A DEPTH PERCEPTION AWARE PEN-BASED 3D SKETCHING SYSTEM

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Cansın Yıldız

June, 2012

ABSTRACT

A DEPTH PERCEPTION AWARE PEN-BASED 3D SKETCHING SYSTEM

Cansın Yıldız
M.S. in Computer Engineering
Supervisor: Assist. Prof. Dr. Tolga Çapın
June, 2012

This thesis proposes a method that resembles a natural pen and paper interface to create curve based 3D sketches. The system is particularly useful for representing initial 3D design ideas without much effort. Users interact with the system by the help of a pressure sensitive *pen* tablet, and a camera. The input strokes of the users are projected onto a drawing plane, which serves as a *paper* that they can place anywhere in the 3D scene. The resulting 3D sketch is visualized emphasizing depth perception by implementing several monocular depth cues, including motion parallax performed by tracking user's head position. Our evaluation involving several naive users suggest that the system is suitable for a broad range of users to easily express their ideas in 3D. We further analyze the system with the help of an architect to demonstrate the expressive capabilities of the system that a professional can benefit.

Keywords: Human Computer Interaction, Sketch Based Modeling, Sketching, Depth Perception, Depth Cues, Face Tracking, Pen Tablet.

ÖZET

DERİNLİK ALGISI VURGUSU İÇEREN, GRAFİK TABLET TABANLI 3 BOYUTLU ÇİZİM SİSTEMİ

Cansın Yıldız
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Yrd. Doç. Dr. Tolga Çapın
Haziran, 2012

Bu tez normal kağıt ve kalem kullanıyormuşcasına kavisli şekiller çizmeyi sağlayan bir yöntem öne sürmektedir. Sistem özellikle akla gelen 3 boyutlu fikirleri zaman kaybetmeden dijital ortama aktarabilmek için kullanışlıdır. Kullanıcılar, basınca duyarlı grafik-tablet ve kamera yardımıyla sistemle etkileşim haline geçerler. Kullanıcıların tablet yüzeyine dokunuşları bir çizim düzlemine aktarılır ve bu düzlem 3 boyutlu sahnede herhangi bir yere yerleştirilebilir. Sistem tek gözle ilgili derinlik ipuçlarını ve kullanıcının kafa pozisyonunundan elde ettiği hareket paralaksını uygulayarak 3 boyutlu çizime derinlik anlamı katar. Sistemin kullanışlılığı üzerine çizim tecrübesi olmayan kullanıcılarla yaptığımız testler, bu sistemin geniş bir kitlenin 3 boyutlu çizimler yapabilmesi için uygun olduğunu göstermektedir. Ayrıca profesyonel bir kişinin sistemin daha anlamlı ve etkili özelliklerinden nasıl yararlanabileceğini göstermek için bir mimarın katılımıyla daha ileri seviyede bir inceleme de yaptık.

Anahtar sözcükler: İnsan Bilgisayar Etkileşimi, Çizim Tabanlı Modelleme, Çizim, Derinlik Algısı, Derinlik İpuçları, Yüz İzleme, Grafik Tablet.

Acknowledgement

This thesis would not be possible without the guidance and the help of several individuals who in one way or another contributed in the preparation and completion of this study.

First and foremost, I have been indebted in the preparation of this thesis to my supervisor, Assist. Prof. Dr. Tolga Çapın of Bilkent University, whose patience and kindness, as well as his academic experience, have been invaluable to me. I also want to express my gratitude to Prof. Dr. Bülent Özgüç and Assoc. Prof. Dr. Veysi İşler for showing keen interest to the subject matter.

I am extremely grateful to Mr. Abdülkadir Toyran for demonstrating the capabilities of my system by creating the most pleasing 3D sketches. Similarly, Mr. Gökhan Tüysüz helped me a great deal with proof-reading this thesis. My friends and colleagues who participated in the user evaluation of the system have been most helpful as well. The informal support and encouragement of many of them was indispensable.

