(C^{k'}_{*j}) = 0. \tag{4.3}$$

This condition disallows $\chi(C^k_{*i}) = 1$ AND $\chi(C^{k'}_{*j}) = 1$, i.e., candidate primitives that do not share the same relation as their primitive parts cannot be jointly selected.

Thus, a selection Λ is valid if and only if Equations 4.1-4.3 are all satisfied. Among all such valid selection sets, we next determine which one is the most desirable. Figure 4.7 shows a set of candidate primitives and a valid selection. Note that as each candidate primitive has a unique id and anchoring hierarchy, the constraints prevent dependency loops from being created.

Sequencing sketching as a selection problem. We balance the error due to mak-

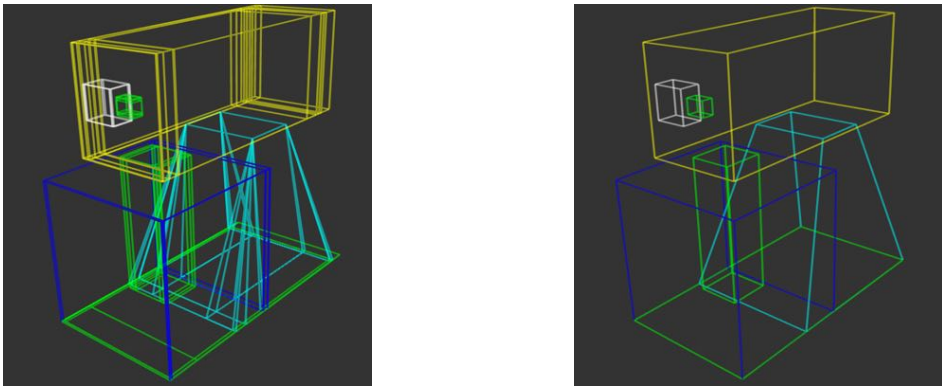


Figure 4.7: From a set of candidate primitives (left), our algorithm selects a subset of primitives that is *valid* and *desirable* as shown on the right. The selection implicitly encodes in which order to draw the primitives and also how to change each primitive (size and/or placement) such that the resulting tutorial is easy to construct. Please refer to the text for details.

ing changes to the geometry with the difficulty of drawing arising from anchoring. Specifically, we consider unanchored parts to be most difficult to sketch. Further, among the anchored ones, we consider a primitive easier to draw if it requires fewer guides. We model this difficulty of drawing as the cost $E_e(C^k_{j\to i})$ with a lower cost denoting *easier to draw* (see *"Error functions"*, below). The total cost is expressed as:

$$E_{\text{difficulty}}(\Lambda) := \sum_{i,j,k} \chi(C^k_{j\to i}) E_e(C^k_{j\to i}). \tag{4.4}$$

Selecting any primitive, however, incurs an associated error that we indicate as $E_d(C^k_{j\to i})$ due to deviation from original geometry. So, the total data cost of selecting a set of primitives is:

$$E_{\text{adjust}}(\Lambda) := \sum_{i,j,k} \chi(C^k_{j\to i}) E_d(C^k_{j\to i}) \tag{4.5}$$

with a higher cost indicating larger geometric deviations from the original parts.

Thus, we arrive at the final formulation for *desirable* selection as,

$$\Lambda^\star := \arg\min_{\Lambda} (E_{\text{adjust}}(\Lambda) + E_{\text{difficulty}}(\Lambda)) \tag{4.6}$$

subject to Equations (4.1)-(4.3) to ensure a valid selection. Thus, we have formu-

lated our problem as a quadratically constrained linear program.

Error functions. The above formulation requires metrics for E_e and E_d. We use the following metrics in our implementation.

For the difficulty of drawing term $E_e(C^k_{j\to i})$, we associate a higher cost for anchors that are harder to replicate (e.g., requiring more construction lines). Specifically, we set the cost to the number of guidelines divided by the area of the parent plane where construction lines are to be drawn. This encourages fewer guides but also using planes/faces with larger areas for drawing sketch guides. (The effect of viewpoint is only considered at runtime as discussed in Section 5.4).

For the data error $E_d(C^k_{j\to i})$, we sum the changes in length along each axis, normalized by the original axis length, with the translation of the midpoint of each axis, again normalized by the input axis length. For an unguided axis we set the data error to the maximum of 2 to discourage unguided candidates.

Final drawing order. The solution to the above optimization directly gives us both the *ordering* and the *size and location modifications* of the parts. The ordering is represented as a directed graph, and we gain the final linear ordering via topological sorting. Note that the directed graph may have a fork in the ordering of candidates primitives. This implies that the relative drawing order of certain primitives are not specified. We break such ties only at runtime once the user selects a view (see Section 5.4).

4.3.4 Implementation details

We now clarify some implementation details. The 3D models were downloaded (e.g., from Turbosquid) and manually part segmented (if part level segmentation was missing). Segments that are not well approximated by one of the primitives described above can be represented as a custom primitive (e.g., line) but such primitives are excluded from our optimization step. Instead, their positions are updated after optimization by enforcing existing relational constraints with optimized primitives.

The candidate primitive generation works in two steps: first, we use the coplanar relations to generate candidate planes $c^k_{i\to j}$, and then depending on the primitive type we combine the planes to create a complete primitive $C^k_{i\to j}$ (here, lowercase c indicates a candidate plane rather than a complete primitive, C). This choice unifies candidate primitive generation across primitive types (recall cylinders are processed based on their axis-aligned bounding box).

For each pairwise coplanar relation $R_{i,j}$ we have two participating planes in P_i and P_j: at this stage the relation is undirected and we produce candidate planes using both combinations $c^k_{j\to i}$ and $c^k_{i\to j}$. To generate a candidate plane, each axis is considered independently then all combinations of axis pairs are used to create planes $c^k_{i\to j}$. An axis can be anchored by the parent plane using the end points of the same axis. This means there are several anchoring possibilities. For example, anchoring the vertical axis of P_i on P_j might involve anchoring the top edge of P_i to the ⅓ line of P_j and the bottom edge P_i to the bottom edge of P_j. An alternative might be to anchor the top edge of P_i to the ⅓ line of P_j and the bottom edge P_i to ¼ line of P_j. We initially generate all such candidates but to reduce the number of candidates to select from we discard those where an axis length or translation change by more than 10% of the input length.

Having generated all the candidate planes using all the pairwise relations, we generate complete primitives by combining the different planes based on the primitive type. To generate a complete cuboid primitive, for example, we find the missing height axis from one of the other planes to complete the primitive. For truncated pyramids we combine top and bottom planes with a height axis to make a truncated pyramid. Finally, we repeat this process but use the first level candidates as the parent primitives to generate second level candidates.

We use the Gurobi Solver [101] to solve the quadratically constrained LP as described above. Typically the solver takes 1-2 minutes in the presented examples.

4.4 Presenting Sketch Sequences

The sequence generated in Section 4.3 provides primitive ordering, sketching guidelines, and adjusted part geometry for drawing the scaffolding of the object. H2S tutorials can be adapted further based on the user chosen viewpoint and user indicated drawing level (novice/apprentice/master), which can be controlled interactively. Our custom viewer indicates when guidelines can be erased and provides hints for drawing in perspective and object contours.

Viewpoint. We use the user specified viewpoint to customize the tutorial as follows: (i) Although primitive ordering is determined based on anchoring strategies, multiple primitives can anchor from the same parent, resulting in a tie. We break such ties by first choosing the primitive that is closest to the user from the indicated viewing position. (ii) The selected viewpoint can make some guidelines cumbersome to draw because of limited space on the projected area of a primitive face. We identify such instances by thresholding based on A_p/k, where A_p indicates the projected area and k the number of guides necessary to draw the primitive. If a primitive falls below a threshold of 0.01, we ask the user to simply 'eyeball' the primitive without drawing intermediate guides. (iii) Finally, a segment that is occluded and its primitive does not help anchor any other visible primitive is deemed unnecessary and hence is left out from the generated tutorial.

User ability. We adapt our tutorials to different sketching abilities by classifying

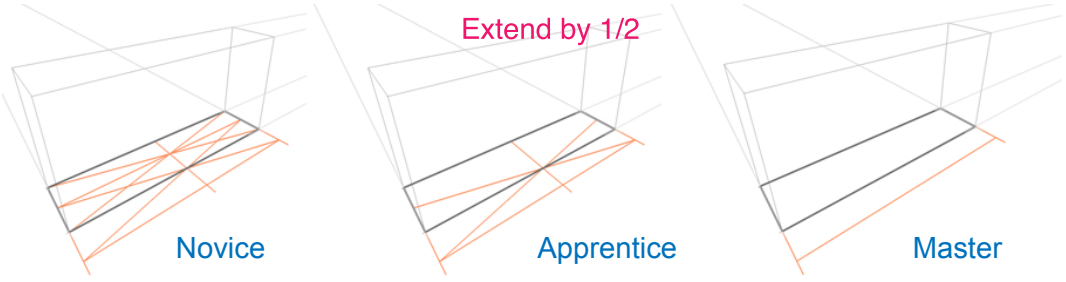


Figure 4.8: User ability. The user specify a preferred drawing level (novice, apprentice, or master) which determines the number of intermediary guides presented for each step. For the 'extend by 1/2' step, novices (left) are shown 9 guidelines, apprentices (center) 6 guidelines, and masters (right) 3 guidelines.

Figure 4.9: Guide lifetime. Guides first appear in orange (left). In subsequent steps guides that are no longer required are removed, while those that are to be reused are marked in blue (middle, right).

the various guidelines as suitable for novice, apprentice, or master users. For example, dividing a face of a primitive into halves requires three guidelines. A novice is shown all the three, an apprentice only the ½ line itself, and a master is not provided with any intermediate guidance. Note that in all cases, the user is instructed to divide the highlighted face into half by a text label in the viewer (see Figure 4.8).

Guide lifetime. In order to reduce the amount of guidelines on a sketch at any point in time, we determine each guide's lifetime to inform the users when a guide can be safely erased. To this end, we first go over the list of generated guidelines to identify the equivalent ones, and store their *lifetime*, i.e., when they first appear and when they are last used. During the tutorial, a guideline is drawn in orange when it first appears. If the guideline is used in any later step, it is changed to blue. After the last step a guide is used, it is no longer shown. As a result, users do not have to unnecessarily erase/redraw guides, which helps to reduce clutter as they sketch (see Figure 4.9).

Vanishing points and ellipses. Vanishing lines and vanishing points are indicated with respect to the paper boundary (shown as green corners) to help users better position the lines. We additionally guide users in sketching ellipses on a primitive face by using guides to the vanishing points. These guides intersect with the edges of the face at the perspective mid-points, which are the points where the ellipse should touch the face of the primitive.

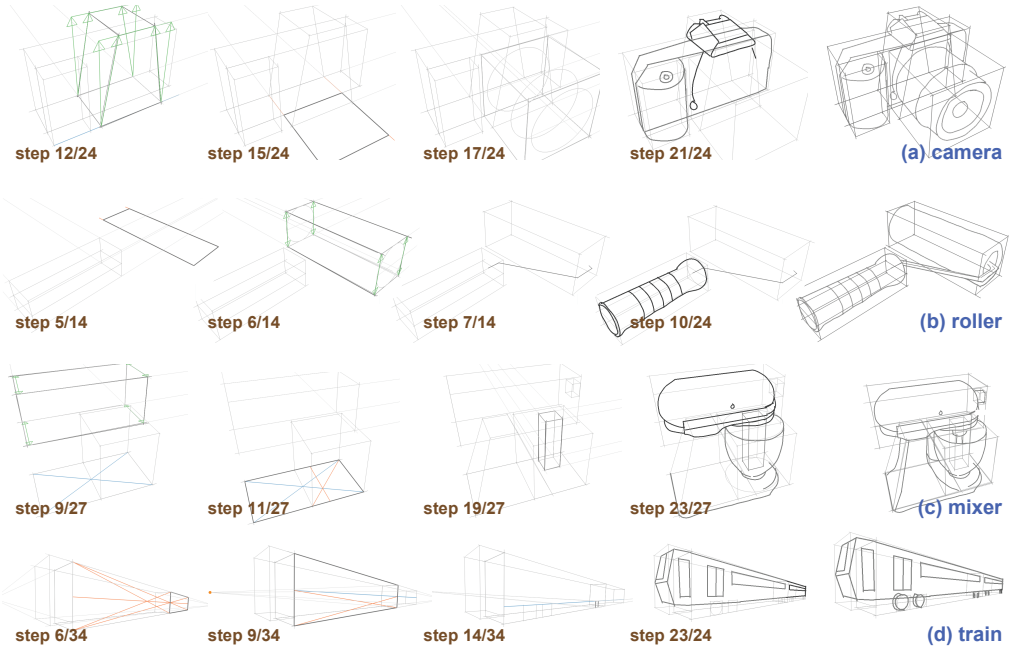


Figure 4.10: Example step-by-step tutorials generated by our system: (a) and (b) were generated in the master-user setting, while (c) and (d) were generated in the novice-user setting. Please refer to the supplementary materials for complete examples.

Contour ordering. Once the user has sketched the scaffolding and ellipses, we guide them to sketch the contours. We progressively display contours on the modified underlying model segments, following the order determined by the primitives. Already drawn parts are used to determine occlusion for the new primitives, thus reducing clutter (see Figure 4.1).

Interface. H2S tutorials can be presented in a few different forms. They can be navigated using an interactive interface, they can be printed (see supplementary material on our project webpage[1]), or sequenced into a tutorial video. Text instructions can be synthesized, as needed.

[1] http://geometry.cs.ucl.ac.uk/projects/2017/how2sketch/

4.5 Results and Discussion

We used How2Sketch to generate sketching tutorials for four man-made objects - a Digital SLR Camera, Kitchen Mixer, Train, and Paint Roller. For these models, numerous tutorials depending on viewpoint and user ability can be generated. Parts of the tutorials are shown in Figure 5.7 (see Appendix A and supplementary material on project webpage for full sequences). We encourage the readers to compare How2Sketch tutorials with existing online tutorials (e.g., Draw-A-Box [99]). Each tutorial takes between 15 and 45 minutes (across the users who used the system) to complete due to their varying complexity. The small changes made to the input geometry by the method are illustrated in Figure 4.11. As desired, the alterations to geometry are subtle but now enable simple anchoring strategies based on the altered segment bounding boxes (also shown).

As demonstrated in Figure 4.10, our tutorials follow a coarse-to-fine strategy, starting with a single primitive that can be used to anchor subsequent primitives. Figure 4.10a shows excerpts from a tutorial sequence with master-user ability. Here, the grip of the camera is anchored on the edges of the camera body and a ¼ guide. Additionally, the grip and flash are both extended by one half the depth of the main body. The lens, an example of a second level anchoring, uses the flash for anchoring by extruding $1\times$. Guides for ellipses are provided before contours are drawn.

In the paint roller tutorial in Figure 4.10b the handle anchors the roller using the common bisector plane. The top edge of the roller is $1\times$ the length of the handle. The bottom edge is $\frac{1}{2}\times$ the length of the handle but due to the limited projected area and number of guides otherwise required, the step is unguided (as per Section 4.4).

Figures 4.1 and 4.10c both show novice-level tutorials for the food mixer but from different viewpoints. The plane primitive for the base of the mixer anchors the bowl using a planar relation and ½ guide. The common bisector plane between the base and the main body of the mixer is used for anchoring the length of the main body. The bisector plane is first drawn before being extended in both directions to create the cuboid primitive. The Mixer's stand is an example of a primitive with two parents, being anchored off both the main body and base. Difference in viewpoint

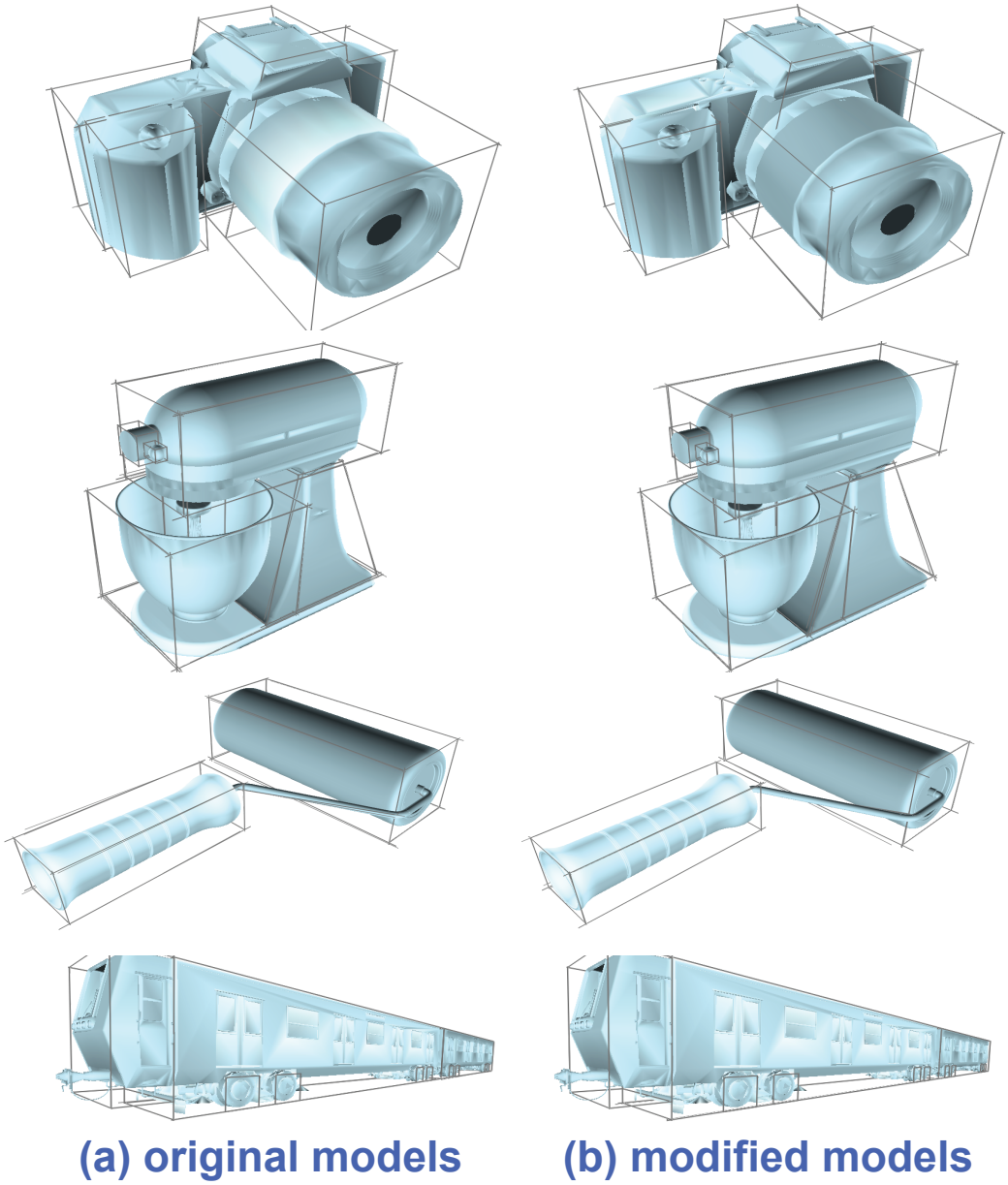


Figure 4.11: (Left) Original models. (Right) Subtle changes proposed by our algorithm in order to make the objects easier to draw.

between the two tutorials means that as one of the knobs is occluded from the view chosen in Figure 4.10c, it is omitted from the tutorial (see Section 4.4).

The train example, Figure 4.10d, anchors the second carriage as $1\times$ the length of the first carriage and the top edge of the wheels using the ¼ guide on the vertical axis of the first carriage. The driver's compartment is unguided.

Limitations. H2S only makes small changes to the input geometry. However, small gaps between object parts can have important semantic meaning. An example of this can be seen in Figure 4.11 where the main body of the mixer and the stand separate slightly in the adjusted version. We know these two segments would be joined by a hinge making such an adjustment unrealistic. Symmetry or regular structure can similarly be lost from the small geometry changes. An example of this is the roller in Figure 4.11, which ceases to be a perfect cylinder. Note that most of these violations are difficult to spot unaided and tend to get masked by drawing inaccuracies. Finally we find relations from the input segments but do not allow adjustments in geometry to create a relation that was not already present. In the future, we might enable such changes to allow for an even wider range of candidates.

4.6 Evaluation

To evaluate the effectiveness of the H2S tutorials, we compare to a simple step-by-step tutorial that shows scaffolding primitives for each part of the object but does *not* simplify the sizes or locations of the primitives to make them easier to draw. In this Basic tutorial type, the scaffolding primitives are shown in order from largest to smallest with a base primitive anchored to the ground plane. No guidelines are shown. Please see the supplemental materials for the complete tutorials used in the study.

Participants. We recruited 10 participants (ages 18-55, 6 men, 4 women) with varied expertise in drawing. Two participants reported negligible drawing experience, four reported drawing once in a while, and four reported drawing at least once a month. Three had taken college-level art classes or private/non-accredited art classes. When asked (free-form) what they found most challenging about drawing, 4 mentioned perspective, proportions, scale, and relative positions. When asked to rate their drawing skills on a scale of 1 (poor) to 5 (great), only 4 people rated their drawing skills above 2.

Methodology. In advance, each participant filled out an introductory questionnaire about their experience with drawing. Upon arrival, each participant was told that they will be asked to draw two objects, a camera and a mixer, using two different tutorials. Participants always followed a H2S tutorial first to disadvantage H2S to any learning effect. The two objects (camera and mixer) counter-balanced with half of the participants using the H2S tutorial type for the camera and half using the H2S tutorial for the mixer. The H2S tutorial was set to the *novice* ability for all the participants.

Before the H2S tutorial, participants were given a written handout (see supplemental material) that described how to draw construction lines for ½, ¼, and ⅓ guidelines and extending planes (see Figure 4.3). This written tutorial was designed to give them context for what they would encounter in the H2S condition.

Both the Basic and H2S tutorials were followed using a 13" laptop; participants used the trackpad to advance forward and backward through the tutorial. All drawings were done on paper. Each participant was given two pencils (HB, 0.3mm and 0.7mm). They were allowed to use a provided straight-edge and eraser. For creating each drawing, the participants were given a sheet of paper that included the vanishing points and the ground plane of the first primitive. This initial anchoring allowed us to easily compare drawings across users. All users drew the scaffolding primitives first on the calibrated paper. For drawing the final contours of the object, the moderator attached a transparent sheet to the paper with the scaffolding. This allowed for easier digitization and separately compare the contour drawings and the scaffolding primitives across users.

Participant filled out a questionnaire after drawing each object, indicating their level of satisfaction with their drawing (1 - not at all, 5 - very much), perceived accuracy of their drawing (1 - not at all accurate, 5 - very accurate), enjoyment with the tutorial experience (1 - not at all, 5 - very much), and ease of following the tutorial steps (1 - not at all easy, 5 - very easy). They also gave free-form responses about what they liked about each tutorial type and how it could be improved. At the end of the study, subjects were asked which tutorial type they preferred (Basic or

How2Sketch). We referred to the Basic tutorial type as the tutorial *without guides* and the H2S tutorial type as the *tutorial with guides*. Each participant was given a $25 gift card for his/her time.