I especially want to thank my beloved Gülşah for her patience and extreme support throughout all those years. She was always there for me when I was in need.

Most important of all, my parents, Nermin and Emin Yıldız, have been a constant source of emotional and moral support during my postgraduate years, and this thesis would certainly not have existed without them.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

3D modeling starts with rough sketching of ideas. The latest efforts in research on the field have focused on bringing the natural pen and paper interface to 3D modeling world. The complicated and hard-to-learn nature of current *WIMP* (windows, icon, pointer, menu) based 3D modeling tools is the reason for the search of a better interface. Several authors have already recognized the importance of this problem[42].

Computer modeling starts with sketching ideas on a real paper medium by an artist. After that, this trained artist converts his ideas on the paper to a real 3D model manually, often spending more time digitizing the idea to 3D model then coming up with it in the first place. Because of this manual nature, 3D modeling is the biggest bottleneck for production design pipelines. There are several high-end systems to create 3D models, such as Maya[**?**] and SolidWorks[**?**], where a WIMP paradigm is used. This paradigm enforces drop-down menus, dialog boxes to enter parameters, moving control points and so on. Although such a paradigm is appropriate for a system who aims to create detailed and precise 3D models as a final product, it often lacks the flexibility an artist will need at the idea creation phase.

The recent trend in modeling research is to automate the process of converting sketches that represent ideas into 3D models. The techniques that are part of

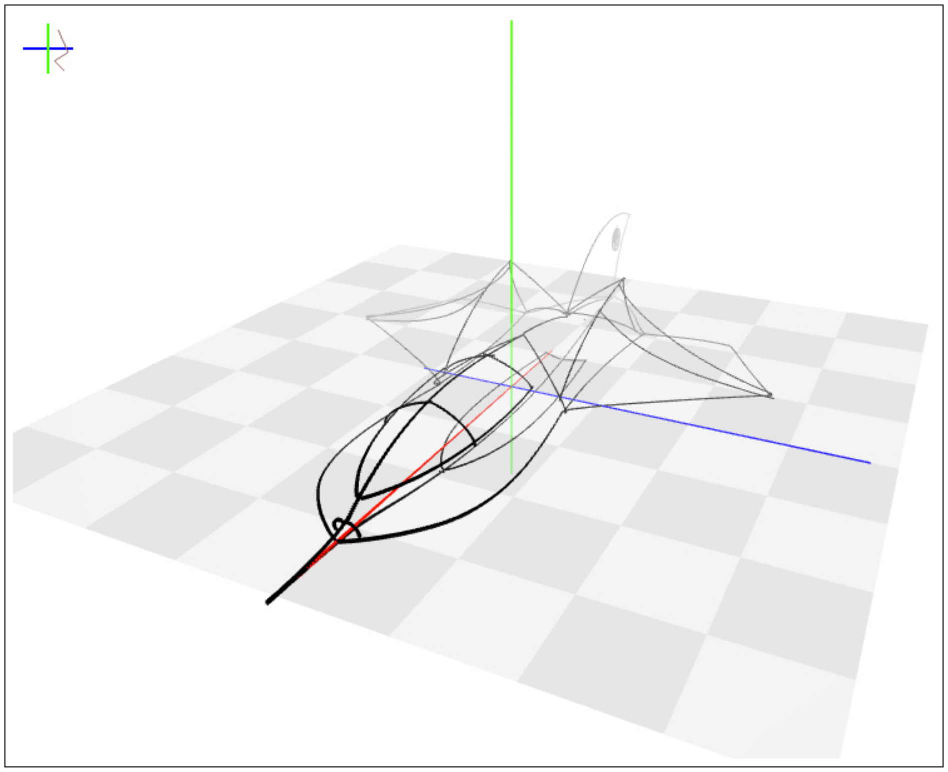


Figure 1.1: A jet fighter created using our system.

this trend are often called Sketch-Based Interfaces for Modeling (*SBIM*). The motivation for an SBIM system is the ease of expressing one's ideas with sketching and the significant flexibility of sketching over traditional WIMP paradigm. How computers will interpret the given sketch and produce a plausible 3D model, is the research question.

In this thesis, we present an SBIM method that tries to mimic the natural interface of pen and paper for creating 3D sketches (Fig. 1.1) that can be used as a starting point for a detailed 3D solid model, or simply an easier way to represent ideas in 3D without much effort. The system is designed to be as minimalistic and simple as possible, since it targets a broad range of users, from expert designers to naive users. There are two main concerns of the system. First is to minimize the learning time needed, yet still be flexible enough to create diverse free form 3D sketches. And second, to emphasize depth perception of the created objects by implementing several monocular depth cues, including *motion parallax* by tracking user's head position. We test whether we are able to achieve these goals, through several user tests (in Chapter 4: Evaluation, Results & Discussion).

Our system is based on the very idea of *curves*, rather than 3D solid objects. Concern of creating surfaces not in mind, it is much easier to develop complicated 3D scenes and objects. Similarly, since the scene consists of only curves, users can easily predict what will be the outcome of drawing a certain stroke. Although there are several other examples of a similar 3D sketching interface, our contributions to the field is to explain an easy to use 3D sketching tool, that is designed with *less is more*[41] thought in mind. A hybrid face tracking algorithm is also developed during the implementation of the system.

The rest of this thesis is organized as follows. Chapter 2 discusses the related work and the underlying motivation for our design decisions. The next chapter gives the details of our system in depth. Chapter 4 explains the user tests we conducted and gives a discussion of the results. Finally, Chapter 5 concludes the thesis by discussing how well we achieved our goals, and what can be done to improve the system further.

Chapter 2

Background & Related Work

Our system has two distinct pipelines to work. These are *Sketching* (Section 3.2) and *Face Tracking* (Section 3.3.2). In order to better understand these components, one should learn the fundamentals of Sketch Based Modeling, Depth Perception, and Face Tracking. The upcoming subsections cover those areas in that order. *Related Work* of the system is detailed in Section 2.1.3.1.

2.1 Sketch Based Interfaces for Modeling

Sketch Based Interfaces for Modeling (SBIM) aims to create an automated (or at least assisted) sketch-to-3D translation[42]. This trend is motivated by the expressiveness and ease of sketching.

The main concern of SBIM is to interpret the given sketch. There are two groups of interpretation an SBIM can make: using the sketch to create a 3D scene; or deforming/manipulating or adding details to an existing 3D model. Regardless of the goal, SBIM applications have a common pipeline (summarized in Fig. 2.1). The first step is sketch acquisition (Section 2.1.1) from user. Then, a filtering process (Section 2.1.2) is performed to clean the input, followed by interpretation of that input (Section 2.1.3) to a 3D operation.

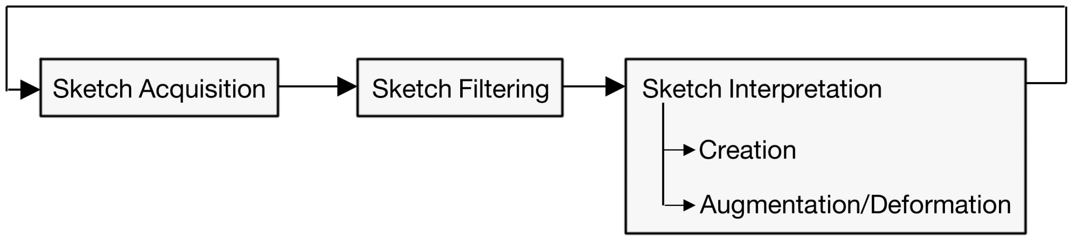


Figure 2.1: The SBIM pipeline: First the input sketch is acquired and filtered. Then, the resulting smoothed input is interpreted as a 3D operation.