User feedback. Nine out of ten participants preferred the H2S tutorial over the Basic tutorial. For each of the four questions, the users preferred H2S over the Basic tutorial (see Figure 4.12).

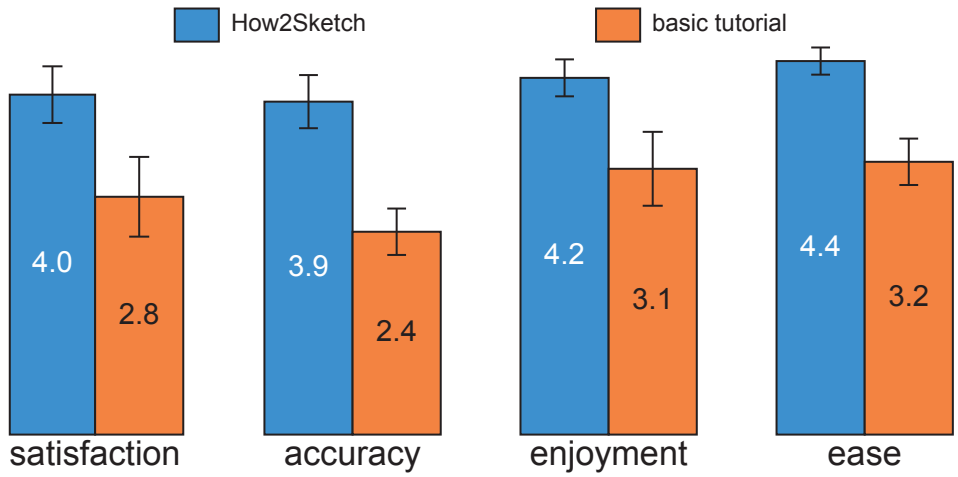


Figure 4.12: Average user ratings for satisfaction, perceived accuracy, enjoyment, and ease of following were all higher for the How2Sketch tutorials than for the basic tutorials. Showing standard error of mean (SEM) bars for $N = 10$.

An ANOVA across tutorial type and object drawn reveals a significant effect of tutorial type on accuracy and ease of following tutorial ($p<0.003$), significant effect on enjoyment ($p<0.034$), and marginal effect on satisfaction ($p<0.058$). The object drawn did not have an effect on any measure, despite their varying difficulty, and there was no interaction between tutorial type and object drawn.

Evaluating sketch quality. Figure 4.13 overlays the registered user sketches from the different conditions on the original model for the condition (e.g., H2S model after part level adjustments). While variation in contour placement is evident in both tutorial types, the variation in the basic tutorial sketches is greater.

In the camera tutorials the basic version starts with the ground plane for the lens and H2S with the ground plane of the main body. With this anchoring in the

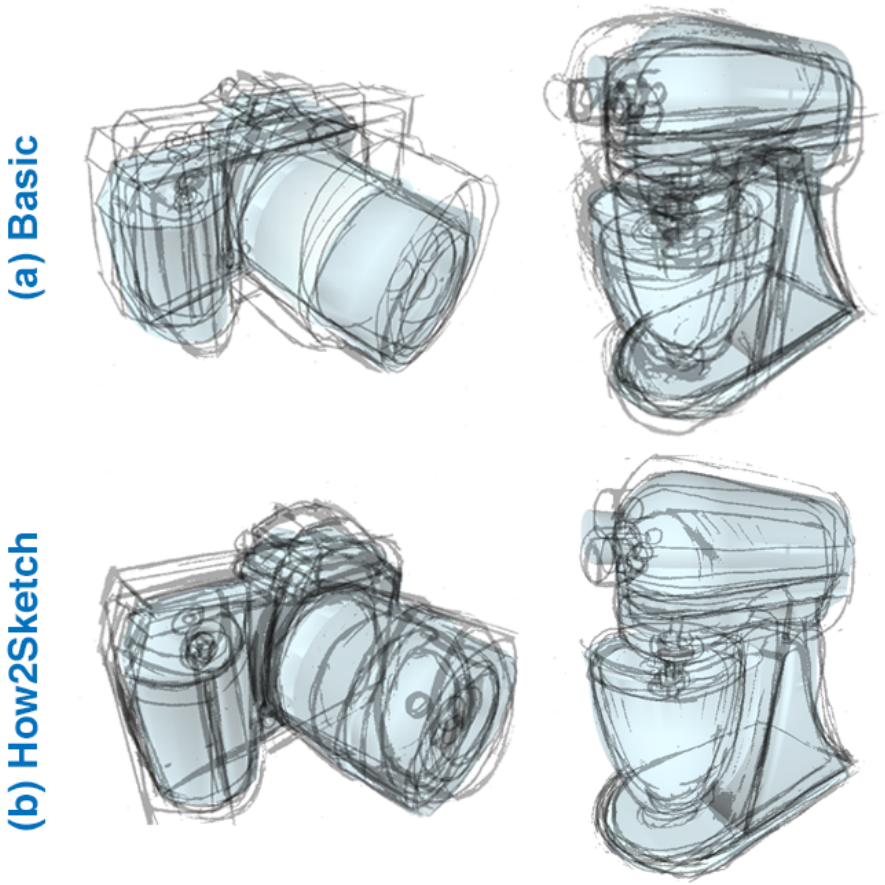


Figure 4.13: User Study Sketches: All the user sketches overlaid on the target objects. Sketches from following basic tutorials (top) show much greater variation in proportions and alignment than sketches from following H2S tutorials (bottom).

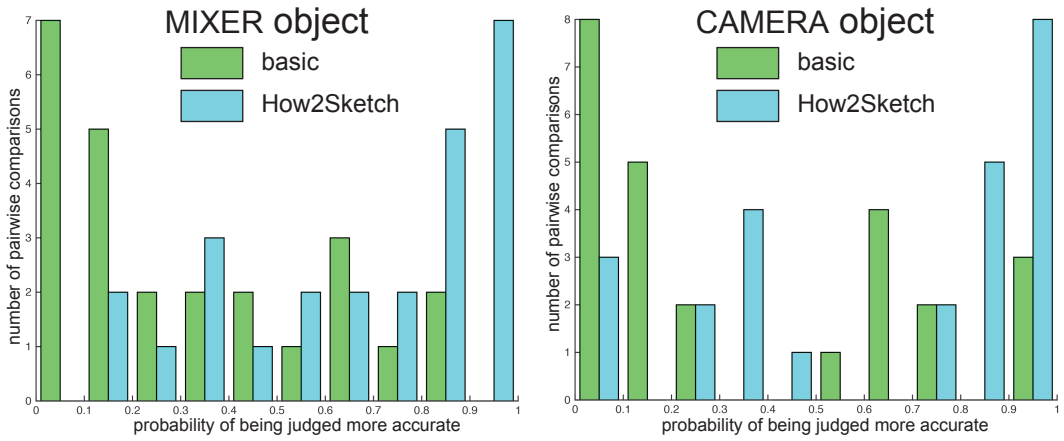


Figure 4.14: Bradley-Terry Model for the Mixer and the Camera Sketches produced by users of our tutorials and evaluated by another user study with Amazon Mechanical Turk rankers.

basic tutorial sketches, the width and length of the lens are accurate. However, the lens height and the other three primitives have a variety of errors in proportion and part placement. Comparing with H2S sketches, there are similar variations in the height of the main body. However, the guided steps for the grip and lens show reasonable consistency in positioning across the users. For the Mixer sketches - where both tutorials start with the base plane of the mixer - there is much more consistency of object part placements across users with the H2S.

As further validation, we conducted an additional user study using Amazon Mechanical Turk (AMT). In the study, we presented users with two sketches of the same object type overlaid and registered to their condition specific model (see supplementary material). The two pairs of object could be from the same or different tutorial types. We asked participants to "Please select the sketch that is more accurate to the underlying model. Do consider the proportions, alignment and perspective. Please ignore style or shading". The studies for the two objects were run independently, so did not have the same responders. Each participant evaluated the 45 unique image pairs for one of the objects. Each study had 220 sets of responses. AMT users were compensated $0.01 per comparison.

The H2S Camera tutorial received 128/220 votes in pairwise comparisons against the basic tutorial. Similarly the H2S Mixer tutorial received 131/220 votes. Using the binomial test to evaluate these results, we can reject the null hypothesis with $p<0.018$ for the Camera and $p<0.006$ for the Mixer.

To evaluate the results further, we performed pairwise comparisons using the Bradley-Terry model. We plot probability histograms for both objects in Figure 4.14. The bottom axis is the probability of being judged more accurate in a pairwise comparison. The left axis is the number of pairwise comparisons that has this probability. Visually we observe H2S tutorials - in both object types - are more likely to be perceived as having higher accuracy.

Scaffold accuracy. To numerically evaluate accuracy we use the corners of the scaffolding primitives. We first rectify the scanned sketches into a normalised coor-

dinate space using the corner registration marks on the paper. For both conditions, we manually marked the 32 corner landmarks and 40 corner landmarks of all scaffolding primitives in the Camera example and the Mixer example, respectively.

For each point, in each condition across users, we computed a mean position, standard deviation, and distance of the mean from ground-truth in 2D (Error) (see Figure 4.15). Standard deviations across conditions were similar, which we take to suggest comparable user drawing skills across conditions. For the Camera example, 2D drawing error was 71% higher in the Basic condition compared with H2S (highly significant with p<2E-10 using two-way ANOVA). For the Mixer example the error was only 3.5% higher in the basic tutorial and was not statistically significant.

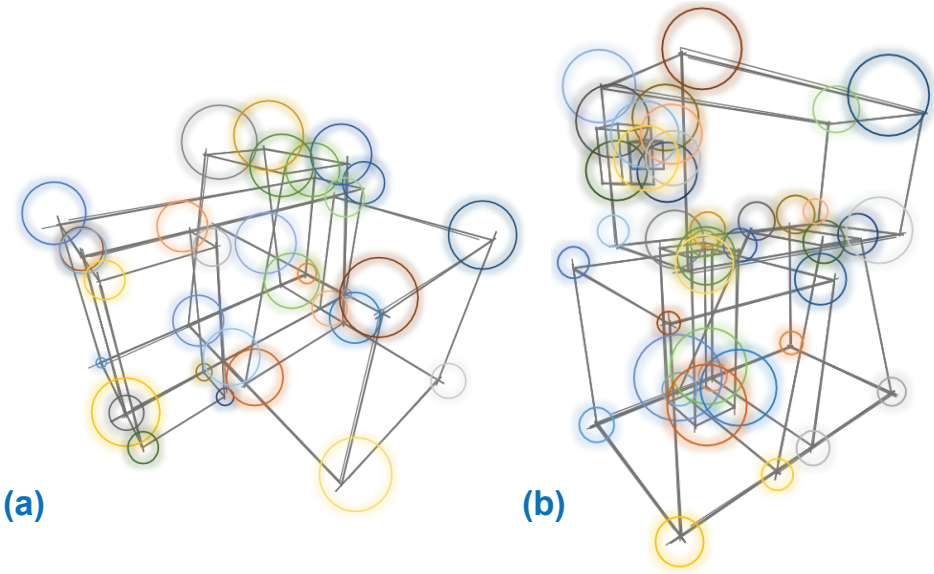


Figure 4.15: How2Sketch Scaffold Accuracy: The mean location of the scaffold primitive corners plotted on the ground truth with the size of each bubble being the mean 2D error.

We explain the variation across objects as follows: The predominant guidance in the Mixer H2S sequence is that many of the primitives share a common bisector plane. There is, however, no guidance for the height of the primitives. Thus it is unsurprising that the scaffold corners are not accurate compared to ground truth, as many primitives accuracy rely solely on getting the height of another primitive correct (see Figure 4.15b). We believe the guidance for the common bisector plane helps with the alignment and perspective of the sketch, hence the perceived accuracy

improvement in the H2S condition from the AMT user study. In the Camera H2S sequence there are no primitives that are solely dependent on a parent primitive with an unguided axis, therefore the users are more accurate at drawing the scaffolding (see Figure 4.15a).

Limitations. Our user study has two potential sources of bias: (i) By having users follow H2S tutorials first we intended to disadvantage H2S against a learning effect, however, this could have potentially introduced a bias in our user feedback regarding enjoyment and satisfaction. (ii) By providing users with a tutorial on how to use construction lines it could imply that construction lines are good practice, therefore the absence of construction lines in the Basic tutorial could have biased the users. In future studies randomizing the ordering of tutorials may help avoid these issues.

4.7 Closing Remarks

We have presented an assistive tool that can aide novice artists to sketch an object of their choice. Continuing with the running theme in this thesis the tool uses multiple representations of a reference scene to power the assistive tool. In this instance the multiple representations are various candidate scaffolding primitives that approximate the object and the inter-primitive relations found in the input reference scene. By analysing all of these modalities we are able to generate an easy-to-follow tutorial for sketching the input model. Through our user study we demonstrate that artists find our tutorials easier to use compared a baseline tutorial. They also had more satisfaction and enjoyment completing H2S tutorials compared to the baseline. We also qualitatively demonstrate that sketches following our tutorials are more accurate through a AMT user study. Through our evaluation we argue that we have met all of the objectives of an assistive tools outlined in Chapter 1.

In Chapter 5 we propose the final tool in this thesis. Similar to Chapter 3 the tool focuses on image editing but this time in the context of transferring image-based edits for multi-channel compositing. To enable this tool the multiple scene representations that are are used to find multimodal correlations are light-path expressions [4] that are output by commercial physically-based renderers.

Chapter 5

Transferring Image-based Edits for Multi-Channel Compositing

A common way to generate high-quality product images is to start with a physically-based render of a 3D scene, apply image-based edits on individual render channels, and then composite the edited channels together (in some cases, on top of a background photograph). This workflow requires users to manually select the right render channels, prescribe channel-specific masks, and set appropriate edit parameters. Unfortunately, such edits cannot be easily reused for global variations of the original scene, such as a rigid-body transformation of the 3D objects or a modified viewpoint, which discourages iterative refinement of both global scene changes and image-based edits. We propose an assistive tool to automatically transfer such user edits across variations of object geometry, illumination, and viewpoint. This transfer problem is challenging since many edits may be visually plausible but non-physical, with a successful transfer dependent on an unknown set of scene attributes that may include both photometric and non-photometric features. To address this challenge, we use the same approach of using multiple representations of a reference scene and multimodal correlation analysis. Specifically, we create an augmented set of photometric and non-photometric guidance channels of the input reference scene. Using the user's original image-edit we then analyse the channels to give an importance weighting to each of them. Finally, we use these weights in an adaptively weighted image analogies formulation to transfer the edit. We demon-

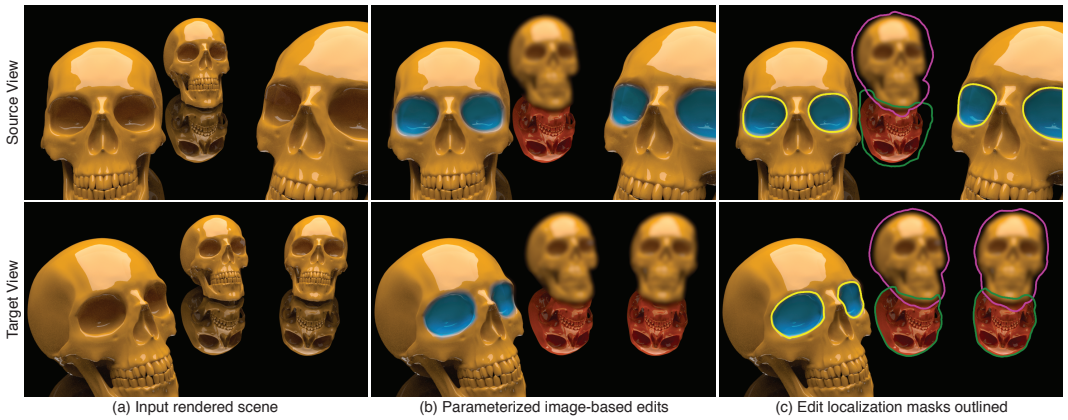


Figure 5.1: (Top) Input source view rendered using a set of photometric render channels. (a) Composite of ALL PHOTOMETRIC channels. (b) The user applies 2D image-based edits to specified channels such as: blurring the background object to create depth of field effect (ALL PHOTOMETRIC channels); adjusting gamma, hue, and saturation to emphasise floor reflections (REFLECTION channel); Making the eye sockets of foreground skulls appear to glow blue by adjusting the hue, saturation, and lightness (DIFFUSE and GLOBAL ILLUM channels). (Bottom) Given a target view (a) with a different scene configuration (skulls are positioned in different 3D locations and orientations) (b) our method transfers the 2D image-based user edits automatically. The right column (c) shows the outlines of the corresponding localization masks for the two views. Multiple instances of the same object make this a challenging scene. For baseline comparisons, please see supplementary material.

strate the usefulness of our tool in a variety of complex edit-transfer scenarios and evaluate them in two separate user studies.

5.1 Introduction

Physically-based rendering algorithms have matured to the point where they are increasingly used to create photorealistic product images. For example, IKEA reports [102] that 75% of their catalogue images are rendered rather than photographed. In addition to being more cost-effective than real photography, one key advantage of rendered content is that it provides artists with greater editing flexibility. While some edits can be easily achieved by changing 3D rendering parameters (e.g., changing the color or intensity of light sources), many other edits are not physically valid and are thus difficult to express in 3D (e.g., removing distracting reflections, emphasizing specific object contours). Artists typically make such non-physical edits in 2D by editing the individual render channels (e.g., DIFFUSE LIGHTING, SECULAR REFLECTIONS, REFRACTIONS, etc.) that together make up the final ren-

dered result. The typical workflow is to mask out a specific element of the image, like a specific reflection or object contour, and then either mute or emphasize it by applying some parameterized adjustment (e.g., brightness, contrast, exposure, levels). Many rendered images are "retouched" in this manner to produce the final composited image. One such example is shown in Figure 5.1. There are several video tutorials demonstrating this workflow [103, 104].

While editing multi-channel renderings is a powerful approach, it also has some challenges. Most high-quality renderings include a large number of render channels (typically 4–15), which requires artists to flip through many channels to determine which one to edit. For many image editing experts who lack 3D rendering expertise, this task is especially difficult since they may have little intuition about which channels contribute to the image element they want to adjust. More importantly, once the artist has made edits on one rendered version of a scene, those edits cannot be re-used to create variations of the scene. For example, if a client or art director requests even small changes to the position or orientation of objects, lights or the camera, all the edits must be redone from scratch for the new scene configuration. Another common scenario is inserting or replacing objects in the scene. This unfortunate limitation adds significant inefficiencies to the authoring process and discourages iterative design space exploration for rendered product images.

In this work, we propose a novel compositing workflow that addresses these challenges. To retouch a rendered image, the user marks a region that requires an edit. Our system then automatically identifies suitable render channels to modify and, based on the selected channels, proposes a candidate mask (which the user can refine if necessary). The user can then make a number of parameterized adjustments - levels, exposure, gamma, blurring, hue, saturation, lightness - to modify the appearance of the masked region, and repeats this process until all the desired edits have been made. Given a modified version of the 3D scene, our system automatically transfers over all of the image-based edits, which allows users to quickly experiment with variations in viewpoint, object positions, object configurations (e.g., replacing an object), and lighting effects while preserving the image-based edits.

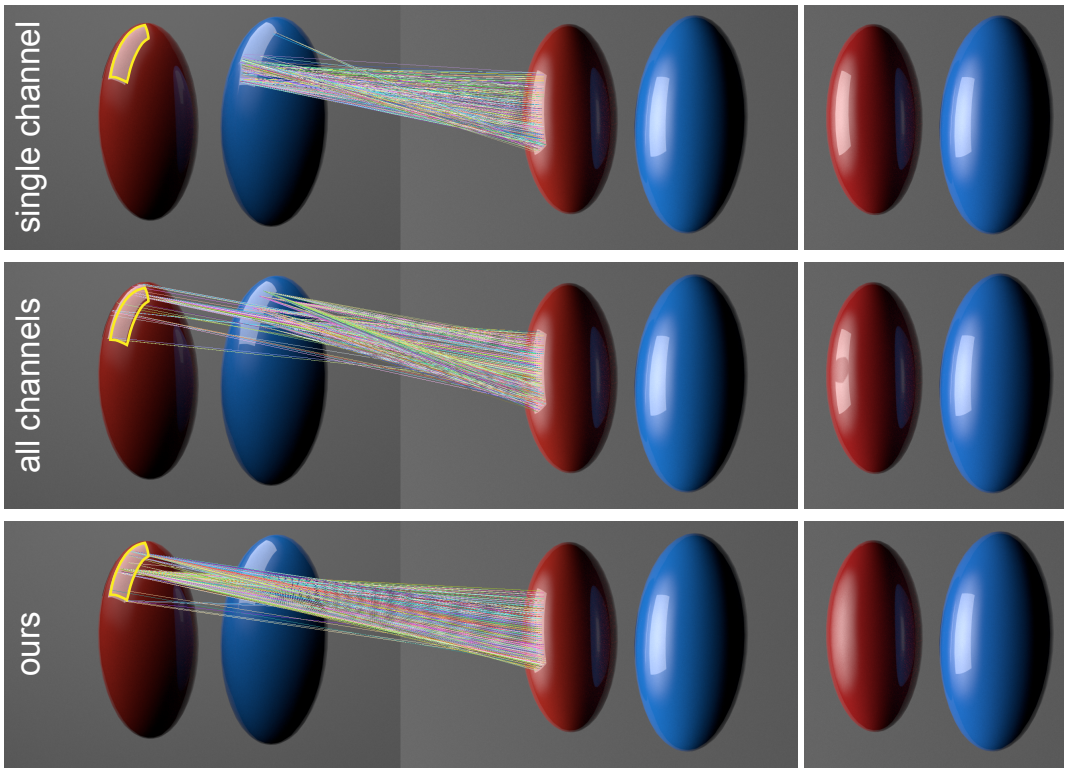


Figure 5.2: Given an example edit for an input view (left) where the user masked out the reflection on the red object (outlined in yellow) to be removed, the challenge is to transfer the edit mask to a novel view (centre). Existing variants of image analogy can easily fail: (top) a single channel (reflection channel) is not sufficient as it wrongly establishes correspondence with the blue object; adding all the photometric channels (middle) with fixed weights is also not sufficient as the channels that are not relevant to the edit corrupt the correspondence, resulting in a bad mask transfer. (bottom) Our method, which adaptively estimates weights for the different channels to best explain the example edit, results in a successful edit transfer. The right column shows the resulting edit using the transferred mask.

For example, Figure 5.1 shows several edits to rendering with multiple instances of a skull being transferred to new scene configuration.

The main technical challenge in supporting this workflow is how to perform the edit transfer. One approach is to formulate the task as an image analogies problem [62, 85], where the input is the original rendered image (A), the edited image (A'), and an unedited rendering of the modified scene (B). The goal is to generate the analogous edited version of the modified scene (B'). Previous work demonstrates that providing the synthesis procedure with additional guidance channels (e.g., PHOTOMETRIC RENDER CHANNELS, otherwise known as *light path expres-*

sions [4]) can be very effective. However, choosing the right guidance channels is not a trivial task. While the edited render channel is an obvious candidate, a single channel is often not sufficient to characterize the edit in a unique way. On the other hand, adding additional channels that are not correlated with the edit is problematic since they add noise and corrupt the signal of the correlated channels, hence can have a negative impact on the synthesized output. Figure 5.2 shows how such problems can arise even in a very simple editing scenario. In short, transferring image-based edits across different 3D scene configurations is a difficult task.

In our approach, we introduce a new image analogies formulation that automatically adapts the weights for a large set of candidate guiding render channels based on the characteristics of each edit. In particular, for each edit, we solve for a sparse set of render channels that best reconstruct the edit via L_1-regularized regression. This technique allows us to transfer edits that depend on a broad spectrum of different scene features (e.g., normals, depth, lighting effects, etc.). Furthermore, rather than synthesizing the appearance of edited image regions, we synthesize the edit masks and then solve for the appropriate adjustment parameters in the modified scene. This approach makes it convenient for users to refine the results by editing the transferred masks and parameters.

We evaluate our method on a range of challenging edit transfer scenarios under different scene variations involving object manipulation, illumination adjustment, and viewpoint changes. In most cases, the automatically transferred edits successfully reproduce the modifications to the original scene configuration and require no additional user refinement. In the few situations where the fully automatic transfers are not completely satisfactory, small tweaks to the synthesized edit masks or adjustment parameters are typically sufficient to achieve the desired result. We conducted a user study demonstrating significant time savings compared to manually transferring edits to different scene variations.

In summary, we present a novel editing workflow for multi-channel compositing; develop a smart selection tool for identifying relevant render passes and automatically creating corresponding local masks; and formulate an optimization for

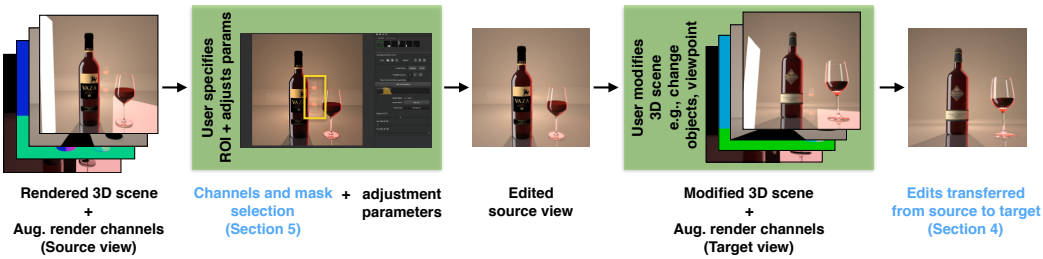


Figure 5.3: System overview. Starting from an input source view of a rendered 3D scene, along with corresponding augmented render channels, the user may make a number of 2D edits. To make an edit, the user first outlines a region of interest (ROI). Our method then automatically determines a region mask and a selection of one or more relevant photometric render channels for the edit. The user then makes a parametric adjustment within the region mask to the selected channels to obtain an edited source view. In this example, the user removes the wine glass reflection and adjusts highlights on the labels. The user may then modify the 3D scene by replacing the 3D objects or changing the viewpoint to yield a target view. Our system automatically transfers the user edits from the source view to the target view. Text on green background denote the user interaction and blue text the computational aspects of our method.

transferring local parametric edits in an adaptive Image Analogies framework.

5.2 System Overview

We illustrate our overall system in Figure 5.3. The input to our system is a 3D scene configuration that includes one or more objects at the desired positions and orientations, materials for those objects, a lighting setup, and a camera viewpoint. Such configurations can be created with most 3D modeling and rendering software (e.g., Maya, VRay). The user may also specify a background photograph in which to composite the rendered scene. Given this input, we provide an interactive editing tool that helps users specify and transfer parameteric image-based edits from the initial configuration of the input scene (*source view*), to a modified configuration (*target view*) that may involve a different viewpoint, lighting, object arrangement, or in some cases, new objects with similar geometry. We represent image-based edits as a 2D region mask that identifies the relevant part of the image to modify and a parametric adjustment that is applied within the mask. As mentioned earlier, each edit is applied to one or more specific render channels.

The main technical contribution is in our synthesis-based approach for transferring image-based edits from the source to the target view. Specifically, we intro-

duce an adaptive version of Image Analogies that automatically determines how to weight various candidate guidance channels in order to transfer each edit. We also present an interface that helps users select and modify the appropriate render channels to specify the image-based edits in the source view. We now provide details for these two aspects.

5.3 Transferring Parameterized Edits

Given a set of image-based edits in the source view, we transfer the edits to the target view in two stages. First, we transfer the 2D region mask to the target view using a new adaptive version of Image Analogies [62]. Next, given the transferred region mask, we update the edit adjustment parameters for the target view. Before describing the details of mask and adjustment parameter transfer, we first introduce our set of augmented render channels that supports both of these steps.

5.3.1 Augmented Render Channels

The Image Analogies method is based on a repeated computation of a dense correspondence field (or nearest neighbor field) using guiding channels from the target to the source view. A critical challenge then is finding the right guiding channels resulting in a correspondence field that would be appropriate for the task of transferring edit masks and adjustment parameters. Note that the desired correspondence field may not simply be the rigid-body transformation of the 3D scene objects or a dense 3D correspondence field between two different 3D shapes. Many common edits, such as adjusting specular highlights or adding a halo around an object, may depend on one or more photometric and non-photometric factors, such as specularity, direction to light source, and the view-dependent silhouette of the object. Thus, our approach leverages a diverse set of rendered guidance channels derived from the 3D scene to help determine the correspondence between views.

Given a 3D asset with positioned lights or environment maps, we can output a full global illumination render of the 3D asset using a renderer such as VRay or Mitsuba [105]. Moreover, we can render a set of photometric channels, also called *light path expressions* [4], that separate the different global illumination effects at

each pixel. The different lighting effects can be diffuse or specular, and together sum to the full global illumination render of the 3D asset.

In addition to the standard set of photometric channels, we also render a set of complementary channels. Such channels are useful for finding edit-dependent dense correspondences. For example, the bottle rim lighting example in Figure 5.7 relies on the distance to the silhouette of the object. We render a number of channels relating to the 2D layout and 3D geometry of the object, such as surface normals and distance transform to the object silhouette. Moreover, we found that including log-channels $\log(A_i + \varepsilon)$, where A_i is a photometric render channel, boosts weak signals and improves transfer results. We used $\varepsilon = 0.001$ for our experiments. Note that for augmented render channels with multiple dimensions at each pixel, we separate each dimension into its own augmented render channel. We normalize the Lab color channels into the range $[0,1]$.

The success of our method does not rely on a specific set of render channels. The technique only requires a diverse superset of channels that are consistent between renderings. We demonstrate results using the VRay and Mitsuba renderers, which generate different sets of augmented render channels. Appendix C lists the specific set of channels for each renderer along with the render times.

5.3.2 Mask Synthesis via Adaptive Image Analogies

Given a set of augmented render channels $A = \{A_i\}$ for the rendered scene in the source view, a user edit e_A in the source view, and augmented render channels $B = \{B_i\}$ for the rendered scene in the target view, our goal is to infer the user edit e_B for the target view. We parameterize a user edit in the source view as $e_A = (X_A, A', \theta_A)$, where X_A are indices into the augmented render channels A indicating which photometric channels were selected by the user for the edit, A' is a real-valued user mask, and θ_A are parameters for the adjustment within the mask A' (see Section 5.4 for how edits e_A are specified using our interface). Similarly, we have $e_B = (X_B, B', \theta_B)$ for the target view. We assume that the selected photometric channels for the target view are the same as the source view, so we set $X_B \leftarrow X_A$. In this section we describe how to synthesize the user mask B' in the target view.

We formulate the mask synthesis task as one of finding an image analogy where $A : A' :: B : B'$ [62]. While one could explicitly reason about the 3D scene via techniques for inverse rendering [67, 63, 106, 107, 64] to recover the unknown mask, we argue that formulating the mask transfer task via image analogies is more flexible as it allows transfer of visually plausible but non-physically valid user edits.

The Image Analogies formulation proposed by Hertzmann et al. [62] is a multi-scale iterative optimization algorithm. At each scale every iteration starts by computing a dense correspondence field given a previously computed B'. For every target patch around pixel q a best-matching source patch p is found that minimizes the following energy:

$$E_q(p) = ||A'(p) - B'(q)||^2 + \mu||A(p) - B(q)||^2, \tag{5.1}$$

where μ is a tunable scalar hyperparameter. Note that for the first iteration only the second term is used so that an initial B' mask can be synthesized. Given the dense correspondences, B' is updated by averaging the mask values for all overlapping best-matched patches for every pixel q. The overall energy is decreased after a few iterations and the result is upsampled to a finer scale until a solution (transferred mask and final correspondence field) at the finest scale is achieved. In [62] the inputs are RGB images or steerable filter responses. More recently, Fišer et al. [85] introduced StyLit, which uses photometric render channels as inputs to Image Analogies for illumination-guided stylization of 3D renderings. We build on the StyLit formulation for our task.

Transferring user-edit masks presents different challenges than the 3D rendering stylization transfer demonstrated in StyLit. As we will demonstrate in Section 5.5, simply applying the StyLit Image Analogies formulation produces a transferred edit mask with significant artifacts. We identify two reasons for this failure: (i) the information required for a particular edit transfer might not be present in the standard photometric render channels; and (ii) StyLit treats each photometric render channel equally in the image analogies formulation. For example, to adjust a specular highlight, the system needs knowledge of not only the specular component, but

also the direction to the light source. Moreover, not all photometric render channels are relevant to transfer the edit.

To address these issues, we leverage our augmented render channels to add non-photometric information that can aid in the transfer. To make use of the additional channels, we extend the standard image analogies formulation to one that adapts the weights of the different augmented render channels to a given user edit e_A:

$$E_q^{(e_A)}(p) = ||A'(p) - B'(q)||^2 + \mu \sum_i w_i^{(e_A)} ||A_i(p) - B_i(q)||^2, \tag{5.2}$$

where $\{w_i^{(e_A)}\}$ are given scalar weights for the augmented render channels dependent on user edit e_A.

5.3.3 Finding Edit-Dependent Weights

The adaptive edit-dependent image analogies energy in Equation (5.2) requires knowledge of a set of edit-dependent weights $\{w_i^{(e_A)}\}$, which guides the synthesis algorithm to know which augmented render channels are important for synthesis. We seek to automatically infer the edit-dependent weights given the user edit. This is challenging as we do not know a priori what type of edit the user is making, e.g., adjusting specular highlight or adding silhouette halo, or which channels are important for the edit.

Since the desired weights are dependent on the user edit, and we do not have training examples with synthesized masks B' in the target view, we make the assumption that rendered channels important to synthesize B' in the target view are the same as the ones important to synthesize images of the user edit in the source view. As B' is related to the user edit, we find that this is a reasonable assumption that holds in practice and demonstrated in our final results. Moreover, we assume that not all channels are important and there can be some redundancy due to having an overcomplete superset of channels, meaning a sparse subset of all the channels will be sufficient to successfully transfer edits.

We formulate our edit-dependent weight recovery problem as an L_1-regularized regression to synthesize the user-edited source view. Let I_A be the

image of the source rendered scene and $I_{A'}$ the image of the edited source rendered scene. We define the source edit-difference image at pixel location p as $\Delta_A(p) = I_{A'}(p) - I_A(p)$. We seek to find the weights $w^{(e_A)} = [w_1^{(e_A)}, \ldots, w_N^{(e_A)}]^T$ that reconstructs the source edit-difference image from the augmented render channels $\{A_i\}$ for a set of sampled pixel location S,

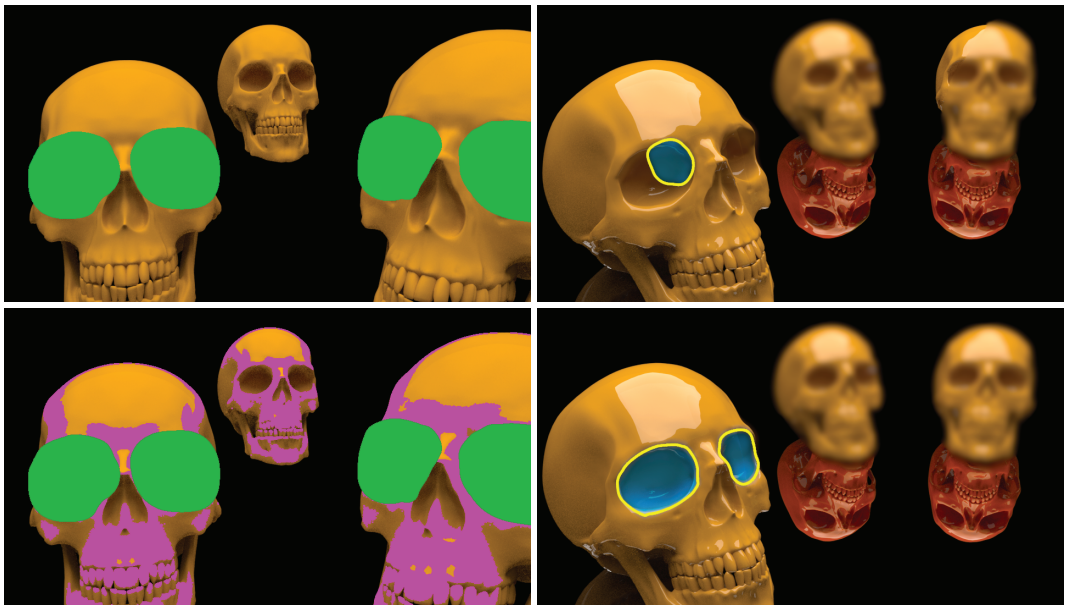
$$w^{(e_A)} \leftarrow \underset{w}{\operatorname{argmin}} \sum_{p \in S} \left(\sum_i A_i(p) w_i - \Delta_A(p) \right)^2 + \lambda ||w||_1. \tag{5.3}$$

We use an L_1 sparsity prior over the edit-dependent weights, weighted by hyper-parameter λ to select augmented render channels important for reconstructing the edit-difference image. Note that we use the Lab lightness channel for Δ_A instead of all color channels as we found that reconstructing the lightness channel provides better correspondences when there is a significant change in color in the target view.

Sampling. Since there are often many fewer non-zero pixels in the edit mask A' (dubbed *inside mask pixels*; we dub the complement set as *outside mask pixels*), we do not regress over the entire edit-difference image Δ_A. Instead, we balance the number of inside and outside mask pixels by including only the hardest outside mask pixels. The union of the inside and hardest outside mask pixels form the set of pixels S. Formally, let k be the number of inside mask pixels. We find the k outside mask pixels that are closest to the mean vector of the user-selected photometric channels $\{A_i\}_{i \in X_A}$ for the inside mask pixels. We also found that including pixels around the edge of the inside mask pixels found via dilation improves results. We used a unit dilation kernel of 3×3 pixels. This sampling scheme helps find weights $w^{(e_A)}$ that differentiate between regions with similar features for inside and outside mask pixels. Figure 5.4 demonstrates the ability of this sampling strategy to find features unique to the given edit.

5.3.4 Adjustment Parameter Transfer

Our goal is, given the synthesized user mask B', to recover the user adjustment parameters θ_B for the user mask (see Figure 5.5). To aid in the recovery, we first seek to synthesize the image $\tilde{I}_{B'}$, which is an estimate of the edited target rendered



(a) Sampling Strategies (b) Transferred edits w/ localization mask outlined

Figure 5.4: Using the source and target views from Figure 4.1 we show: (a) sampling strategies overlaid on the source view (showing user-selected photometric channels), and (b), the target view with resulting transferred edits. We only show the sampling and outline of the transferred mask for a single edit (eye sockets). (Top) Naively sampling only the masked region and immediate surrounding pixels (green) results in channels with non-zero weights (LOG SPECULAR, LOG REFLECTION, LIGHT DIRECTION and HALF ANGLE) that fail to uniquely describe the users edit and to transfer the masks. (Bottom) Our sampling additionally samples hard negatives (magenta), resulting in selection of render channels that characterizes the edit (LOG LIGHTING, SHADOW, LOG SHADOW and Z-DEPTH), and successfully transferring the edits.

scene $I_{B'}$. We can then estimate the adjustment parameters θ_B by optimizing over the adjustment that best matches the estimate image $\tilde{I}_{B'}$.

To synthesize the estimate image $\tilde{I}_{B'}$, we leverage the learned edit-dependent weights $w^{(e_A)}$ to reconstruct the edit-difference image Δ_B from the augmented render channels $\{B_i\}$,

$$\Delta_B(p) = \begin{cases} \sum_i w_i^{(e_A)} B_i(p). & \text{if } p \in \{p | B'(p) \neq 0\} \\ 0. & \text{otherwise} \end{cases} \tag{5.4}$$

Here, we only synthesize within non-zero pixels in the synthesized mask B'. Given the image of the target rendered scene I_B, we obtain the estimate image $\tilde{I}_{B'}(p) = I_B(p) + \Delta_B(p)$ at pixel location p.

Given the estimated image of the edited target rendered scene $\tilde{I}_{B'}$, we can recover the adjustment parameters θ_B by minimizing the following energy,

$$\theta_B \leftarrow \underset{\theta}{\text{argmin}} \sum_{p \in \{p | B'(p) \neq 0\}} ||\tilde{I}_{B'}(p) - I_{B'}(p;\theta)||^2, \tag{5.5}$$

where the image of the edited target rendered scene is given by

$$I_{B'}(p;\theta) = I_B(p) + \sum_{i \in X_B} B'(p) \left(f_\theta \left(B_i, p \right) - B_i(p) \right), \tag{5.6}$$

where f_θ is a parameterized image adjustment function. We optimize the above objective via grid search over the parameter space θ_B. We provide details of these f_θ functions and for the grid search in Section 5.4.2 and Appendix D when we introduce our editing tool.

Figure 5.5: Adjustment parameter transfer. (Left) Baseline where we simply copy the user-provided adjustment parameter from the source view. (Right) Our approach for adjustment parameter transfer. Notice that simply copying the parameter results in a brighter reflection, whereas our approach more closely matches the edited source view (Figure 5.7).

5.3.5 Implementation Details

User-specified masks can often be coarsely specified if a masked region in the user-selected photometric channels $\{A_i\}_{i \in X_A}$ is surrounded by black pixels. This is due to many parameter adjustments having no effect in these black regions. Including all of these masked pixels can lead to over-sampling pixels p where the edit-difference $\Delta_A(p)$ is zero. This can make Equation 5.3 ineffective at choosing relevant features. So as a pre-processing step we removed pixel locations p from the mask $A'(p)$ where $\sum_{i \in X_A} f_{\theta_A}(A_i, p)$ is less than 10^{-3}. In an additional pre-processing step for

gaussian-blur edits we set $A_i(p) = f_{\theta_A}(A_i, p)$ as the blur operation has a spatial extent not captured in Equation 5.3 as it does not take into account neighbouring pixels. This pre-processing allows for edits with a spatial extent to work in our formulation.

We used a CPU C++ implementation multi-scale guided synthesis algorithm [85]. Similar to their method, we used a fixed patch size of 5×5 pixels and pyramid down-sampling ratio of 2. We ran synthesis up to 6 levels in the pyramid and used fewer levels if the down-sampled user mask comprised less than 30 pixels in a given level. We set the hyperparameter μ for the adaptive image analogies Energy (5.2) to $\mu = 3$ for the first level and $\mu = \frac{1}{3}$ for the last level, and linearly interpolated the intermediary levels. Intuitively, the hyperparameter setting μ at the different levels allow for more guidance over the features at the beginning, and later to previous level's mask B'.

Additionally, after each level in the adaptive image analogies pyramid, we discarded correspondences that went to pixel locations in the target view where all of the selected photometric channels were less than 10^{-3}. At a given level of the pyramid it may not matter masking a region that is nearly black. However, a problem arises when the mask propagates to later levels of the pyramid where it should not be masked but due to the decreasing μ parameter the correspondence does not update, leading to spurious artifacts. In the special case of an edit e_A with a spatial extent (e.g. blur), we apply the edit at the pixel location using the θ_A parameters before testing for small values.

Similar to StyLit, we initialized B' by randomly assigning from A'. Additionally, on the first iteration we applied no weighting to the $||A'(p) - B'(q)||^2$ term. Fišer et al. [85] introduce a new way to compute a correspondence field from the target to the source view, that avoids "washout" and obvious repetition artifacts. Their solution involved multiple source-to-target search iterations that significantly slowed down the computation. Since these artifacts are less relevant for textureless masks as they are for RGB images, we use the regular target-to-source search [62, 108] using PatchMatch [9].

The hardest regions to find correct correspondences are mask boundaries due to the averaging of conflicting features in the image pyramid. In cases of underestimating the boundary location, as a post-processing step, we compute the mean of the selected render channels in the output mask. For for all mask boundary pixels, we allow the mask to grow if the neighbouring pixel in the selected render channels was within 0.1 distance to the mean, up to a maximum of 5 pixels. To optimize the L1-regularized regression Energy (5.3), we used the publicly available POGS solver[1]. For scenes rendered using VRay we set hyperparameter $\lambda = 10000 + 300 \cdot N_L$, where N_L is the number of lights in the 3D scene as additional light sources introduce additional channels requiring more regularization. For scene rendered with Mitsuba we set $\lambda = 20000$ as the number of channels is fixed.

5.4 Interface

Our interface allows users to quickly select render channels to edit, generate masks, and set adjustment parameters. The user starts by loading a stack of photometric render channels into our interface (see Figure 5.3 and supplementary video). By default, the users are only shown the composited image, but can switch to other channels as desired. For each edit, they specify a rough region of interest on the composited image, and our method automatically selects a subset of channels (named X_A in Section 5.3.2). The user can verify the correct channels were chosen via hot keys and use the auto-mask feature to create an edit mask (named e_A Section 5.3.2). The user then edits the selected channels inside the masked region by adjusting some of the supported adjustment parameters (named θ_A Section 5.3.2). The user can perform multiple edits on the same example scene, and transfer them to other comparable scene variations.

5.4.1 Render Channel Selection For Editing

In order to select a subset of render channels, the user simply specifies a coarse region of interest (ROI) using either a rectangular marquee or polygon selection tool directly on the the final composited image. Our selection method then identifies the

[1] http://foges.github.io/pogs

relevant channels based on the assumption that the user is interested in only those layers that make the selected region *unique* with respect to the neighboring regions. In the following, we first describe how to sample neighboring regions, formulate the selection problem given a choice of such neighboring regions, and finally create the edit mask (see Figure 5.6).