2.1.1 Sketch Acquisition

Getting the input sketch from the user is the most basic operation that needs to be performed by all SBIM applications. An input device for an SBIM system should allow freehand input. Even a standard mouse fits that broad description, but in reality devices that try to mimic the real pen and paper feel are much more suitable; such as pen tablets and recently tablet displays (Fig. 2.2(a)).

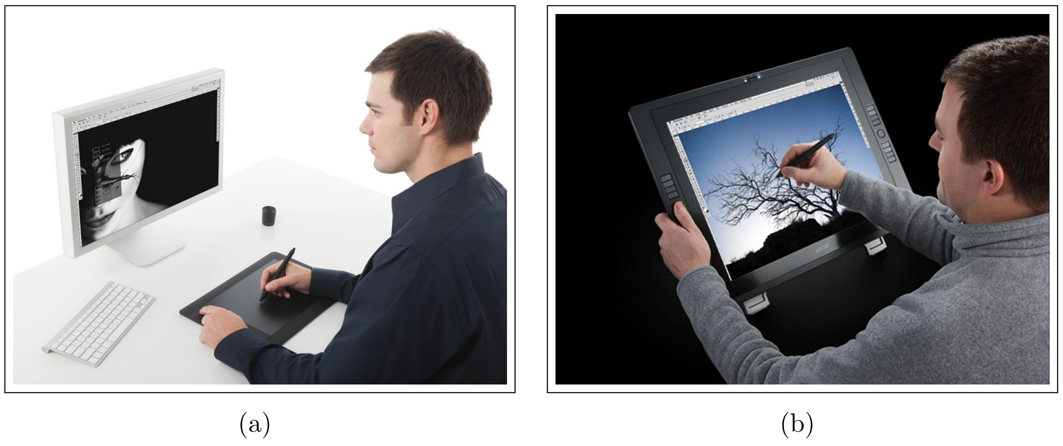

(a) (b)

Figure 2.2: SBIM systems acquire input from pen-based devices such as a pen tablet (a) or tablet display (b). (Wacom Bamboo and Cintiq, respectively.)

The benefit of a tablet display over a pen tablet is that the display is coupled with the input as well, providing even a better natural interaction.

2.1.1.1 Sketch Representation

At bare minimum, a pen tablet provides the positional information in 2D window coordinates. That positional information forms a piecewise linear approximation of the actual continuous gesture. The sample rate varies according to several factors, including but not limited to; drawing speed at a given time, or the input device's model. Therefore, the sample points are not evenly spaced. The points tend to be closer when user moves his hand slower (e.g. corners), and further away when the gesture is faster (e.g. straight lines). This fact can be exploited to detect important parts of a drawing [48, 51].

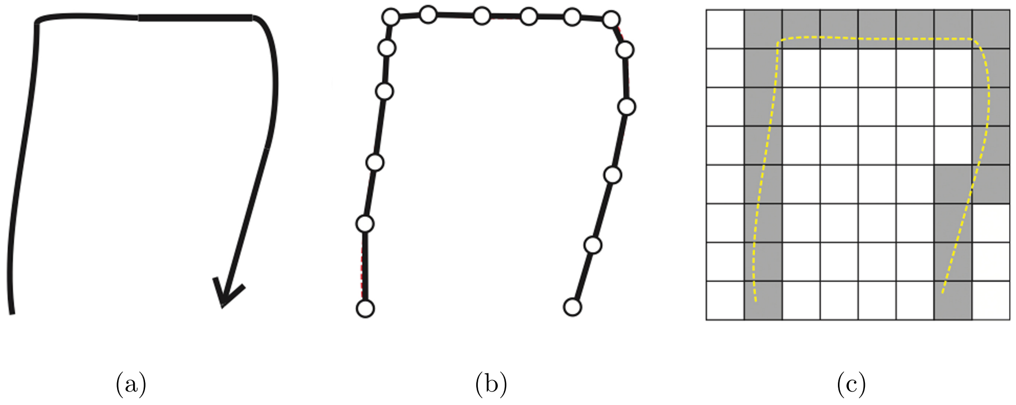


Figure 2.3: (a) A stroke is performed by the user, (b) captures as a sequence of discrete points by pen device; (c) an image-based representation can also be used to represent the input. Reprinted from [42].