Sampling neighboring regions. Let $P_\star$ denote the set of all pixel locations in the user-selected ROI. Given $P_\star$, we first sample other regions at random by displacing the ROI by random translations with magnitude in the range $[\delta, 2\delta]$ with δ denoting the diameter of ROI $P_\star$ bounding circle. From the random samples we remove overlapping selections and those intersecting the ROI to generate m candidate (neighboring) patches $\{P_1, \ldots, P_m\}$, where P_j denotes a set of pixel locations in the jth neighboring region.

Selecting among the render channels. Among the channels $\{A_i\}_{i=1}^N$, we seek to identify the ones that are distinct within the ROI with respect to the spatially neighboring regions. Similar to approaches for bottom-up saliency [109], we measure distinctness for a channel A_i by computing a difference between the statistics within ROI $P_\star$ and all neighboring regions P_j.

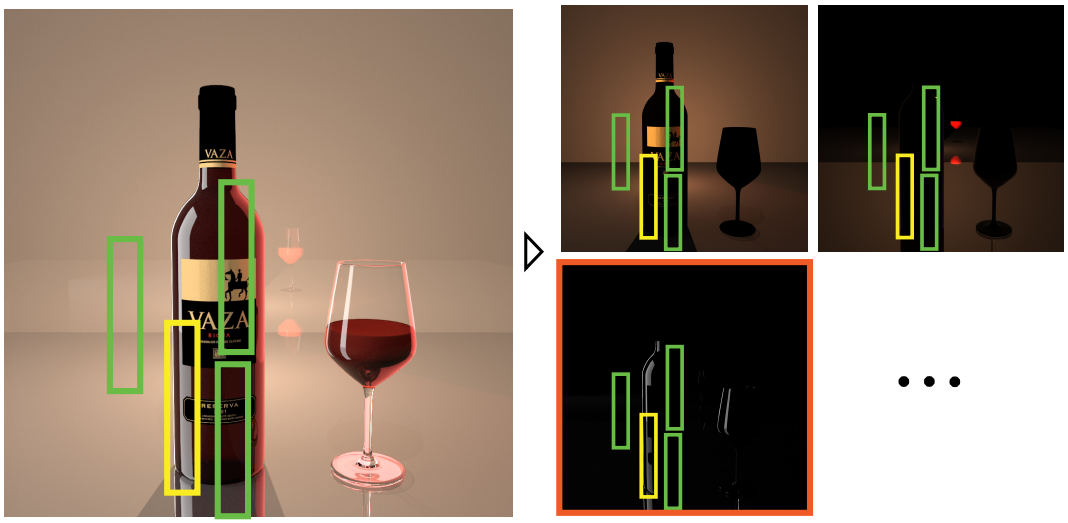


Figure 5.6: (Left) The user marks a region of interest (ROI) $P_\star$ (shown as the yellow polygon) on the input composited image. (Right) By comparing patch statistics against neighboring regions P_j (shown as green polygons), our method automatically chooses which render channel(s) maximizes the uniqueness of ROI $P_\star$. In this example the reflection channel (highlighted in orange) was chosen.

Let $\mu_i(P) = \mathbf{E}_{p\sim P}[A_i(p)]$ be the mean value within the render channel A_i for pixel locations P, and $\sigma_i^2(P) = \mathbf{E}_{p\sim P}[(A_i(p) - \mu_i(P))^2]$ the variance, stored as a vector of statistics $(\mu_i(P), \sigma_i(P))$. We define the difference between the statistics within the ROI $P_\star$ and neighboring region P_j for channel A_i using $d_{j,i} = \|(\mu_i(P_j), \sigma_i(P_j)) - (\mu_i(P_\star), \sigma_i(P_\star))\|$ as the L_2 distance between their respective statistics vectors. We define the vector of differences between ROI $P_\star$ and neighbor region P_j across all render channels as $\mathbf{d}_j = \left(d_{j,1}, \ldots, d_{j,N}\right)^T$.

Our goal is to find a selection vector $\mathbf{x} = (x_1, \ldots, x_N)^T$ that makes the user-selected ROI unique with respect to its neighbor regions, i.e., maximizes the accumulated differences across all neighbor regions P_j:

$$\max_{\mathbf{x}} \sum_j \left(\mathbf{x}^T \mathbf{d}_j\right)^2 \qquad \text{s.t.} \quad \mathbf{x}^T\mathbf{x} = 1, \tag{5.7}$$

where $\mathbf{x}^T\mathbf{x} = 1$ is used to regularize the problem.

Such an optimal $\mathbf{x}$ can be directly computed as the eigenvector corresponding to the highest eigenvalue of the matrix $C = \sum_j \mathbf{d}_j \mathbf{d}_j^T$. Please refer to Appendix B for details. In order to convert the vector $\mathbf{x}$ to the final selection of channels X_A, we sort the channels based on x_i^2 and pick the top ones that accounts for $0.9\sum_i x_i^2$. In our experiments, this resulted in typically $1-3$ selected channels.

Creating the edit mask. Using our selected channels X_A we use GrabCut [10] to create the final mask A'. Specifically, we composite the selected channels $\sum_{i\in X_A} A_i$ and sample pixels inside and outside the ROI $P_\star$ to form the mixture model for the foreground and background, respectively. We erode pixels from the ROI $P_\star$ boundary using a 3x3 kernel to avoid boundary artifacts during GrabCut, and discard pixels within 20% of the ROI $P_\star$ diameter. We then run GrabCut to get an edit mask (c.f., Figure 5.6 top-right). If desired, the user can adjust the edit mask using a brush tool. As a final step we set a mask pixel to zero if all of the selected channels are zero at that pixel. This prevents the adapted image analogies returning spurious correspondences when transferring the mask to the target view.

5.4.2 Parameterized Adjustments

To complete the edit e_A, our interface allows the user to adjust several parameters affecting the selected channels X_A in the region masked by A'. Additionally, after an edit has been transferred to the target scene (c.f., Sections 5.3.2 and 5.3.4), the user can similarly continue editing the transferred edit e_B. In Equation (5.6) we outline how the adjustment of a parameter affects the final composite using the parameters θ. We currently support the following adjustments: exposure, levels, gamma, hue, saturation, lightness and Gaussian blur. These cover a wide range of edits as demonstrated by the variety of examples in our paper. Furthermore, comparing with the editing operations used in online tutorials [103, 104], the only editing operation we do not support is painting colors directly. The details of the specific parameters θ and how they are applied to a render channel can be found in Appendix D.

Grid Search. To find the optimal parameters θ in Equation (5.5), we first normalize the values of the individual parameters into the range $[0, 1]$ and perform a grid search sampling every 0.05, before denormalising into the original domain. In addition to sampling at every 0.05, we sample the exact parameter setting for the source view, as this may be the most appropriate parameter value for the transfer. We only perform grid search for parameters that are not at their default settings in the source view. We do not attempt to search for the Gaussian blur parameters and simply use the θ_A provided by the user in the source view edit.

5.5 Results

In this section we show results of our automatic system for transferring parameterized edits. In our experiments, we used thirteen different 3D scenes, three of which were composited onto background photographs. Motivated by our target application, we selected 3D scenes that may appear in product images, such as a car model, a bottle, and a wristwatch. We created the majority of the scenes ourselves, using 3D assets we collected exclusively from Turbosquid[2] and Adobe Stock[3], with

[2]http://www.turbosquid.com
[3]https://stock.adobe.com

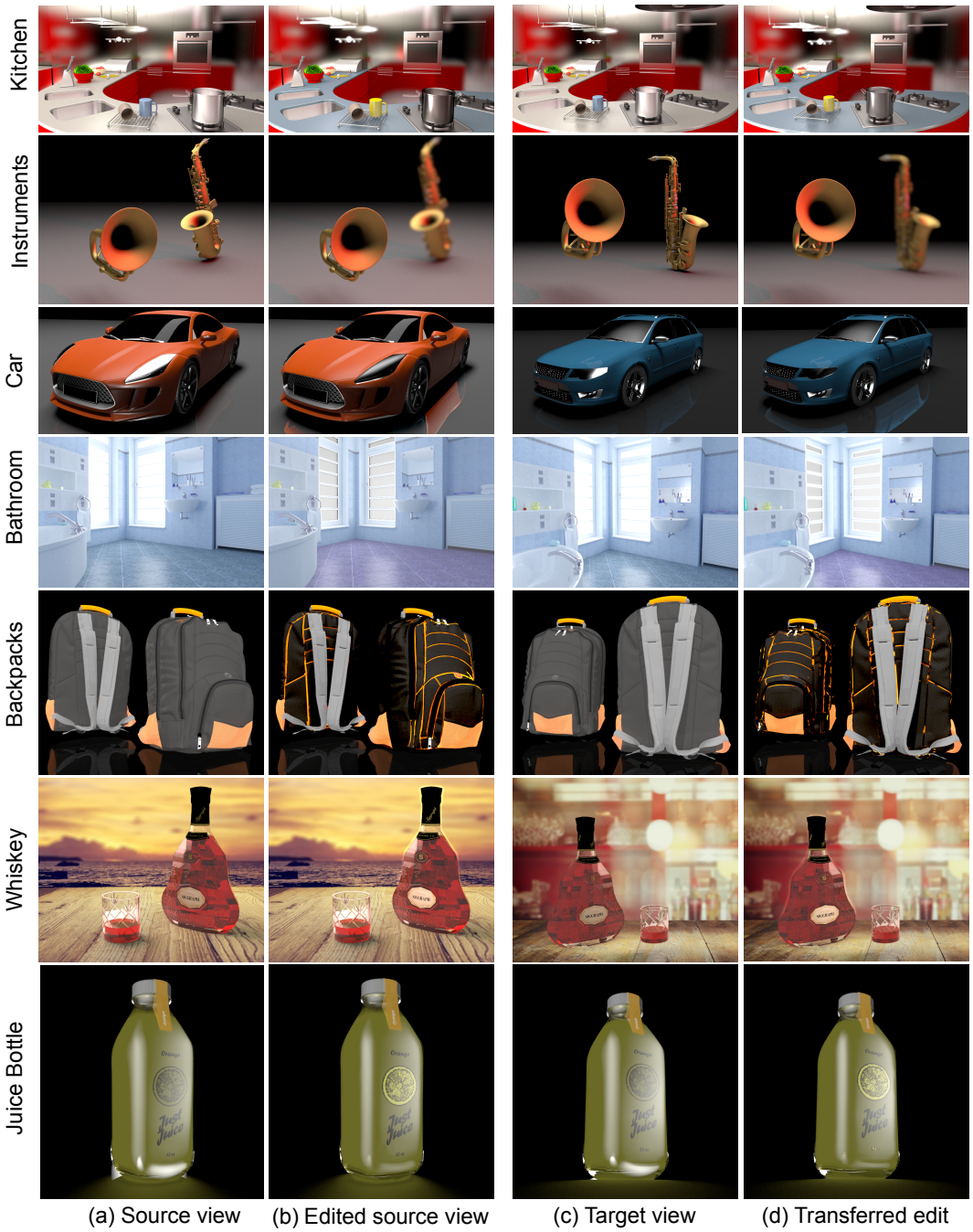


Figure 5.7: Our results. To best view the transferred edits, please see the electronic paper version, the supplemental video, and the suppl. PDF found on the project webpage[5].

the exception of the *dragon*, made available by Stanford University[4]. Additionally, we used the San Miguel, Kitchen and Bathroom [110] scenes. When compositing a rendered view into a photograph, we used stock photographs as background images. The input renderings, user image-edits and source code can be found on the project webpage[5].

As there are no publicly-available datasets of 3D rendered scenes with 2D touchups, we manually set up and edited different 3D scenes to highlight a variety of touchups and effects that our automatic edit transfer approach can handle. Setting up the initial scene took between 30 and 120 minutes, with most of the time spent on adjusting lights and material properties.

We then applied common image-based edits to refine lighting effects, emphasize shape or material properties, and highlight important details and objects. Finally, to create the target views, we modified the 3D scenes in various ways, such as changing the camera viewpoint, re-arranging objects, and in some cases, replacing or adding object geometry. Please see Appendix E for a complete description of the edits to our 3D scenes.

Figure 5.7 shows the image-based edits and automatically transferred results for some of our example scenes. We show additional results in our supplemental PDF, which also includes the masks for all edits. These examples demonstrate the variety of different 2D touchups and scene modifications that our method is able to support. Since some of the edits are (by design) subtle, you may want to zoom into the electronic version of the paper and supplemental materials. Typically, finding edit-dependent weights (Section 5.3.3) takes 10 - 40 seconds to compute and the parameter grid search (Section 5.3.4) 20 - 60 seconds, the timings vary depending on the size of the mask.

5.5.1 Limitations

To test the limits of our method we transfer the edits from a single exemplar to frames from an animation sequence. In cases where features remain consistent

[4]https://graphics.stanford.edu/data/3Dscanrep/
[5]http://geometry.cs.ucl.ac.uk/projects/2017/edit-transfer/

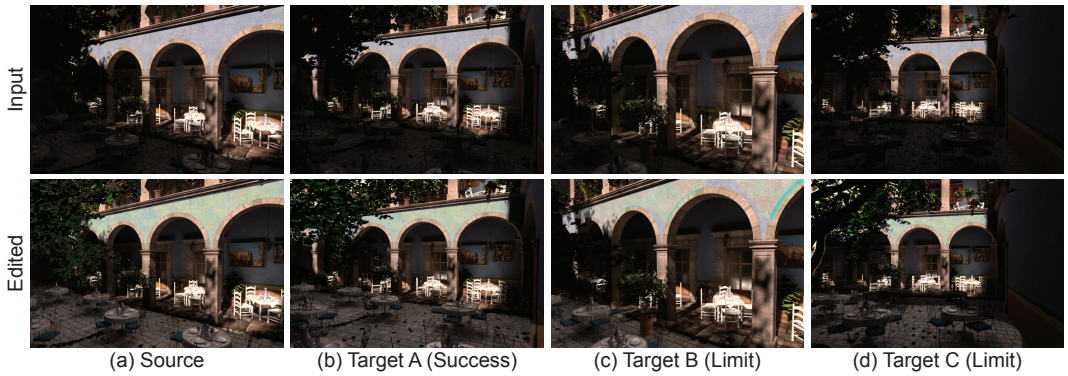


Figure 5.8: Testing Limits: (a) Given a single source input frame (top) and user edit (bottom) we transfer the edit to frames of an animation where the camera moves along a path. (b) For moderate camera moves the edits transfer successfully. For severe camera movements (c) forward or (d) backwards the method reaches its limits as the scene geometry scales and/or new geometry comes into view. Note the artefacts in (c) where bricks in the archway are incorrectly turned turquoise or (d) only part of the table is brightened. For additional frames in the camera path please see supplementary material.

throughout the animation, such as the rotating dragon in the supplemental video, the edits transfer successfully. However, if the content in the source and target views changes significantly throughout the sequence, the transfer begins to fail as the features in the source view are not present in the target view. We demonstrate this by zooming the camera in/out and revealing new geometry and lighting effects in the San Miguel scene (Figure 5.8).

In addition to the aforementioned limitation, we have identified four other potential limitations of our approach. First, the edit-dependent adaptive image analogies approach performs the synthesis in a coarse-to-fine fashion. As a result, features over small spatial extent may be missed by the coarse scales, resulting in mask synthesis artifacts, e.g., along an object boundary. Second, our approach may have difficulty in pixel regions when a second light source interferes with the target view. Thirdly, not all edit operations can be easily described using a mask and adjustment parameter (e.g. clone brush tool) and therefore cannot be transferred using our method. Finally, our formulation assumes the photometric render channels have a linear blending relationship, which may not be true for certain advanced edit operations.

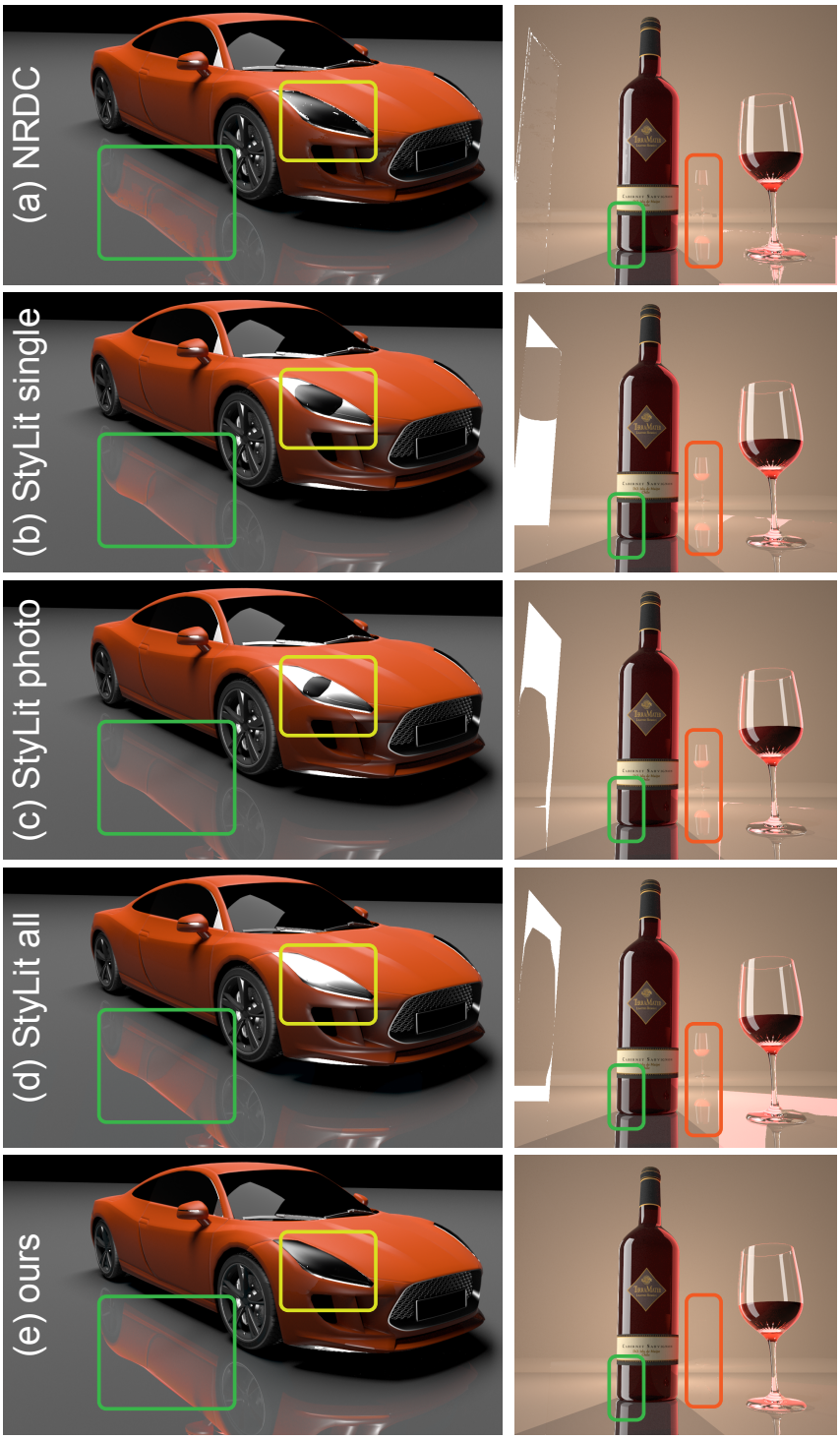


Figure 5.9: Baseline comparison. (a) Non-rigid dense correspondences [75], (b) StyLit with only the single user-edited render channel, (c) StyLit that uses all of the photometric render channels, (d) StyLit that uses all of the augmented render channels, (e) Ours. Notice that all baselines are unable to transfer the full edit from the source view to target view in all cases, whereas our approach successfully handles the edits. Note only more subtle edits highlighted. Please refer to Figure 5.7 for the Car source images and Figure 5.3 for the Wine Bottle source and target images. The Car target image and additional comparisons can be found in the supplemental material on the project page[5].

5.5.2 Baseline Comparisons

For the edit transfer task, we compare our approach against a number of baselines and existing approaches for finding dense correspondences. Our first baseline is to simply use the known 3D shape correspondences between the two views (*correspondences*). The second baseline is StyLit [85]. For StyLit, we compare against three variants: (i) "out of the box" StyLit that uses ALL PHOTOMETRIC render channels (*StyLit photo channels*), (ii) StyLit that uses only the edited render channel (*StyLit single*), and (iii) StyLit that uses all of the augmented render channels (*StyLit all channels*). We also compare against two algorithms for finding dense correspondences between two images using their source code: non-rigid dense correspondences (*NRDC*) [75] and Transfusive Image Manipulation (*Transfusive*) [77].

We show output comparisons for NRDC, StyLit single, StyLit photo channels, and StyLit all channels in Figure 5.9. Notice how all baselines are unable to transfer the full edit from the source view to the target view for all cases. For example, all methods fail to remove the wine glass reflection in the background. NRDC and StyLit all channels introduce artifacts within the watch face. While all the baselines can transfer the car reflection, there are either artifacts in the transfer for the front light, or in the case of StyLit all channels the edit for the light fails to transfer at all.

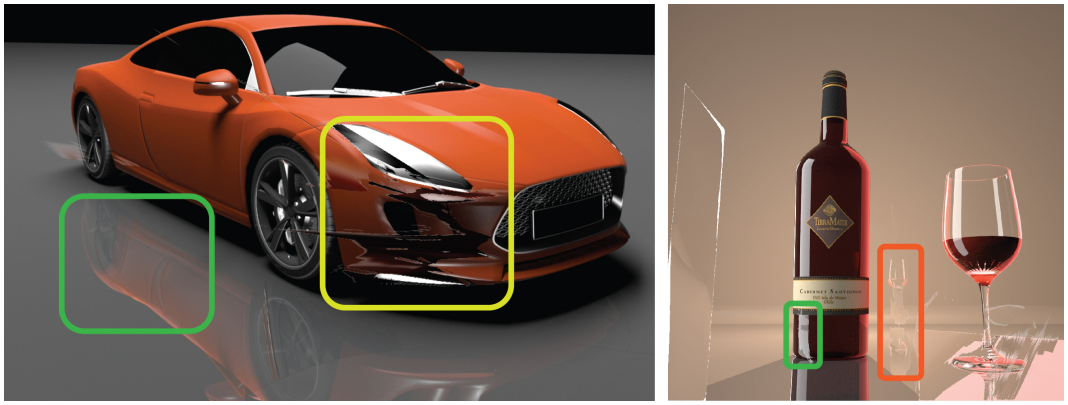


Figure 5.10: Comparison with Transfusive Image Manipulation [77]. In these examples, SURF matches (as used in their paper) failed to find reasonable correspondences and the method was initialized by manually selecting pairs of corresponding points between the source and target views. In the two examples the method fails to transfer all edits successfully and the edits which are transferred have ghosting artefacts . Please note that the tranfusive image manipulation work was designed for an entirely different application and it works directly on the composited image without access to the render channels.

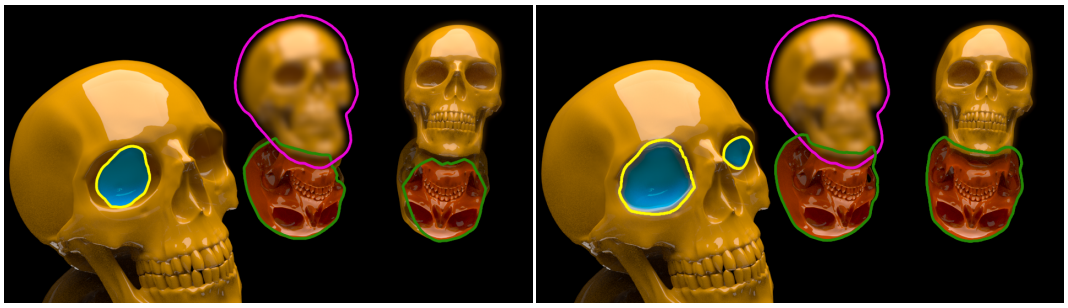


Figure 5.11: OBJECT MASK comparison. Using the source and target views from Figure 4.1 we show that (left) using StyLit with ALL PHOTOMETRIC render channels, additionally augmented with OBJECT MASK fails to transfer the masks correctly. (right) Additionally adding the log of ALL PHOTOMETRIC render channels to the set to available channels improves results but still fails to transfer all edits correctly. Having OBJECT MASK for guidance means edits can only be transferred to the same object they were applied to, in this example the desired outcome is to have both background skulls blurred as shown in our result in Figure 4.1.

Comparisons with Transfusive Image Manipulation are shown in Figure 5.10. The method was initialised with manually annotated pairs of points in the two views due to poor feature matching. Despite this additional interaction, the results suffer from inaccuracies in the correspondences and erroneously transfer edits.

We show two qualitative comparisons with additional baselines. In Figure 5.11 we show how using StyLit with ALL PHOTOMETRIC render channels, additionally augmented with the log of each channel and the OBJECT MASK also fails to transfer edits correctly. Secondly, we show the effectiveness of our approach for adjustment parameter transfer in Figure 5.5. We compare against a baseline where we simply copy over the adjustment parameter the user selected in the source view to the target view. Notice that simply copying the parameter to the target view results in a bright reflection of the car. Our inferred adjustment parameter for the target view allows the reflection to more closely match the edited source view.

Perceptual study. To quantitatively evaluate our approach, we performed a perceptual study comparing our results against several baseline edit-transfer techniques: NRDC, StyLit single, StyLit photo channels, and StyLit all channels. We used a two-alternative forced choice (2AFC) design that shows a raw (A) and edited (A') source view, a raw target view (B), and two candidate edits for the target view generated by two of the methods under evaluation. The judge is asked to select the

candidate edit that is more similar to A'. We generated all pairs of comparisons for three different scenes and ran the experiment on Amazon Mechanical Turk (AMT). In total, we had 147 distinct AMT workers and obtained 50 judgements for each pair of candidate edits. To analyze the data, we used the Bradley-Terry model [111] to compute the likelihood of an edit transfer technique being selected by an AMT worker in a comparison. Our results are shown in Figure 5.12. Please refer to the supplemental for the interface shown to the AMT workers.

5.5.3 User Study

While the comparisons above demonstrate the effectiveness of our automatic edit-transfer technique, we also wanted to investigate the utility of our method within an interactive editing workflow where users may want to refine the automatic-transfer results. To this end, we conducted a comparative user study where participants used Adobe Photoshop to transfer edits to target scenes in two different ways: manually (i.e., specifying all the masks and image adjustment parameters from scratch) and using our automatic transfer results as a starting point. We use Photoshop in both conditions to achieve a more controlled comparison and provide an ecologically valid setting where users have access to an industry-standard set of editing features to refine auto-transferred edits. We recruited 16 participants from a university and a large software company for the study. Since our approach is designed primarily for artists with some image editing expertise, we focused on candidates who are reasonably familiar with Adobe Photoshop; ten of the participants had at least five years of Photoshop experience, and only two had used the software for less than a year. We report qualitative feedback from the editing sessions and quantitative data on the quality and completion time of the edits.

5.5.3.1 Methodology

We asked each participant to perform a total of four edit-transfer tasks on two different scenes, *Juice Bottle* (Figure 5.7) and *Car* (source view in Figure 5.7 and target view in Figure 5.9) . For each scene, we first presented a *source* Photoshop document containing both a raw (A) and edited (A') version of the scene, along with a

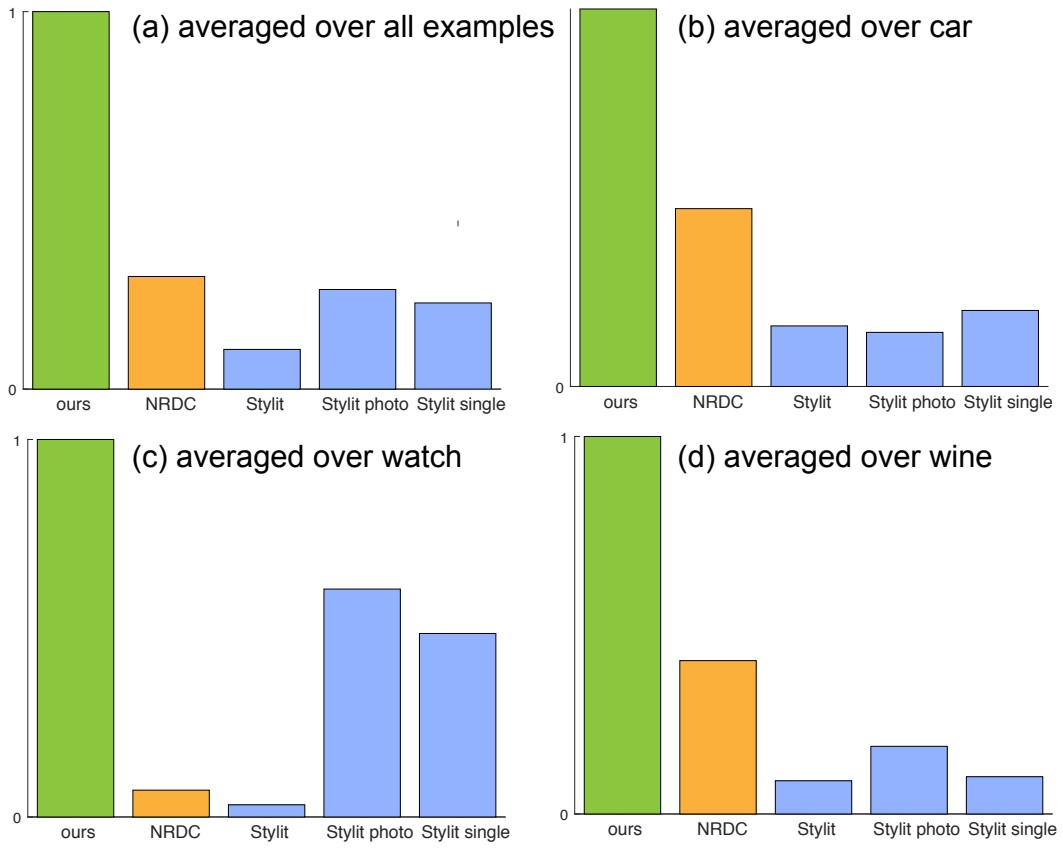


Figure 5.12: Quantitative evaluation. We performed a pairwise-comparison user study on Amazon Mechanical Turk. Shown are likelihoods from the Bradley-Terry model [111] (normalized to 1) for the different approaches over (a) all scenes, (b) car scene, (c) watch scene, (d) wine scene. Please see the text for more details.

text description of the edits with annotated figures highlighting the changes. In the Photoshop document, edits were represented as *adjustment layers* that encode a parameterized image adjustment and mask applied to a specified render channel. By toggling the visibility of these layers and the associated render channels, users were able to see the effect of each edit. They could also inspect the image adjustment parameters and masks.

After users familiarized themselves with the edits, we gave them a *target* Photoshop document with a modified configuration of the scene (B) and asked them to produce an edited version (B') that is analogous to the differences between A and A'. We created two types of target documents. The *manual* version provides the same set of adjustment layers (applied to the same set of render channels) as the source document, but each adjustment is set to its default parameters (which have no effect)

and the mask is set to modify the entire image. This setup approximates current edit transfer workflows where users manually propagate each edit from source to target view by specifying the mask and image adjustment parameters from scratch. We also created an *automatic* version of the target document where the parameters and mask for each adjustment layer are initialized with the results of our automatic edit-transfer method. For each scene, we asked participants to transfer the edits using both the manual and automatic target documents to produce a pair of edits $(B'_{\text{man}}, B'_{\text{auto}})$. We counterbalanced the order of the tasks to account for the potential learning effects from performing the edits twice.

We instructed users to complete the tasks as quickly as possible and recorded their completion times. To limit the duration of each session, we capped each task at ten minutes and alerted participants when they started to run out of time. After each task, we asked users to rate how well their B' matched A' as well as the perceived difficulty of the task on a 5-point scale. At the end of the session, we also asked whether they preferred the manual or automatic condition. Finally, in addition to these self-assessments, we obtained external judgements on the relative quality of each pair of user-generated edits $(B'_{\text{man}}, B'_{\text{auto}})$ using the same 2AFC design as the perceptual study described above.

Qualitative Feedback. Overall, participants expressed a clear preference for the automatic condition over the purely manual workflow. Amongst the 16 users, 14 indicated that they preferred the automatic version. They noted that working from the automatically transferred edits saved time and effort, even when they had to refine the masks and adjustment parameters. Our observations of the editing sessions support these sentiments; in the automatic condition, users spent far less time creating masks compared to the manual condition. The two participants who preferred the manual condition complained that they found it difficult to understand how some of the automatically-generated edits worked. However, both noted that they would probably prefer the automatic condition if they had created the original edits in the source view (which would typically be the case in real-world scenarios).

The self-assessments on the quality of edits and the difficulty of the tasks also

clearly favour the automatic condition. For the Juice scene, only one of the 16 participants felt that the manual condition produced a better result than the automatic condition, and only three participants found the manual task easier than the automatic version. For the Car scene, two participants felt that their manual result was better, and one found the manual task easier.

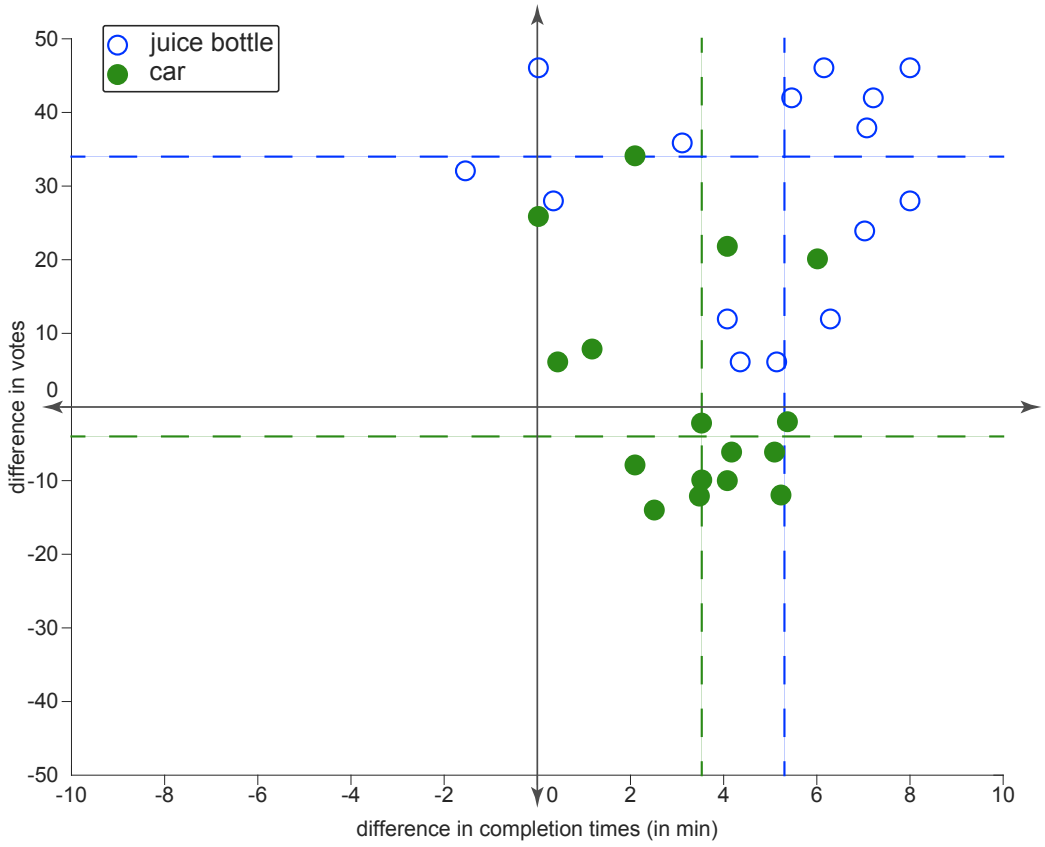


Figure 5.13: Quality vs. Time: Scatterplot showing the difference in completion times and difference in number of votes. The dotted lines show the median value for each axis. The juice bottle (blue) results show significant improvement in completion time and quality. The car (green) has significant improvement in completion time but in quality the results are varied.

Quality versus Completion Time. The task completion times and external quality judgments also support the qualitative findings. We visualize this data by encoding each $(B'_{\text{man}}, B'_{\text{auto}})$ pair generated by a given participant as a single (x,y) data point where x represents the difference in completion times and y represents the difference in the number of votes from the 2AFC comparison between the two conditions. In particular, $x = T_{B'_{\text{man}}} - T_{B'_{\text{auto}}}$, where T is completion time, and $y = V_{B'_{\text{auto}}} - V_{B'_{\text{man}}}$,

where V is the number of votes. Using this encoding, Figure 5.13 provides a rates quality versus completion times for the two scenes.

The fact that most points lie in the top right quadrant indicates that users were generally faster and produced higher quality edits when starting with our automatically transferred edits. However, there are some differences in the relative quality of the manual and automatic results across the two scenes. For the Juice scene, all the automatic results received more votes, but for the Car scene, the votes are more evenly distributed. We believe the reason for this discrepancy is that the masks for the Car edits were easier to specify manually than the masks for the Juice edits, some of which required more careful brushing. Still, it is important to note that the automatic Car edits were at least comparable in quality to the manual edits, and participants consistently completed the edits much more quickly in the automatic condition, which is a key benefit of our approach.

5.6 Closing Remarks

This chapter presented an assistive tool for transferring image-based edits made to multichannel renderings to global scene variations. The method allows users to apply an edit once to a rendering of one scene configuration, then automatically transfer the edit to another variation of the scene. To enable the tool we use multiple representations of the original rendered scene, these are both photometric and non-photometric render channels. The user specified edit is then used and analysed to find correlations between the various scene representations. From this multimodal correlation analysis we learn a channel weighting for each of the scene representation and use this to transfer the edit in a weighted image analogies formulation. We also presented a new editing workflow that more easily allows users to find the right channel they would like to edit. This method again works by finding correlations between channels using a user specified region of interest. To ensure that we meet the objectives set out in Chapter 1 we conducted a user study showing that our method is more time efficient at transfer edits compared to the existing workflow. Additionally, we compared our transfer method with existing algorithms demonstrating our

transferred edits are more accurate. Finally, in qualitative feedback from user study participants a clear preference to our workflow compared to the exiting one was shown. We are happy that our tool meets the objectives we outlined for assistive tools.

In the final chapter, Chapter 6, we conclude the thesis by summarising the findings of the research presented and discuss potential future work.

Chapter 6

Conclusion

In this thesis, three different examples of assistive tools for creating visual content have been discussed. The three tools all use multiple representations of a reference scene and some form of multimodal correlation analysis of these representations to power the tools. In this final chapter the key contributions from these tools are summarised and for each of the tools specific future work is discussed. This is followed by a section discussing future work more broadly in the area of assistive tools.

6.1 Summary

Image Degradation Model. In Chapter 3 we presented a scene abstraction and image degradation model for single RGB-D images. Central to the method is understanding the correlations between the scene represented as an RGB photograph and as a depth image. Using this understanding, we demonstrated how a variety of objects, or even groups of objects, can be approximated by simple planar proxies created out of rough depth information loosely synchronized with RGB information. These proxies can then be used to determine occlusions in a scene and assist with image completion in these occluded regions. This scene abstraction allows for an image degradation model to be created that captures the confidence in the quality of the image completion step. We use the model to assist the user in performing edits. We demonstrated the use of the degradation model in the context of parallax photography from single images.

In the future, we would like to further explore applications of the image degradation model. One particular area of interest is using it to create a smart interface for image editing. Such an interface could allow the user to perform 3D edits in the scene - by way of geometric proxies - and have the output degradation evaluated. If the degradation is high the system could suggest a similar alternative edit by moving objects in the scene or adjusting the camera pose to one that has less degradation.

How2Sketch. In Chapter 4 we presented How2Sketch, a system that automatically generates *easy-to-follow* tutorials for drawing part-segmented man-made 3D objects from selected views. The algorithm creates multiple candidate representations of primitive approximation for the object, before solving a selection problem constrained by geometric relationships found in the input object. The output of the selection optimisation - the selected candidates and their ordering - enables the tool to generate the sketching tutorial. We evaluated our system using a user study, and found that sketches made by following H2S tutorials had more accurate proportions and relative part placements compared to a basic step-by-step tutorial with scaffolding primitives. Users preferred the H2S tutorials over the basic tutorial, giving significantly higher ratings for satisfaction, accuracy, and enjoyment.

One possible future direction is to provide stroke level support to help users draw the final object contours, possibly by explicitly providing guidelines with respect to the scaffold primitives. Another direction would to explore new types of guidelines that can help reduce the number of unguided steps in a tutorial. A very interesting future question is to investigate if H2S really *teaches* users to sketch better by drawing "what you know." While this is the ultimate goal of any sketching tutorial, answering this question will require a much more involved user study where we have to track and quantify user-specific improvements.

Edit Transfer. Finally, in Chapter 5 we developed an interactive editing tool for 2D and 3D editing of rendered 3D scenes, which allows transfer of parametric 2D edits to new views of the scene or scenes with different objects. At the heart of

our method is a new edit-dependent adaptive image analogies method. This requires edit-dependent weighting to be found through understanding the correlation between the user edit and multiple render channels of the scene. We demonstrated that our edit-dependent approach successfully transfers edits for a variety of 3D scenes and 2D touchups, and outperforms prior approaches that rely on dense correspondences that do not take into account the user edits. Additionally, we evaluated the usefulness of our transfer method in a user study. Our tool opens up the possibility of additional functionalities that blur the boundary between 2D and 3D for editing, such as propagating 2D and 3D edits to automatically inferred 3D scene properties from the background photograph, e.g., to transfer edits to object shadows that affect others depicted in the background.

Our approach to the edit transfer problem is a practical one that fits with the existing workflow of digital artists that is intuitive to them. A different way to approach the problem would be to pose it as one of inverse rendering. Such a tool could allow the artist to make in image-based edit then automatically infer the 3D scene parameters that would result in that image. This would mean the edit transfer problem is intrinsically solved. However, the artists would then be limited to only physically-valid edits.

6.2 Future Work

In addition to the application specific areas of future work suggested in Section 6.1, there are many interesting avenues of research into assistive tools for visual content creation. In all the the tools proposed in the previous chapters the problems have been approached using limited input with either one or no exemplars. In future work it would be interesting to explore using data-driven approaches to assistive tools, however, this will come with new challenges to overcome.

One challenge to overcome is giving the artist more control of data-driven tools. There have been some very successful applications of using artist input to then generate an image. For example the network proposed in Pix2Pix [112] has been used to generate images of cats based on an artist sketch. However, the gener-

ated images, you could argue, are quite difficult to the artist to control. For example if they wanted to keep one feature but change others this is challenging without providing a perfect sketch. Some work has attempted to look into this problem giving artists more control. For example Delanoy et al. [113] use multiple sketches from different viewpoints in a data driven sketch-based modelling application. In the context of style transfer Gatys et al. [114] decompose style into different perceptual factors to allow more artist control. However, it could be argued that more artist control needs to be provided to these data-driven tools. One particular area of future research would be to use multimodal input with such tools. For example, input sketches and verbal / written descriptions could be combined to synthesise images or 3D models. The sketch could provide geometric information and the verbal appearance.

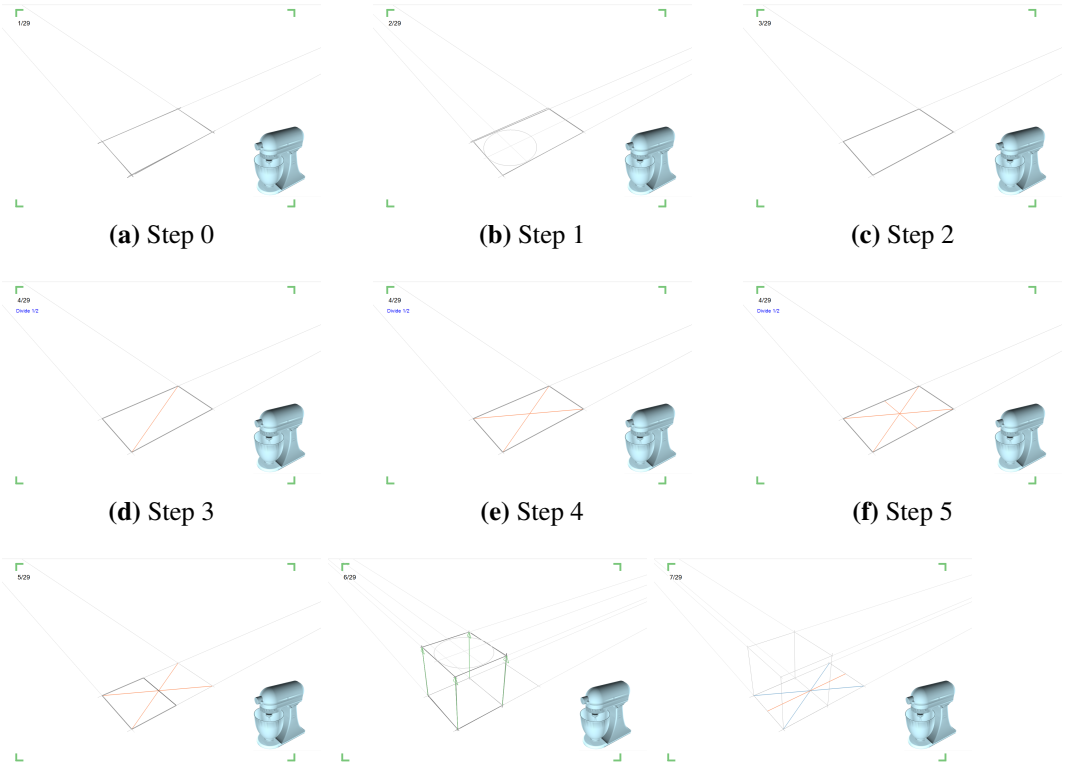
Another challenge for using data-driven approaches is generating the training data or how to pose the problem as one of unsupervised learning. There have been some successful examples of this [112, 113], however, the solutions are not general. For the tools proposed in Chapters 4 and 5 it is not immediately clear how training data could be collected efficiently or how it would be posed as a unsupervised learning problem. Overcoming this problem in a general way, even for a group of interactive tasks such an image editing or 3D compositing would be a great step forward in the field.