A time ordered sequence of input points that begins with a pen-down action and ends with pen-up is called a *stroke* (Fig. 2.3(a)). A *sketch* is the composition of many strokes. At bare minimum, a stroke is represented by a set of points, where every point p contains 2D coordinates (Fig. 2.3(b)). This information can be further enhanced by storing additional information such as the pressure that is applied to the pen at that point of time.

It is also possible to represent strokes with an image-based approach (Fig. 2.3(c)). Image-based stroke representation is mostly preferred by SBIM applications that aim to use the advantage of a fixed memory size and automatic blending of strokes. One disadvantage of this approach is that, the temporal (i.e. time related) nature of the strokes is lost.

To be able to draw strokes to a 3D scene, the notion of a "drawing canvas" is introduced by several SBIM systems [21, 22]. Any *Cartesian* (e.g. x-y plane) or user-specified plane can become a drawing canvas, where the sketch is projected on. It is also possible to use non-planar surfaces as drawing canvas as well.

2.1.2 Sketch Filtering

A sketch needs to be filtered and smoothed before it can be interpreted, since the input is prone to be noisy and erroneous. There are two main sources for such noise: *user* and *device error*[52]. It is possible to end up with curves not that smooth, if the user is not proficient with drawing. Similarly, digitization of the input curve by the mechanical hardware may also induce noise. Therefore, it is essential to filter out noise by means presented below.

2.1.2.1 Resampling and Smoothing

As user's drawing speed changes the distance between sampled points by input device also varies. Resampling that input data to even out distances is a way to reduce the noise (Fig. 2.4(a)). It can be done in real time by inserting new data points between further away sample points (i.e. interpolating), or by eliminating sample points that are too close to each other. Another approach for resampling is to apply a linear or smooth interpolation after the stroke gesture is finished.

Even a resampled input data does not guarantee a smooth curve. Therefore, it is necessary to further process input strokes to reduce discontinuities. A Gaussian filter[53] or a local averaging filter[1] can be applied to each sample point to achieve the desired effect.

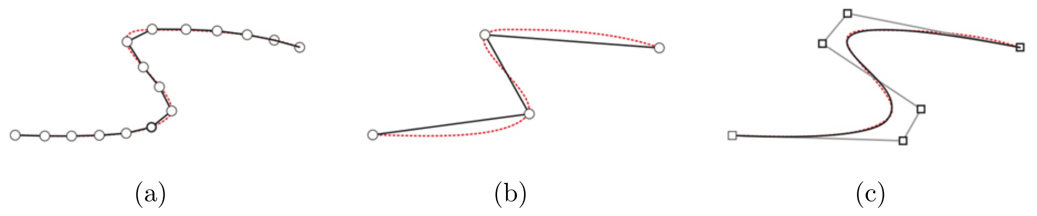


Figure 2.4: Filtering operations: (a) smooth uniform resampling; (b) coarse polyline approximation; (c) fit to a spline curve. Reprinted from [42].

2.1.2.2 Fitting

The number of sample points is still large after resampling and smoothing. It is important to simplify the input by fitting it to another representation. Polyline approximation is the easiest fitting that can be done, but the resulting output is not that representative (Fig. 2.4(b)). Curve fitting is the other option, which is the process of approximating the input point by means of curves, rather than lines (Fig. 2.4(c)). Obviously, curves are more representative.

There are a few different approaches of curve fitting. In general, parametric curve fitting such as Bezier[43], or B-spline[47] are more preferable than least-squares polynomial fitting[42]. One way to achieve a B-spline curve fitting is applying reverse Chaikin subdivision to generate the control points for the b-spline curve[17].

2.1.2.3 Over-sketching

When a user makes a mistake on the sketch, *oversketching* can be performed on the undesired region by carefully re-sketching it. The system is responsible for finding the affected region by over-sketching gesture and replacing it with the new input. The transition between old segments and the new segment should also be smoothed out (Fig. 2.6). It is possible to perform over-sketching in 2D, before the 3D interpretation [**?**].

Another form of over-sketching is where the artist draws an object out of

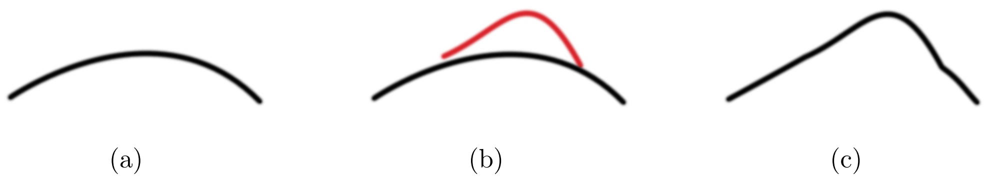


Figure 2.5: Over-sketching: (a) initial curve; (b) oversketch gesture in red; (c) resulting curve.

several small overlapping strokes, rather than full-length smooth curves. Some SBIM systems can operate on such input by automatically blending these small strokes together to form a curve [**?**].

2.1.3 Sketch Interpretation

After sketch acquisition and filtering, the main step of an SBIM system is to interpret the sketch by mapping it to 3D. A freehand sketch input is open to several different interpretations, unlike selecting a menu item from drop-down menu. There are numerous open questions an SBIM system should answer. What is the user's intention? Is the input valid? What is the correct way to map the input to a 3D operation?

There are many different interpretation of these questions, as expected. Olsen et al.[42] propose a taxonomy of SBIM systems to better explain these different interpretations. SBIM systems fall into two main categories. Those that try to fully create a 3D scene out of sketch input are called *Model Creation Systems*. And those that try to augment or deform an already existing 3D object with the guidance of the given input sketch can be grouped under *Augmentation and Deformation*.

As explained, a model creation system aims to construct a 3D model or a scene from the 2D sketch input given by user. There are two distinct groups of model creation systems; *Evocative Systems* and *Constructive Systems* (Fig. 2.6). A constructive system tries to directly create the model out of the input strokes, whereas an evocative system merely uses these input strokes to come up with a