Given the future research directions proposed here and in Section 6.1, it is clearly a field with many avenues of future work. Moreover, given the history of the development of the tools used by artist and the number of challenges people still face, it is hard to imagine that the tools used today will be the tools used in the future. This thesis has argued that assistive tools is an interesting direction for interactive computer graphics tools and one that has potential to cross from academic research to being used in industry.

Appendix A

How2Sketch Tutorials

This appendix contains some example How2Sketch tutorials from Chapter 4. For additional, higher resolution, examples please see the project webpage.

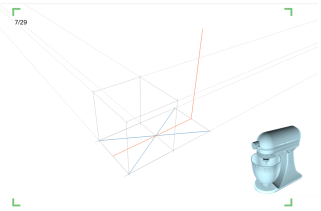


(a) Step 0 **(b)** Step 1 **(c)** Step 2

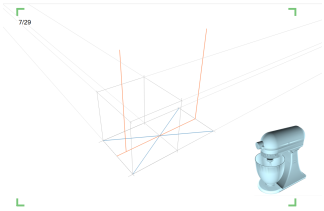
(d) Step 3 **(e)** Step 4 **(f)** Step 5

(g) Step 6 **(h)** Step 7 **(i)** Step 8

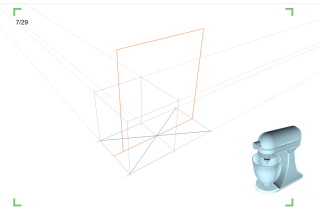
Figure A.1: How2Sketch Mixer Tutorial Experience Setting Novice Part 1



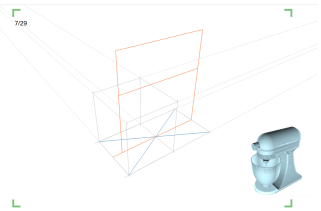
(a) Step 9



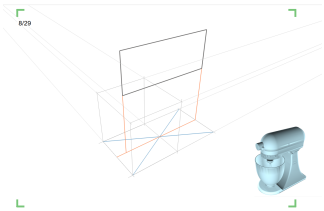
(b) Step 10



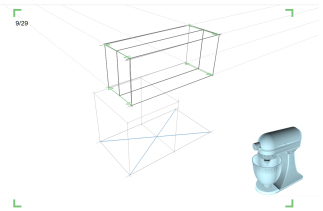
(c) Step 11



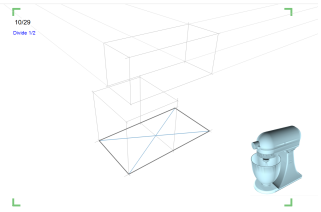
(d) Step 12



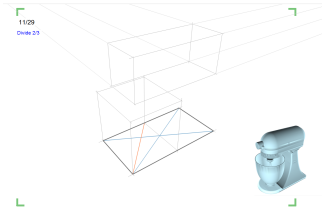
(e) Step 13



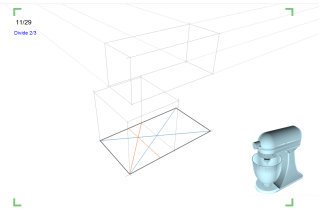
(f) Step 14



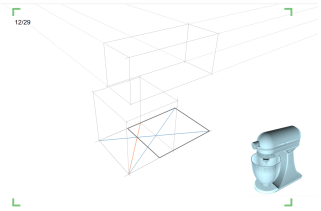
(g) Step 15



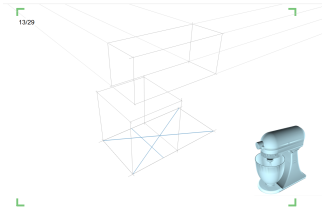
(h) Step 16



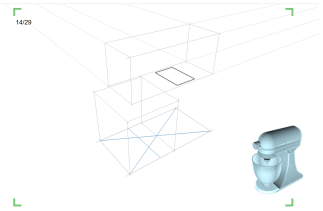
(i) Step 17



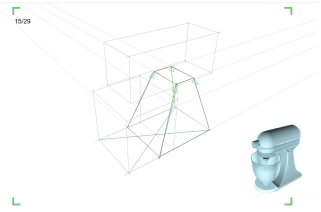
(j) Step 18



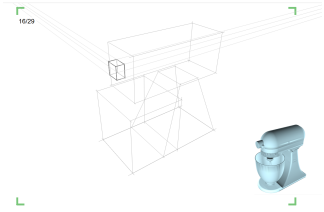
(k) Step 19



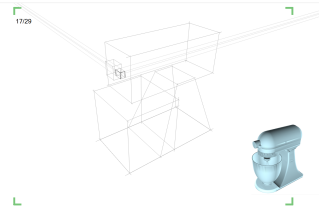
(l) Step 20



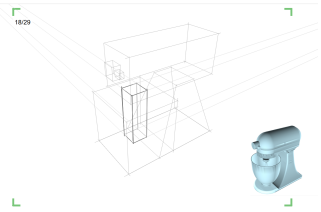
(m) Step 21



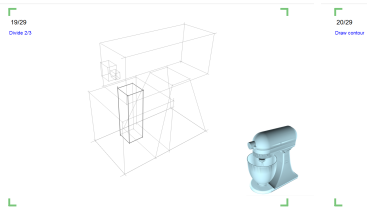
(n) Step 22



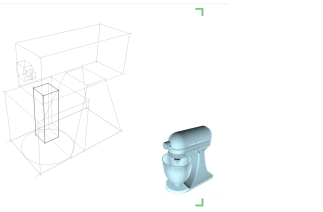
(o) Step 23



(p) Step 24

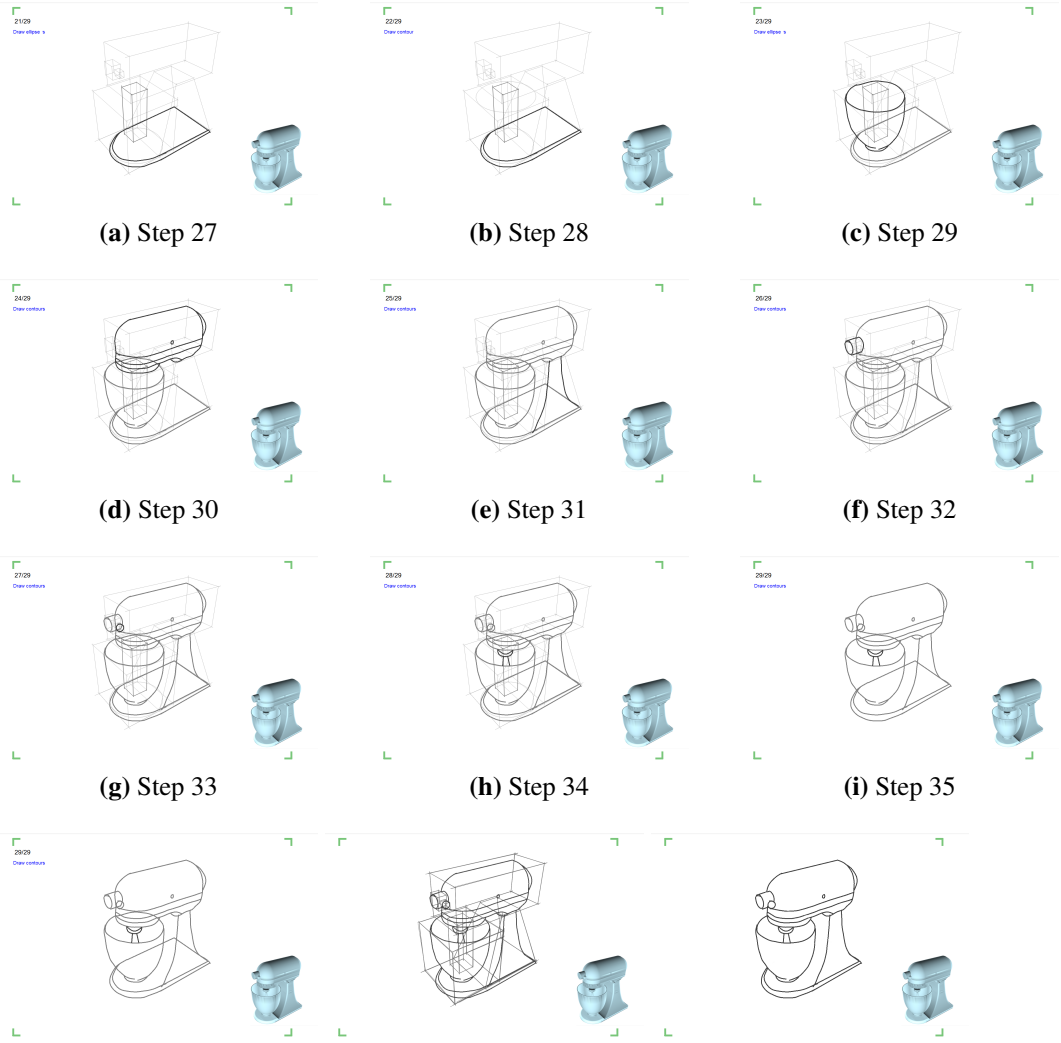


(q) Step 25



(r) Step 26

Figure A.2: How2Sketch Mixer Tutorial Experience Setting Novice Part 2



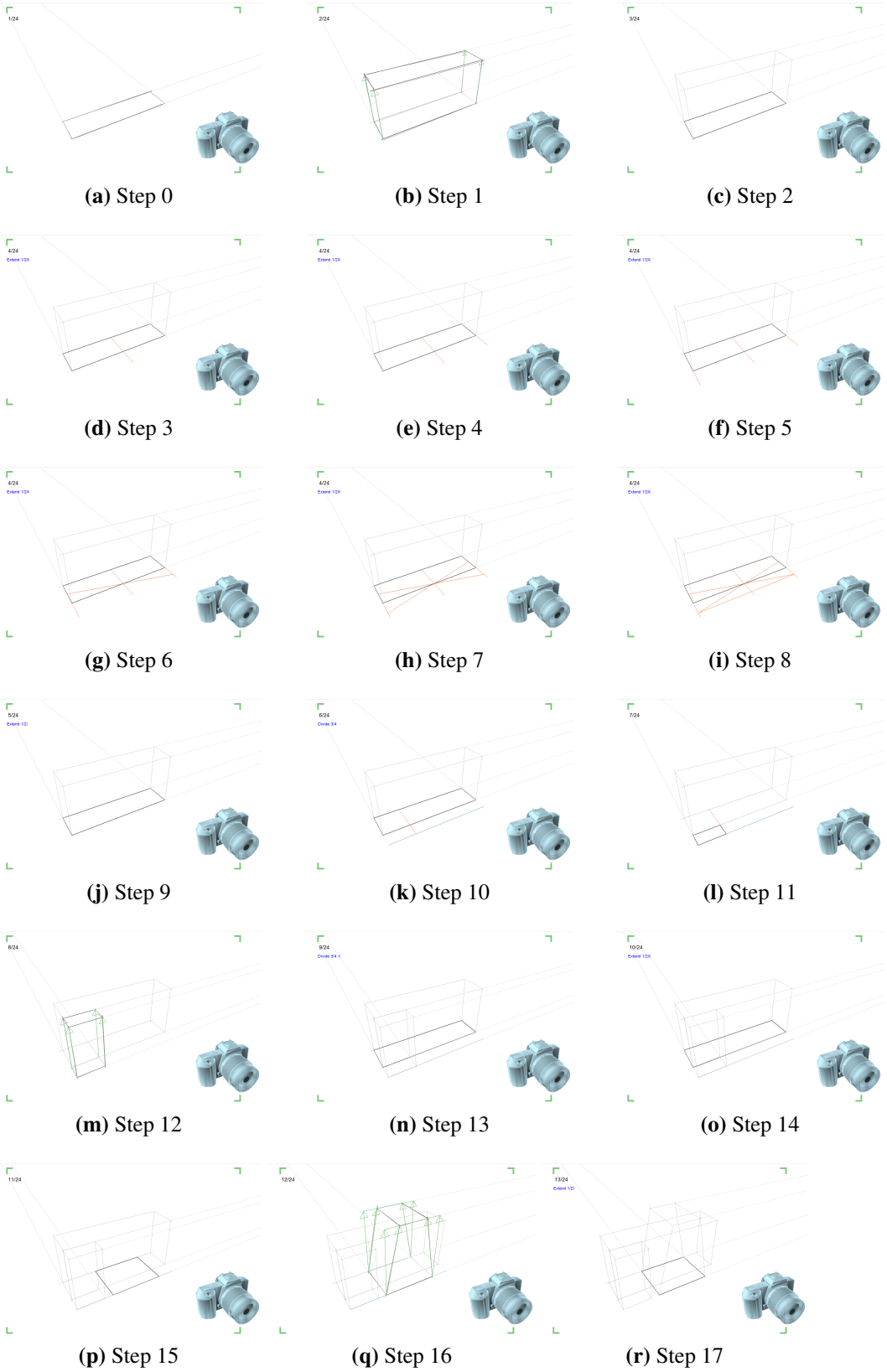
(a) Step 27 **(b)** Step 28 **(c)** Step 29

(d) Step 30 **(e)** Step 31 **(f)** Step 32

(g) Step 33 **(h)** Step 34 **(i)** Step 35

(j) Step 36 **(k)** Step 37 **(l)** Step 38

Figure A.3: How2Sketch Mixer Tutorial Experience Setting Novice Part 3



(a) Step 0 **(b)** Step 1 **(c)** Step 2

(d) Step 3 **(e)** Step 4 **(f)** Step 5

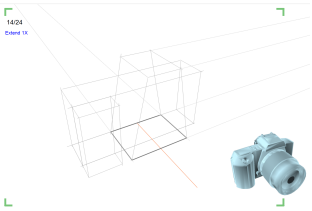
(g) Step 6 **(h)** Step 7 **(i)** Step 8

(j) Step 9 **(k)** Step 10 **(l)** Step 11

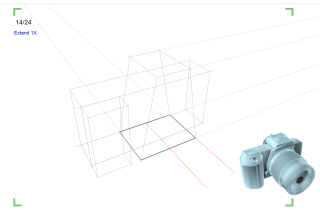
(m) Step 12 **(n)** Step 13 **(o)** Step 14

(p) Step 15 **(q)** Step 16 **(r)** Step 17

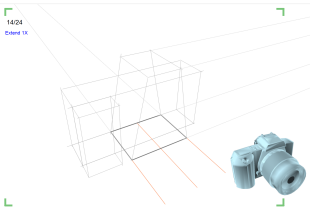
Figure A.4: How2Sketch Camera Tutorial Experience Setting Apprentice Part 1



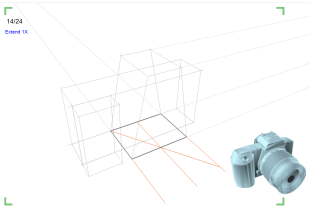
(a) Step 18



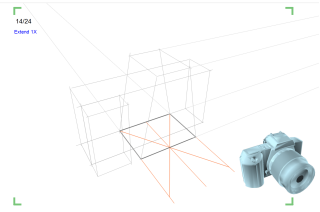
(b) Step 19



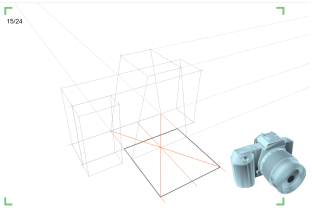
(c) Step 20



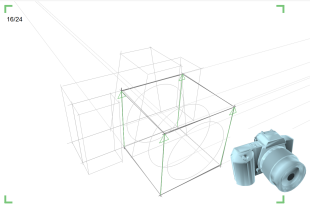
(d) Step 21



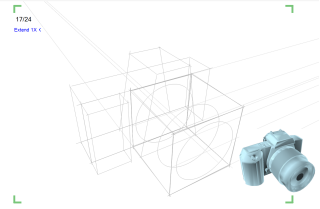
(e) Step 22



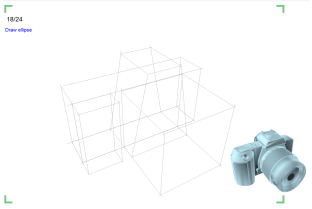
(f) Step 23



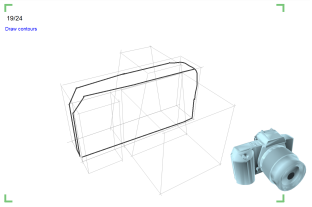
(g) Step 24



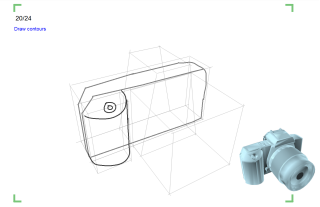
(h) Step 25



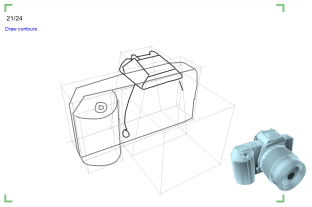
(i) Step 26



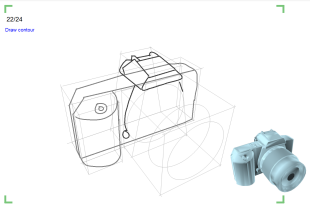
(j) Step 27



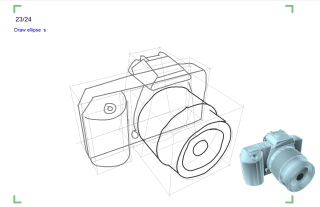
(k) Step 28



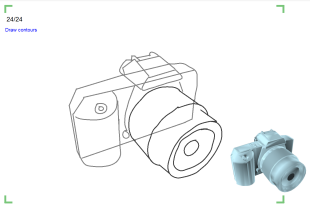
(l) Step 29



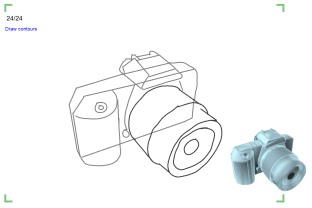
(m) Step 30



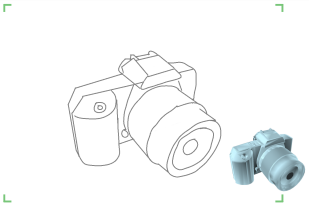
(n) Step 31



(o) Step 32

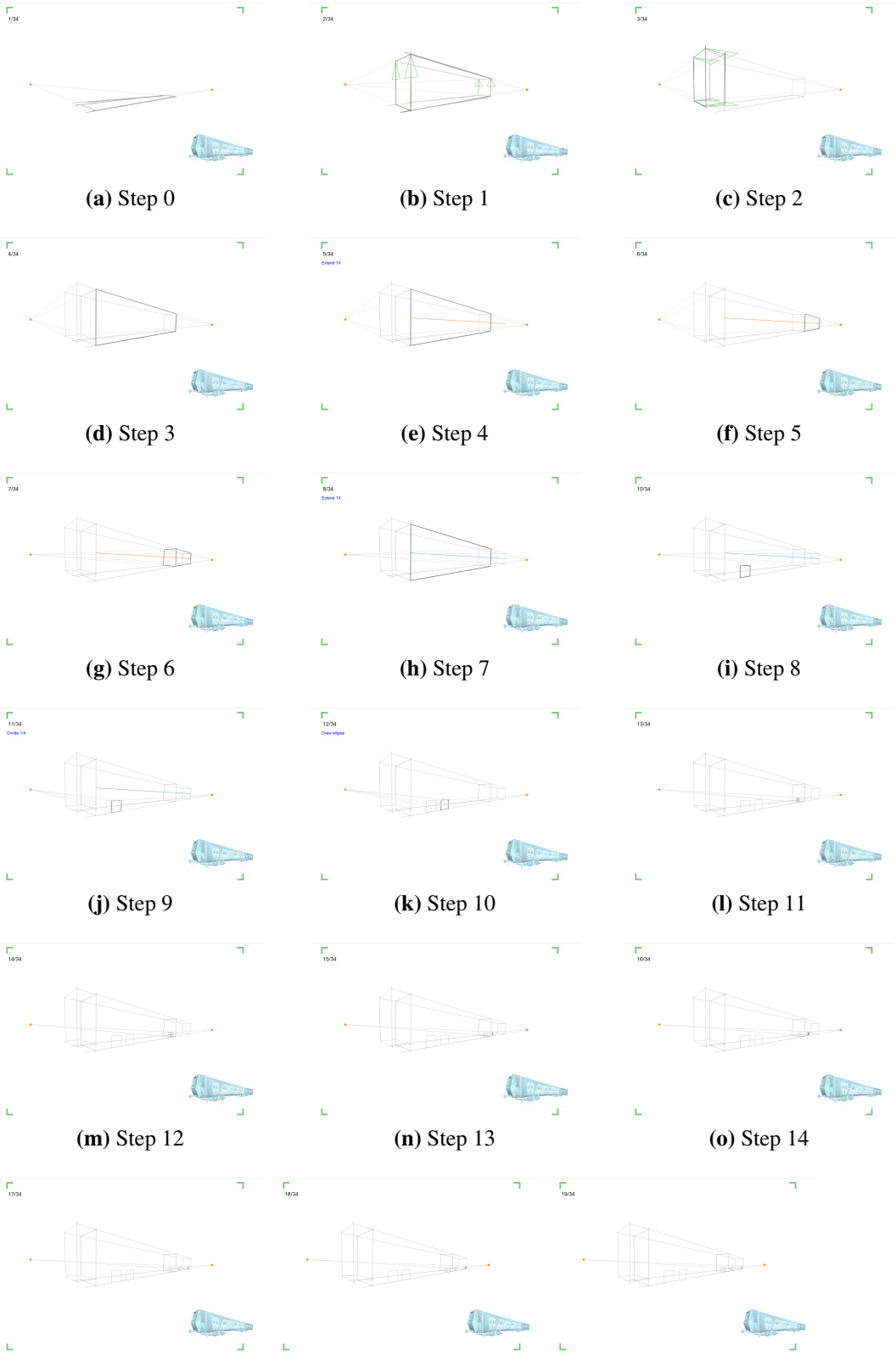


(p) Step 33



(q) Step 34

Figure A.5: How2Sketch Camera Tutorial Experience Setting Apprentice Part 2



(a) Step 0 **(b)** Step 1 **(c)** Step 2

(d) Step 3 **(e)** Step 4 **(f)** Step 5

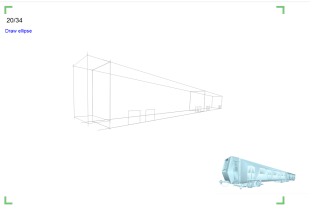
(g) Step 6 **(h)** Step 7 **(i)** Step 8