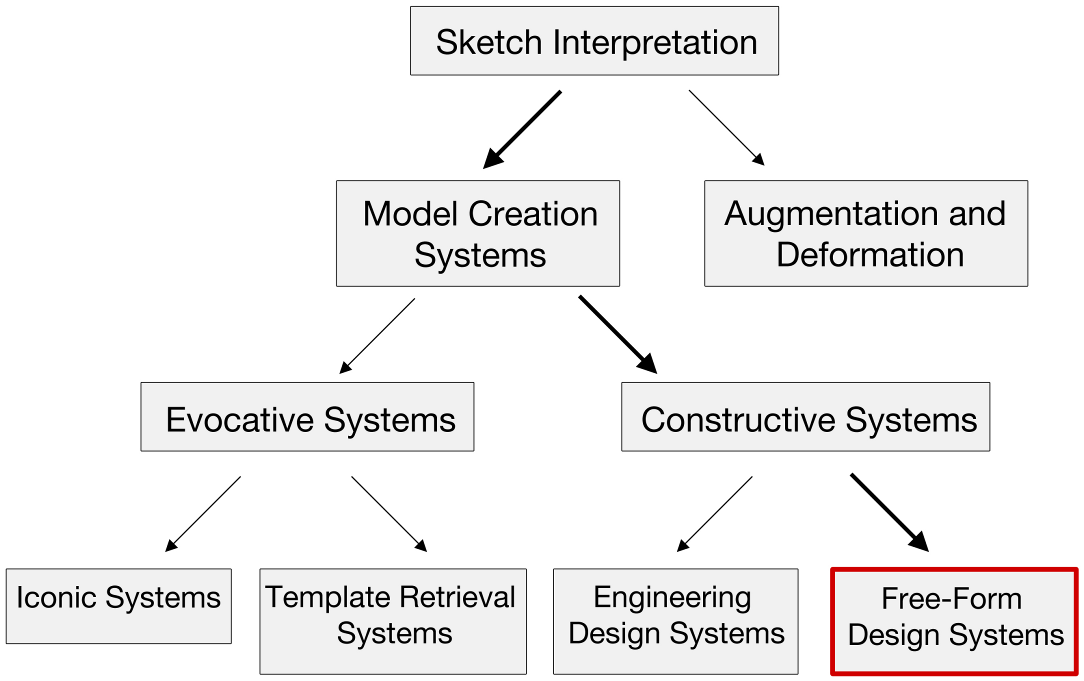


Figure 2.6: A taxonomy of sketch interpretation techniques. Our system fall in *Free-Form Design Systems* under *Constructive Systems* for *Model Creation*

modified version of one of the built-in 3D model types that resembles the input.

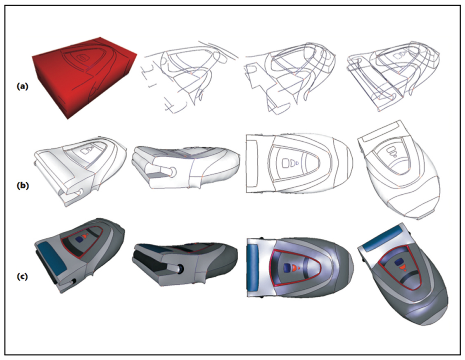
Constructive systems are harder to achieve than evocative systems, since the ambiguity of the input sketch can easily be reduced by the recognition step of an evocative system. On the other hand, the constructive systems need to reconstruct the 3D scene by just depending on the rules generated from input strokes alone. Since reconstruction is such a difficult problem, there are many diverse attempts to solve it. We can group these attempts in two main groups; *Engineering Design Systems* and *Free-Form Design Systems*.

Engineering design systems deal with hard-edged mechanical objects and symmetrical properties. On the other hand, a free-form design system is mostly about (but not limited to) smooth, natural objects that need to be represented by curves, rather than straight lines. Since our system can be categorized as a *Free-Form Design System*, we are going to explain the related work in this specific area in the next subsection.

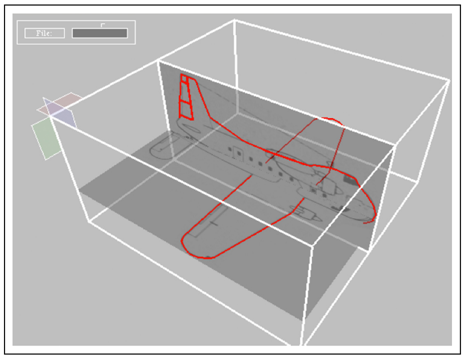
2.1.3.1 Free-Form Design Systems

Creating free-form 3D objects and sketches from 2D user interfaces has been studied for a long time[9, 18, 59, 31]. Baudel et al.[9] describe a sketching interface for creating, manipulating and erasing curves in 2D, one of the oldest efforts. Cohen et al.[18] later propose an idea of creating 3D curves by drawing a 2D silhouette of the curve, and its shadow onto the surface. Unfortunately these invaluable efforts do not try to create a fully developed 3D sketch interface. Igarashi et al.[31] suggest such a true 3D scene creation tool later on.

(a) Bourguignon et al.