(j) Step 9 **(k)** Step 10 **(l)** Step 11

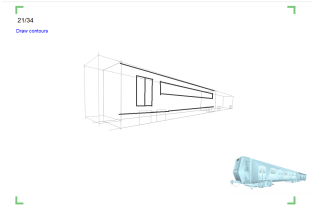
(m) Step 12 **(n)** Step 13 **(o)** Step 14

(p) Step 15 **(q)** Step 16 **(r)** Step 17

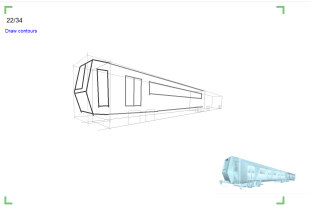
Figure A.6: How2Sketch Train Tutorial Experience Setting Master Part 1



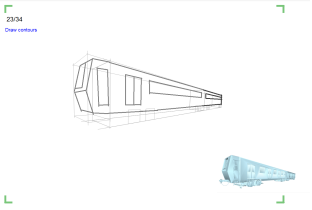
(a) Step 18



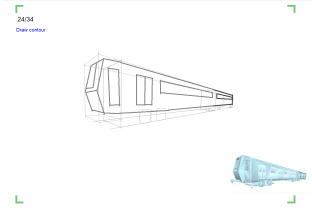
(b) Step 19



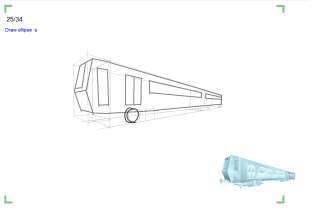
(c) Step 20



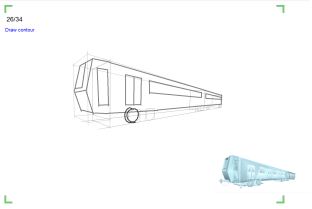
(d) Step 21



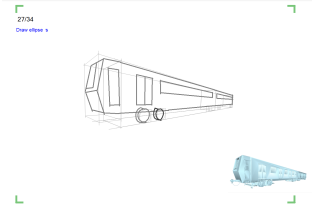
(e) Step 22



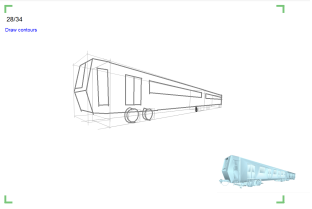
(f) Step 23



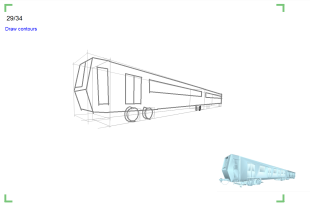
(g) Step 24



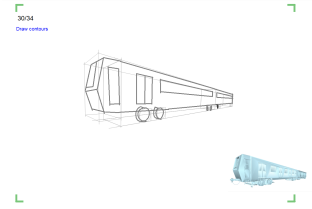
(h) Step 25



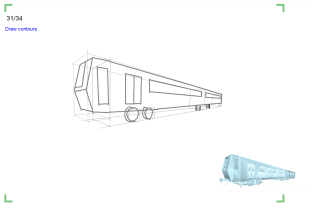
(i) Step 26



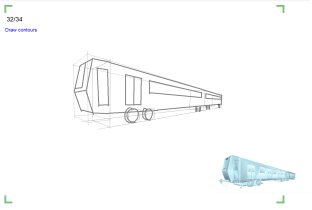
(j) Step 27



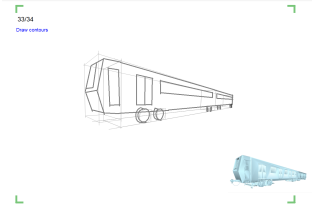
(k) Step 28



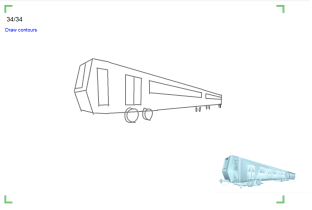
(l) Step 29



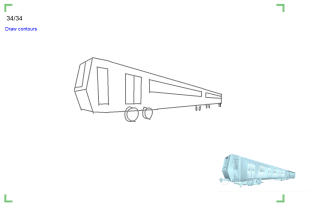
(m) Step 30



(n) Step 31

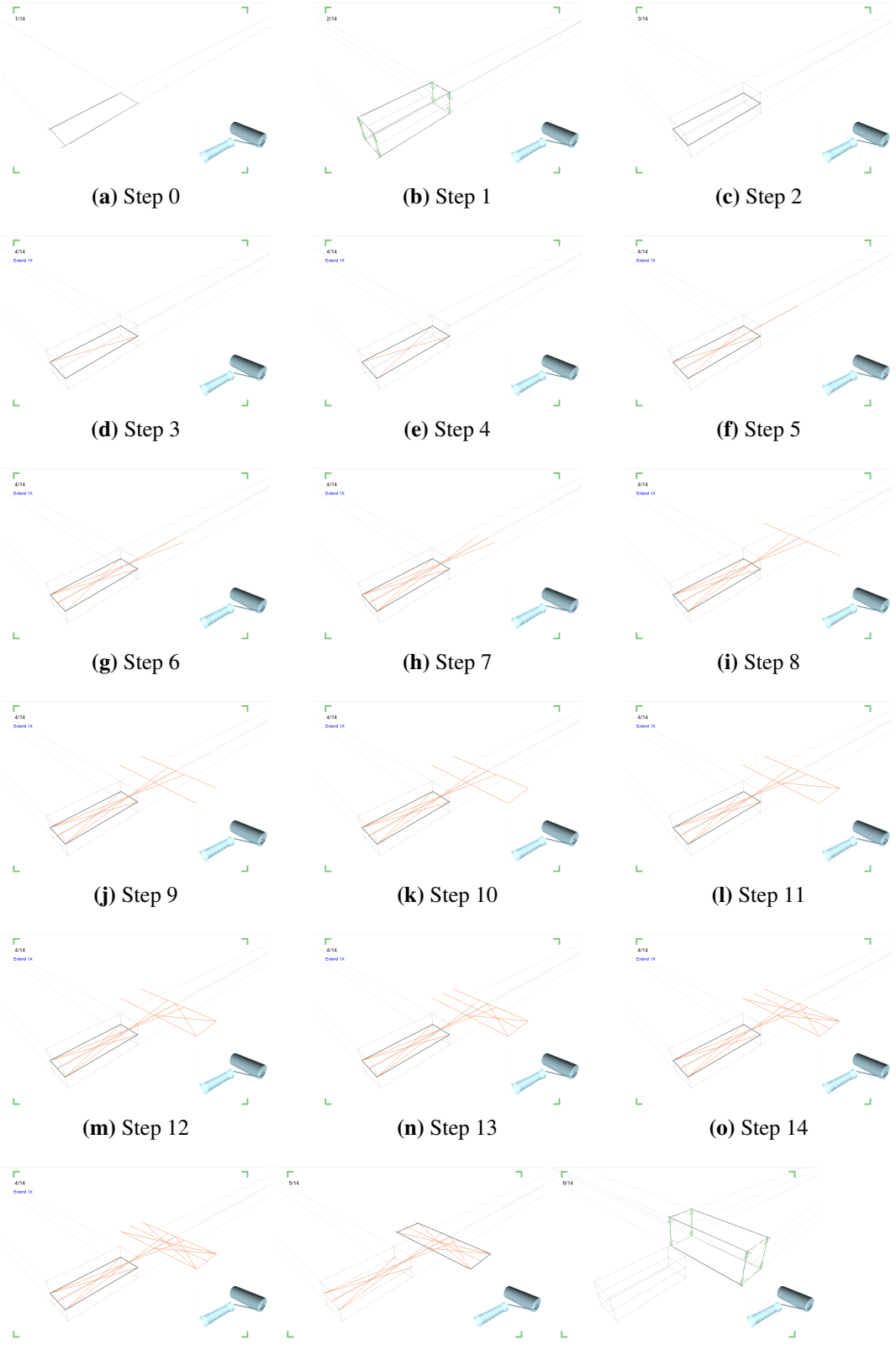


(o) Step 32



(p) Step 33

Figure A.7: How2Sketch Train Tutorial Experience Setting Master Part 2



(a) Step 0 **(b)** Step 1 **(c)** Step 2

(d) Step 3 **(e)** Step 4 **(f)** Step 5

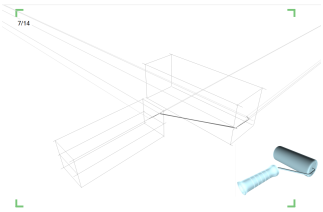
(g) Step 6 **(h)** Step 7 **(i)** Step 8

(j) Step 9 **(k)** Step 10 **(l)** Step 11

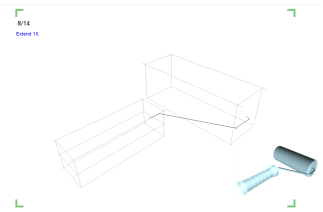
(m) Step 12 **(n)** Step 13 **(o)** Step 14

(p) Step 15 **(q)** Step 16 **(r)** Step 17

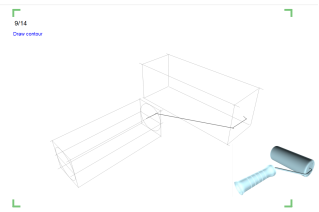
Figure A.8: How2Sketch Roller Tutorial Experience Setting Novice Part 1



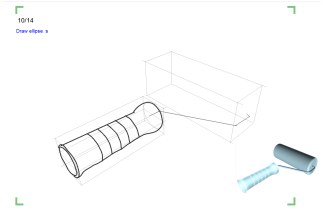
(a) Step 18



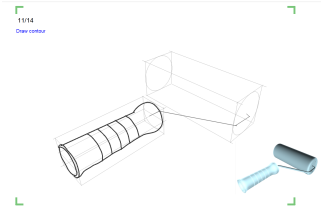
(b) Step 19



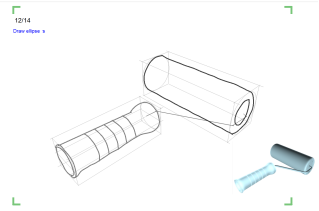
(c) Step 20



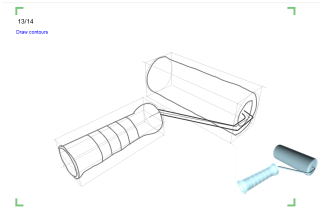
(d) Step 21



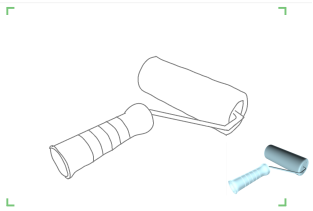
(e) Step 22



(f) Step 23



(g) Step 24



(h) Step 25

Figure A.9: How2Sketch Roller Tutorial Experience Setting Novice Part 2

Appendix B

Derivation of Optimal Channel Selection

As formulated in Section 5.4.1, our goal of determining a channel selection vector $\mathbf{x} = (x_1, \ldots, x_N)^T$ for a user-selected ROI can be cast as:

$$\max_{\mathbf{x}} \sum_j \left(\mathbf{x}^T \mathbf{d}_j\right)^2 \qquad \text{s.t.} \quad \mathbf{x}^T \mathbf{x} = 1.$$

Using Lagrangian multiplier λ, we can reformulate the above as $\max_{\mathbf{x}} E(\mathbf{x})$ where,

$$E(\mathbf{x}) := \sum_j \left(\mathbf{x}^T \mathbf{d}_j\right)^2 + \lambda(1 - \mathbf{x}^T \mathbf{x}).$$

Simplifying $E(\mathbf{x})$, we get:

$$\begin{aligned} E(\mathbf{x}) &= \mathbf{x}^T (\sum_j \mathbf{d}_j \mathbf{d}_j^T) \mathbf{x} + \lambda(1 - \mathbf{x}^\intercal \mathbf{x}) \\ &= \mathbf{x}^T \mathbf{C} \mathbf{x} + \lambda(1 - \mathbf{x}^T \mathbf{x}) \end{aligned}$$

with $C = \sum_j \mathbf{d}_j \mathbf{d}_j^T$. In order to find extrema of $E(\mathbf{x})$, we set

$$\frac{\partial E(\mathbf{x})}{\partial \mathbf{x}} = 2\mathbf{C}\mathbf{x} - 2\lambda\mathbf{x} = 0.$$

Thus, to find an extrema of $E(\mathbf{x})$ we have to select an eigenvector of C. Let $\mathbf{x}_e$ be

such an eigenvector, i.e., $C\mathbf{x}_e = \lambda \mathbf{x}_e$. For such a choice, $E(\mathbf{x})$ evaluates to

$$E(\mathbf{x}_e) = \mathbf{x}_e^T (C\mathbf{x}_e) = \mathbf{x}_e^T (\lambda \mathbf{x}_e) = \lambda,$$

where we used $\mathbf{x}_e^T \mathbf{x}_e = 1$ since $\mathbf{x}_e$ is an eigenvector.

Thus, to maximize $E(\mathbf{x})$, we have to pick the eigenvector with the *largest eigenvalue* among the N eigenvectors of C.

Appendix C

Renderer Specific Augmented Render Channels

The rendered of a scene using VRay took, on average, 10 minutes and Mitsuba on average took 8 hours. The list of Augmented Render Channels used for these examples can be seen in Table C.1. Note that Mitsuba cannot separate lighting effects per light source but has a diverse set of non-photometric channels enabling our transfer method to work.

Table C.1: Augmented Render Channels: The photometric render channels 1-3 are optionally rendered per light source. Geometry channels 4 -7 are rendered per light source. VRay object channels inner- and outer-distance transforms (DT) are generated in 2D using the object masks. We further augment the photometric render channels by adding the log of each channel. The Mitsuba outgoing- and incoming ray channels are the average rays for each pixel. We use ALL PHOTOMETRIC to refer to all channels in the Photometric column for a given renderer.

	Photometric	**Geometry**	**Object**
		VRay	
1.	SPECULAR	Z-DEPTH	OBJECT MASK
2.	DIFFUSE	NORMALS	INNER-DT
3.	REFLECTION	VIEW DIRECTION	OUTER-DT
4.	REFRACTION	NORMAL VIEW COS	
5.	GLOBAL ILLUM	LIGHT DIRECTION	
6.	SHADOW	NORMAL LIGHT COS	
7.	CAUSTICS	HALF ANGLE	
8.	SELF ILLUM		
9.	CAUSTICS		
10.	SUBSURF SCATTER		
		Mitsuba	
11.	DIRECT SPECULAR	Z-DEPTH	OBJECT MASK
12.	DIRECT DIFFUSE	NORMALS	PRIMITIVE ID
13.	GLOBAL ILLUM SPECULAR	OUTGOING RAY	BSDF TYPE
14.	GLOBAL ILLUM DIFFUSE	INCOMING RAY	SAMPLE TYPE
15.	SUBSURF SCATTER	OR NORMAL COS	ALBEDO
16.	ENVIRONMENT MAP	IR NORMAL COS	
17.	EMITTER	HALF ANGLE	
18.	CAUSTICS	CURVATURE	
19.		RAW DIRECT ILLUM	

Appendix D

Adjustment Parameters Implementation Details

Exposure has a single scalar value $\theta \in [-10, 10]$ and adjusts the input as $f_\theta(B_i, p) = B_i(p) * 2^\theta$.

Levels require several parameters $\theta = \left(\theta_{in}^{min}, \theta_{in}^{max}, \theta_{out}^{min}, \theta_{out}^{max}, \theta^\gamma\right) \in [0,1]^5$ and applied to a pixel as

$$f_\theta(B_i, p) = \left(\frac{B_i(p) - \theta_{in}^{min}}{\theta_{in}^{max} - \theta_{in}^{min}}\right)^{\frac{1}{\theta^\gamma}} \left(\theta_{out}^{max} - \theta_{out}^{min}\right) + \theta_{out}^{min}, \tag{D.1}$$

where γ allows for a single scalar parameter in the range $\theta \in [0, 10]$ and adjusts a pixel as $f_\theta(B_i, p) = B_i(p)^\theta$.

Hue, Saturation and Lightness changes are adjusted using the parameters $\theta = (\theta_h, \theta_s, \theta_l) \in ([-180, 180], [-180, 180], [-100, 100])$ and is applied to a pixel in HSL domain as $f_\theta(B_i, p) = B_i(p) + \theta$.

Gaussian blur is parameterized using a kernel size and standard deviation, $\theta = (\theta_x, \theta_\sigma)$ which is applied to a pixel by,

$$f_\theta(B_i, p) = \sum_{k=-\theta_x}^{\theta_x} \sum_{j=-\theta_x}^{\theta_x} G_{\theta_\sigma}(k, j) B_i(p') \tag{D.2}$$

where $p' = p + q$, $q = \begin{bmatrix} k & j \end{bmatrix}^\intercal$ and

$$G_{\theta_\sigma}(x,y) = \frac{1}{2\pi\theta_\sigma^2} e^{-\frac{x^2+y^2}{2\theta_\sigma^2}} . \tag{D.3}$$

Appendix E

Scene, Edit and Transfer Descriptions

In this appendix we outline the scene rendering setup, the image-based edits performed, and the non-zero weighted render channels used to transfer edits to the target view.

Car. The car scene was rendered with an area light source above the car and a point light source behind the camera. We made four image-based edits: (i) emphasize reflection on the ground; (ii) remove specular glare on the headlight; (iii) reduce specular reflection on the windscreen wipers; and (iv) reduce specular highlight on the bumper. We created many 3D scene variations: 3$\times$viewpoint change, change in material, add duplicate geometry and new geometry. For each edit, the following channels were selected for transfer and reconstruction: (i) LOG SPECULAR (light source 1), LOG DIFFUSE (light source 1), LOG SPECULAR (light source 2), LOG REFLECTION; (ii) SPECULAR (light source 1), LOG SHADOW, NORMALS, NORMAL VIEW COS; (iii) SPECULAR (light source 2), REFLECTION, GLOBAL ILLUM; (iv) SPECULAR (light source 1), LOG SHADOW, LOG REFLECTION.

Juice Bottle. The bottle is lit by two point light sources, one behind the object and the other above and in front. We made three image-based edits: (i) remove specular glares around the outside of the bottle and push the specular highlight away from the bottle contour; (ii) remove harsh reflection on the bottom of the bottle; (iii) emphasize the label. In the target view, we changed the viewpoint. The following

render channels were selected: (i) LOG SPECULAR, OUTER-DT, GLOBAL ILLUMINATION; (ii) REFLECTION, LOG REFLECTION; (iii) LOG REFLECTION, LOG SPECULAR, REFLECTION, SELF ILLUM.

Wine Bottle and Glass. The scene has 4 light sources: two area lights (one red and one white) and two point lights (one in front and one behind the wine and glass). We made five image-based edits: (i) remove big white reflection from the white area light reflecting off the table and wall; (ii) remove big red reflection from the red area light on the table; (iii) remove reflection of the wine glass on the back wall; (iv) emphasize bottom half of the white reflection on the wine bottle so it matches the reflection above; (v) remove distracting red refraction on the wine glass. In the target view, we changed the object geometry and viewpoint. The following render channels were selected: (i) NORMALS, LOG REFLECTION (light 1), REFLECTION (light 1); (ii) REFLECTION (light 2), NORMALS; (iii) REFLECTION (light 1), LOG REFLECTION (light 2), LOG REFLECTION (light source 4), LOG REFRACTION (light 2), LOG REFRACTION (light 3); (iv) REFRACTION (light source 4), LOG REFLECTION (light 4), SHADOW, LOG GLOBAL ILLUM; (v) DIFFUSE (light 2), LOG REFRACTION (light 2), LOG REFLECTION (light 2).

Watch. The scene is lit with a single point light source on the opposite side of the watch to the camera (not visible to camera). The image-based edits were (i) increase brightness of the watch face; (ii) add exaggerated highlight on the watch face; (iii) make metal material more reflective. In the target view, the watch was rotated and translated on the table. The selected render channels were (i) LOG REFLECTION, LOG BACKGROUND; (ii) LOG REFRACTION, LOG REFLECTION, LOG DIFFUSE, LOG SHADOW, LOG BACKGROUND; (iii) LOG GLOBAL ILLUM, LOG SHADOW, LOG BACKGROUND, LOG REFRACTION.

Whiskey. The original scene is rendered with a background photo of a beach with a directional light source above and to the left of the bottle. The image-based edits were (i) add halo effect around the outline of the bottle; (ii) emphasize the label to make it more visible. In the target view, we chose a different background image, the objects have been rotated and translated, and the light source position has moved to

the right of the bottle. The following render channels were selected for the transfer: (i) OUTER-DT, NORMALS, LOG BACKGROUND; (ii) LOG GLOBAL ILLUM, DIFFUSE.

Dragon. There are three lights in this scene: two point lights on either side of the Dragon and a soft area light above. The image-based edits were (i) increase specular highlights to emphasize the Dragon's curvature; (ii) boost the GI channel inside the Dragon's body to give a glowing effect. In the target views, we rotated the Dragon 360°. The selected render channels were (i) LOG SPECULAR, NORMALS, LOG REFLECTION, DIFFUSE; (ii) LOG GLOBAL ILLUM, INNER-DT, LOG SHADOW.

Backpacks. The scene is lit from above by a single point light source in between the bags and the camera position. Three image-based edits were made: (i) make the fabric appear darker; (ii) remove unwanted highlight on the side of the bag; (iii) make creases of the bag orange matching the handle color. We modified the 3D scene by rotating and translating the two bags. The selected render channels were (i) LOG DIFFUSE, DIFFUSE, GI; (ii) LOG SPECULAR, SPECULAR, LOG REFLECTION, LOG SHADOW; (iii) LOG REFLECTION, NORMALS, LOG SHADOW, LOG DIFFUSE, LOG SPECULAR.

3D Text. The 3D text is composited into a background photo with a single point light behind the text. The material of the text is translucent. We edited the source view to emphasize translucency by increasing the exposure at some of the edges of the text. In the target view, the text has been rotated and translated, in addition to changing the background image. To transfer the edit, our method selects the LOG SUBSURF SCATTER, SUBSURF SCATTER, LOG DIFFUSE, LOG SPECULAR and GLOBAL ILLUM channels.

San Miguel. The scene's lighting comes from an environment map, which is only visible thought the atrium. The image-based edits were (i) increasing the exposure of the indirect global illumination channel to make the region in shade more visible (ii) adjusting the levels on the tree leaves to make them more prominent and green (iii) adjusting the hue, saturation and lightness of the wall. In the target views the camera moves and rotates revealing new geometry. To transfer the edits the

selected channels were (i) ALBEDO, SAMPLE TYPE, LOG GLOBAL ILLUM DIFFUSE, NORMALS (ii) ALBEDO, LOG GLOBAL ILLUM DIFFUSE and PRIMITIVE ID (iii) ALBEDO, SAMPLE TYPE, and NORMALS.

Bathroom. The scene's lighting comes from an environment light outside, which is coming though the windows. The image-based edits were (i) reducing the exposure of the glass windows to make them appear frosted and (ii) hue, saturation and lightness of the floor to make it a different color. In the target view the camera is translated up and towards the left. To transfer the edits the selected channels were (i) BSDF TYPE, RAW DIRECT ILLUM and IR NORMAL COS (ii) BSDF TYPE, RAW DIRECT ILLUM, Z-DEPTH and IR NORMAL COS.

Kitchen. The scene's lighting comes from a sphere area light behind the camera position and several smaller point lights visible in the scene. The image-based edits were (i) blurring light sources, (ii) adjusting hue/saturation/lightness of the work surface, (iii) adjusting hue/saturation/lightness of the mug, (iv) reducing rendering noise on the pot using gamma correction and (v) reducing the specular highlights on the pot using the exposure parameter. In the target view the camera was translated. To transfer the edits the selected channels were (i) ALBEDO, curvature, RAW DIRECT ILLUM, SAMPLE TYPE (ii) ALBEDO, NORMALS, primitive id, SAMPLE TYPE and Z-DEPTH (iii) ALBEDO, PRIMITIVE ID, SAMPLE TYPE and Z-DEPTH (iv) ALBEDO, RAW DIRECT ILLUM, Z-DEPTH (v) BSDF TYPE, RAW DIRECT ILLUM, Z-DEPTH.

Skulls. The scene is lit by a large area light directly above the skulls. The image-based edits were (i) blurring the background skull to create a depth of field effect (ii) changing the hue, saturation and gamma to change the color and emphasize the reflections on the ground plane (Note this and edit (i) together is physically invalid, typical for our target application) (iii) adjusting the hue, saturation and lightness in the foreground skulls' eye sockets to make them appear to glow. In the target view the skulls were rotated and translated into a new configuration. To transfer the edits the following channels were selected (i) LOG DIFFUSE, shadow, LOG SHADOW and Z-DEPTH (ii) LOG DIFFUSE, LOG REFLECTION, LOG SHADOW (iii) LOG

DIFFUSE, LIGHT DIRECTION, NORMAL LIGHT COS, HALF ANGLE

Instruments. This scene is lit by a white area light above the instruments and a red point light next to the camera. The image based edit (i) was blurring the background saxophone. In the target view the viewpoint was changed and the saxophone rotated. To transfer the edit (i) the LOG GLOBAL ILLUM, LOG DIFFUSE, LOG REFLECTION, INNER-DT and OUTER-DT were selected.

Bibliography

[1] John Berger, Sven Blomberg, Chris Fox, Michael Dibb, and Richard Hollis. Ways of seeing. 1973.

[2] John J. Medina. Brain rules: 12 principles for surviving and thriving at work, home, and school. 2008.

[3] Susan B. Barnes. *An Introduction to Visual Communication: From Cave Art to Second Life*. 2017.

[4] Paul S. Heckbert. Adaptive radiosity textures for bidirectional ray tracing. *SIGGRAPH Comput. Graph.*, 24(4):145–154, September 1990.

[5] James W. Hennessey and Niloy J. Mitra. An image degradation model for depth-augmented image editing. *Symposium on Geometry Processing 2015*, 2015.

[6] James W. Hennessey, Han Liu, Holger Winnemller, Mira Dontcheva, and Niloy J. Mitra. How2sketch: Generating easy-to-follow tutorials for sketching 3d objects. *Symposium on Interactive 3D Graphics and Games*, 2017.

[7] James W. Hennessey, Wilmot Li, Bryan Russell, Eli Shechtman, and Niloy J. Mitra. Transferring image-based edits for multi-channel compositing. *ACM Trans. Graph.*, 36(6):179:1–179:16, November 2017.

[8] Youyi Zheng, Xiang Chen, Ming-Ming Cheng, Kun Zhou, Shi-Min Hu, and Niloy J. Mitra. Interactive images: Cuboid proxies for smart image manipulation. *ACM SIGGRAPH*, 31(4):99:1–99:11, 2012.

[9] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan B Goldman. PatchMatch: A randomized correspondence algorithm for structural image editing. *ACM SIGGRAPH*, 28(3), August 2009.

[10] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. "GrabCut": Interactive foreground extraction using iterated graph cuts. *ACM Trans. Graph.*, 23(3):309–314, August 2004.

[11] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CVPR*, November 2015.

[12] Alex Kendall, Vijay Badrinarayanan, , and Roberto Cipolla. Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. *arXiv preprint arXiv:1511.02680*, 2015.

[13] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[14] Vijay Badrinarayanan, Ankur Handa, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for robust semantic pixel-wise labelling. *arXiv preprint arXiv:1505.07293*, 2015.

[15] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. *arXiv preprint arXiv:1703.06870*, 2017.

[16] Aayush Bansal, Xinlei Chen, Bryan Russell, Abhinav Gupta, and Deva Ramanan. Pixelnet: Representation of the pixels, by the pixels, and for the pixels. *arXiv:1702.06506*, 2017.

[17] Raymond A. Yeh*, Chen Chen*, Teck Yian Lim, Schwing Alexander G., Mark Hasegawa-Johnson, and Minh N. Do. Semantic image inpainting with deep generative models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. * equal contribution.

[18] Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. Globally and Locally Consistent Image Completion. *ACM Transactions on Graphics (Proc. of SIGGRAPH 2017)*, 36(4):107:1–107:14, 2017.

[19] Ming-Ming Cheng, Fang-Lue Zhang, Niloy J. Mitra, Xiaolei Huang, and Shi-Min Hu. Repfinder: finding approximately repeated scene elements for image editing. *ACM SIGGRAPH*, 29(4):83:1–83:8, 2010.

[20] Robert Carroll, Aseem Agarwala, and Maneesh Agrawala. Image warps for artistic perspective manipulation. *ACM SIGGRAPH*, 29(4):127:1–127:9, 2010.

[21] Tao Chen, Zhe Zhu, Ariel Shamir, Shi-Min Hu, and Daniel Cohen-Or. 3-sweep: extracting editable objects from a single photo. *ACM SIGGRAPH Asia*, 32(6):195:1–195:10, 2013.

[22] Yu-Shiang Wong, Hung-Kuo Chu, and Niloy J. Mitra. Smartannotator an interactive tool for annotating indoor rgbd images. *CGF Eurographics*, 2015.

[23] Byong Mok Oh, Max Chen, Julie Dorsey, and Frédo Durand. Image-based modeling and photo editing. *ACM SIGGRAPH*, pages 433–442, 2001.

[24] Youichi Horry, Ken-Ichi Anjyo, and Kiyoshi Arai. Tour into the picture: using a spidery mesh interface to make animation from a single image. *ACM SIGGRAPH*, pages 225–232, 1997.

[25] Natasha Kholgade, Tomas Simon, Alexei Efros, and Yaser Sheikh. 3d object manipulation in a single photograph using stock 3d models. *ACM SIGGRAPH*, 33(4):127:1–127:12, 2014.

[26] Kevin Karsch, Kalyan Sunkavalli, Sunil Hadap, Nathan Carr, Hailin Jin, Rafael Fonte, Michael Sittig, and David Forsyth. Automatic scene inference for 3d object compositing. *ACM SIGGRAPH*, 33(3):32:1–32:15, 2014.

[27] Can Erdogan, Manohar Paluri, and Frank Dellaert. Planar segmentation of rgbd images using fast linear fitting and markov chain monte carlo. In *Proceedings of the 2012 Ninth Conference on Computer and Robot Vision*, CRV '12, pages 32–39, 2012.

[28] Si Lu, Xiaofeng Ren, and Feng Liu. Depth enhancement via low-rank matrix completion. *IEEE CVPR*, pages 3390–3397, 2014.

[29] Tianjia Shao, Aron Monszpart, Youyi Zheng, Bongjin Koo, Weiwei Xu, Kun Zhou, and Niloy Mitra. Imagining the unseen: Stability-based cuboid arrangements for scene understanding. *ACM SIGGRAPH Asia*, 2014. * Joint first authors.

[30] Ke Colin Zheng, Alex Colburn, Aseem Agarwala, Maneesh Agrawala, David Salesin, Brian Curless, and Michael F. Cohen. Parallax photography: creating 3d cinematic effects from stills. *Proceedings of Graphics Interface 2009*, pages 111–118, 2009.

[31] 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.

[32] Takeo Igarashi and John F. Hughes. A Suggestive Interface for 3D Drawing. In *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology*, UIST '01, pages 173–181. ACM, 2001.

[33] Seok-Hyung Bae, Ravin Balakrishnan, and Karan Singh. ILoveSketch: As-natural-as-possible Sketching System for Creating 3D Curve Models. In *Proceedings of UIST*, UIST '08, pages 151–160, 2008.

[34] Ryan Schmidt, Azam Khan, Karan Singh, and Gord Kurtenbach. Analytic Drawing of 3D Scaffolds. In *ACM SIGGRAPH Asia 2009 Papers*, SIGGRAPH Asia '09, pages 149:1–149:10. ACM, 2009.

[35] Jun Xie, Aaron Hertzmann, Wilmot Li, and Holger Winnemöller. PortraitSketch: Face Sketching Assistance for Novices. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*. ACM, 2014.

[36] Luca Benedetti, Holger Winnemöller, Massimiliano Corsini, and Roberto Scopigno. Painting with Bob: Assisted Creativity for Novices. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, pages 419–428. ACM, 2014.

[37] Piyum Fernando, Roshan Lalintha Peiris, and Suranga Nanayakkara. I-draw: Towards a freehand drawing assistant. OzCHI '14, pages 208–211, 2014.

[38] Daniel Dixon, Manoj Prasad, and Tracy Hammond. iCanDraw: Using Sketch Recognition and Corrective Feedback to Assist a User in Drawing Human Faces. pages 897–906, 2010.

[39] Yong Jae Lee, C. Lawrence Zitnick, and Michael F. Cohen. Shadowdraw: Real-time user guidance for freehand drawing. *ACM Trans. Graph.*, 30(4):27:1–27:10, July 2011.

[40] Yotam Gingold, Etienne Vouga, Eitan Grinspun, and Haym Hirsh. Diamonds from the rough: Improving drawing, painting, and singing via crowdsourcing. In *Proceedings of the AAAI Workshop on Human Computation (HCOMP)*, 2012.

[41] Alex Limpaecher, Nicolas Feltman, Adrien Treuille, and Michael Cohen. Real-time drawing assistance through crowdsourcing. *ACM Trans. Graph.*, 32(4):54:1–54:8, July 2013.

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

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

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

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

[46] Hao Pan, Yang Liu, Alla Sheffer, Nicholas Vining, Changjian Li, and Wenping Wang. Flow aligned surfacing of curve networks. *ACM Trans. Graph. (SIGGRAPH)*, 34(4), 2015.

[47] Jennifer Fernquist, Tovi Grossman, and George Fitzmaurice. Sketch-sketch revolution: An engaging tutorial system for guided sketching and application learning. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 373–382. ACM, 2011.

[48] Saeko Takagi, Noriyuki Matsuda, Masato Soga, Hirokazu Taki, Takashi Shima, and Fujiichi Yoshimoto. A learning support system for beginners in pencil drawing. In *Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 281–282. ACM, 2003.

[49] D. Cummmings, F. Vides, and T. Hammond. I don't believe my eyes!: Geometric sketch recognition for a computer art tutorial. In *Proceedings of the International Symposium on Sketch-Based Interfaces and Modeling*, SBIM '12, pages 97–106, 2012.

[50] Floraine Grabler, Maneesh Agrawala, Wilmot Li, Mira Dontcheva, and Takeo Igarashi. Generating photo manipulation tutorials by demonstration.

In *ACM SIGGRAPH 2009 Papers*, SIGGRAPH '09, pages 66:1–66:9. ACM, 2009.

[51] John Tchalenko. Eye movements in drawing simple lines. *PERCEPTION*, 36(8):1152, 2007.

[52] John Tchalenko. Segmentation and accuracy in copying and drawing: Experts and beginners. *Vision Research*, 49(8):791 – 800, 2009.

[53] Ryan Schmidt, Azam Khan, Gord Kurtenbach, and Karan Singh. On Expert Performance in 3D Curve-drawing Tasks. In *Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling*, SBIM '09, pages 133–140, New York, NY, USA, 2009. ACM.

[54] Cindy Grimm. Results of an observational study on sketching results of an observational study on sketching. *SBIM*, 2011.

[55] Jan Eric Kyprianidis, John Collomosse, Tinghuai Wang, and Tobias Isenberg. State of the art: A taxonomy of artistic stylization techniques for images and video. *IEEE Transactions on Visualization and Computer Graphics*, 19(5):866–885, 2013.

[56] Doug DeCarlo, Adam Finkelstein, Szymon Rusinkiewicz, and Anthony Santella. Suggestive contours for conveying shape. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 22(3):848–855, July 2003.

[57] Doug DeCarlo, Adam Finkelstein, and Szymon Rusinkiewicz. Interactive rendering of suggestive contours with temporal coherence. In *NPAR*, pages 15–24, June 2004.

[58] Michael Burns, Janek Klawe, Szymon Rusinkiewicz, Adam Finkelstein, and Doug DeCarlo. Line drawings from volume data. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 24(3):512–518, August 2005.

[59] Doug DeCarlo and Szymon Rusinkiewicz. Highlight lines for conveying shape. In *International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, August 2007.

[60] Hongbo Fu, Shizhe Zhjou, Ligang Liu, and Niloy Mitra. Animated construction of line drawings. 30(6), 2011.

[61] Jingbo Liu, Hongbo Fu, and Chiew-Lan Tai. Dynamic sketching: Simulating the process of observational drawing. In *Proceedings of the Workshop on Computational Aesthetics*, CAe '14, pages 15–22, 2014.

[62] Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. Image analogies. *ACM SIGGRAPH*, pages 327–340, August 2001.

[63] Xiaobo An, Xin Tong, Jonathan D. Denning, and Fabio Pellacini. Appwarp: Retargeting measured materials by appearance-space warping. In *ACM SIGGRAPH Asia*, 2011.

[64] Fabio Pellacini, Frank Battaglia, Keith Morley, and Adam Finkelstein. Lighting with paint. *ACM TOG*, 26(2):Article 9, June 2007.

[65] Ming-Ming Cheng, Fang-Lue Zhang, Niloy J. Mitra, Xiaolei Huang, and Shi-Min Hu. Repfinder: Finding approximately repeated scene elements for image editing. *ACM TOG*, 29(4):83:1–83:8, 2010.

[66] Ming-Ming Cheng, Shuai Zheng, Wen-Yan Lin, Vibhav Vineet, Paul Sturgess, Nigel Crook, Niloy J. Mitra, and Philip Torr. Imagespirit: Verbal guided image parsing. *ACM TOG*, 34(1):3:1–3:11, December 2014.

[67] Thorsten-Walther Schmidt, Fabio Pellacini, Derek Nowrouzezahrai, Wojciech Jarosz, and Carsten Dachsbacher. State of the art in artistic editing of appearance, lighting and material. *CGF*, 35(1):216–233, 2016.

[68] Nuke. `https://www.thefoundry.co.uk/products/nuke/`.

[69] Erik Reinhard, Michael Ashikhmin, Bruce Gooch, and Peter Shirley. Color transfer between images. *IEEE Comput. Graph. Appl.*, September 2001 2001.

[70] Anat Levin, Dani Lischinski, and Yair Weiss. Colorization using optimization. In *SIGGRAPH*, SIGGRAPH '04, pages 689–694, New York, NY, USA, 2004. ACM.

[71] Soheil Darabi, Eli Shechtman, Connelly Barnes, Dan B Goldman, and Pradeep Sen. Image Melding: Combining inconsistent images using patch-based synthesis. *ACM SIGGRAPH*, 31(4):82:1–82:10, 2012.

[72] Olga Diamanti, Connelly Barnes, Sylvain Paris, Eli Shechtman, and Olga Sorkine-Hornung. Synthesis of complex image appearance from limited exemplars. *ACM TOG*, 34(2), 2015.

[73] Ce Liu, Jenny Yuen, and Antonio Torralba. Sift flow: Dense correspondence across scenes and its applications. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(5):978–994, May 2011.

[74] S.W. Hasinoff, M. Jozwiak, F. Durand, and W.T. Freeman. Search-and-replace editing for personal photo collections. In *IEEE International Conference on Computational Photography*, pages 1–8, 2010.

[75] Yoav HaCohen, Eli Shechtman, Dan B Goldman, and Dani Lischinski. Non-rigid dense correspondence with applications for image enhancement. *ACM SIGGRAPH*, 30(4):70:1–70:9, 2011.

[76] Yoav HaCohen, Eli Shechtman, Dan B Goldman, and Dani Lischinski. Optimizing color consistency in photo collections. *ACM SIGGRAPH*, 32(4):85:1 – 85:9, 2013.

[77] Kaan Yücer, Alec Jacobson, Alexander Hornung, and Olga Sorkine. Transfusive image manipulation. *ACM SIGGRAPH Asia*, 31(6):176–176, November 2012.

[78] Kaan Yücer, Alexander Sorkine-Hornung, and Olga Sorkine-Hornung. Transfusive Weights for Content-Aware Image Manipulation. CGF Eurographics, 2013.

[79] Floraine Berthouzoz, Wilmot Li, Mira Dontcheva, and Maneesh Agrawala. A framework for content-adaptive photo manipulation macros: Application to face, landscape, and global manipulations. *ACM TOG*, 30(5), October 2011.

[80] Shi-Min Hu, Kun Xu, Li-Qian Ma, Bin Liu, Bi-Ye Jiang, and Jue Wang. Inverse image editing: Recovering a semantic editing history from a before-and-after image pair. *ACM TOG*, 32(6), November 2013.

[81] Connelly Barnes, Fang-Lue Zhang, Liming Lou, Xian Wu, and Shi-Min Hu. Patchtable: Efficient patch queries for large datasets and applications. *ACM SIGGRAPH*, 34(4), August 2015.

[82] F. L. Zhang, J. Wang, E. Shechtman, Z. Y. Zhou, J. X. Shi, and S. M. Hu. Plenopatch: Patch-based plenoptic image manipulation. *IEEE TVCG*, 23(5), 2016.

[83] Connelly Barnes and Fang-Lue Zhang. A survey of the state-of-the-art in patch-based synthesis. *Computational Visual Media*, 3(1), 2017.

[84] Shi-Min Hu, Fang-Lue Zhang, Miao Wang, Ralph R. Martin, and Jue Wang. Patchnet: A patch-based image representation for interactive library-driven image editing. *ACM TOG*, 32(6), November 2013.

[85] 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*, 35(4), 2016.

[86] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. *IEEE CVPR*, pages 2414–2423, June 2016.

[87] Paul Guerrero, Stefan Jeschke, Michael Wimmer, and Peter Wonka. Edit propagation using geometric relationship functions. *ACM SIGGRAPH*, 33(2):15:1–15:15, April 2014.

[88] Jun Xing, Hsiang-Ting Chen, and Li-Yi Wei. Autocomplete painting repetitions. *ACM SIGGRAPH Asia*, 33(6):172:1–172:11, November 2014.

[89] Gilbert Louis Bernstein and Wilmot Li. Lillicon: Using transient widgets to create scale variations of icons. *ACM SIGGRAPH*, 34(4):144:1–144:11, July 2015.

[90] Paul Guerrero, Gilbert Bernstein, Wilmot Li, and Niloy J. Mitra. PATEX: Exploring pattern variations. *ACM SIGGRAPH 2016*, 2016.

[91] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Susstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE PAMI*, 34(11):2274–2282, 2012.

[92] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE PAMI*, 26:359–374, 2001.

[93] James McCrae, Karan Singh, and Niloy J. Mitra. Slices: A shape-proxy based on planar sections. *ACM SIGGRAPH Asia*, 30(6):168:1–168:12, 2011.

[94] Jia-Bin Huang, Sing Bing Kang, Narendra Ahuja, and Johannes Kopf. Image completion using planar structure guidance. *ACM SIGGRAPH*, 33(4):129:1–129:10, 2014.

[95] B. Edwards. *The New Drawing on the Right Side of the Brain*. Jeremy P. Tarcher/Putnam, 1999.

[96] K. Eissen and R. Steur. *Sketching: Drawing Techniques for Product Designers*. Bis B.V., Uitgeverij(BIS Publishers), 2007.

[97] K. Eissen and R. Steur. *Sketching: The Basics*. BIS, 2011.

[98] Sketch-a-day. `http://www.sketch-a-day.com/`, 2016. Accessed: 2016-10-23.

[99] Draw-a-box. `http://drawabox.com/lesson/6`, 2016. Accessed: 2016-10-23.

[100] Ravish Mehra, Qingnan Zhou, Jeremy Long, Alla Sheffer, Amy Gooch, and Niloy J. Mitra. Abstraction of man-made shapes. *ACM SIGGRAPH Asia*, 28(5):#137, 1–10, 2009.

[101] Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2015.

[102] FastCompany. 75% of ikea's catalog is computer generated imagery. `https://www.fastcodesign.com/3034975/75-of-ikeas-catalog-is-computer-generated-imagery`, 2014.

[103] CGalter. Multipass compositing in photoshop - vray render elements. `https://www.youtube.com/watch?v=q0WpIzi8sc4`, 2015.

[104] 3DArtist. Compositing 3d render passes in photoshop. `https://www.youtube.com/watch?v=wrsvFKNgxgU`, 2016.

[105] Wenzel Jakob. Mitsuba renderer, 2010. http://www.mitsuba-renderer.org.

[106] Stephen Robert Marschner. Inverse rendering for computer graphics. Technical report, 1998.

[107] Ravi Ramamoorthi and Pat Hanrahan. A signal-processing framework for inverse rendering. ACM SIGGRAPH, pages 117–128, 2001.

[108] Yonatan Wexler, Eli Shechtman, and Michal Irani. Space-time completion of video. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(3):463–476, March 2007.

[109] L. Itti and C. Koch. Computational modelling of visual attention. *Nature Reviews Neuroscience*, 2(3):194–203, 2001.

[110] Morgan McGuire. Computer graphics archive, August 2011. `http://graphics.cs.williams.edu/data`.

[111] Ralph Allan Bradley and Milton E. Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39((3/4)), 1952.

[112] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *CVPR*, 2017.

[113] Johanna Delanoy, Adrien Bousseau, Mathieu Aubry, Phillip Isola, and Alexei A. Efros. What you sketch is what you get: 3d sketching using multi-view deep volumetric prediction. *CoRR*, 2017.

[114] Leon A. Gatys, Alexander S. Ecker, Matthias Bethge, Aaron Hertzmann, and Eli Shechtman. Controlling perceptual factors in neural style transfer. 2016.