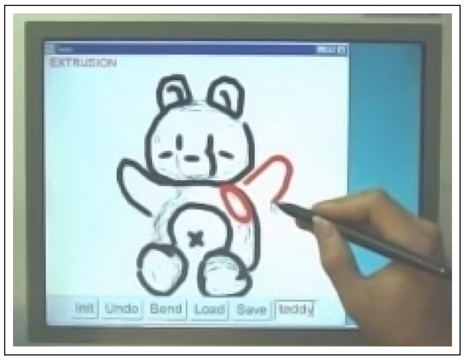
(b) Kara et al.



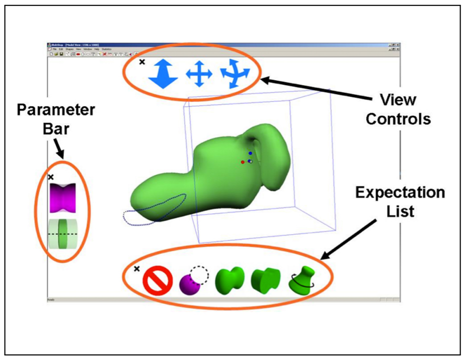
(c) Tsang et al.

Figure 2.7: Related Work: (a) Bourguignon et al. (b) Kara et al. (c) Tsang et al.

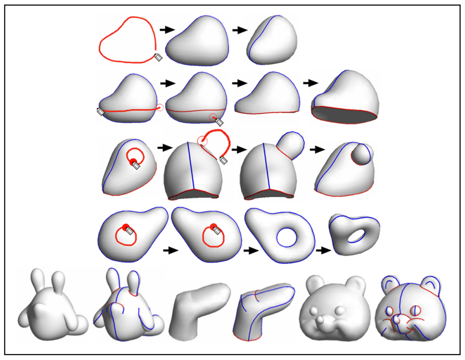
There are recent studies on the subject that aim to enrich an already existing 3D scene, either by annotating the scene or augmenting the 3D object itself[11, 34, 35, 55]. Bourguignon et al.[11] create such a system that can be used for both annotating a 3D object and creating an artistic illustration that can be represented from different viewpoints (Fig. 2.7(a)). Although the resulting scenes are pleasingly beautiful, they are not truly 3D. The system mimics a 3D perspective by manipulating the curves' render mechanism according to the angle they make with the viewport. At Kara et al.[34, 35]'s work, a true 3D object is created by augmenting a simpler pre-loaded 3D template of the target object (Fig. 2.7(b)). The user then can edit this template 3D with a sketch interface. Similarly, Tsang et al.[55] use an image-based template to guide the users (Fig. 2.7(c)).



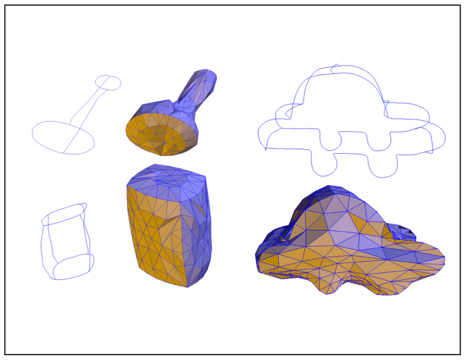
(a) Igarashi et al.s Teddy



(b) Schmidt et al.s ShapeShop



(c) Nealen et al.s FiberMesh



(d) Das et al.

Figure 2.8: Related Work (cont'd): (a) Igarashi et al.'s Teddy (b) Schmidt et al.'s ShapeShop (c) Nealen et al.'s FiberMesh (d) Das et al.

Several other studies focus on creating 3D solid objects using 2D sketches[31, 49, 40, 21]. Igarashi et al.'s *Teddy*[31] is one of the most well known 3D solid object creation systems. In this tool, users are able to create simple 3D objects by drawing 2D silhouettes of the target (Fig. 2.8(a)). Later this silhouette is inflated like a balloon to create the final 3D object. Schmidt et al.'s *ShapeShop*[49] can be described as an extention to Teddy. ShapeShop supports three types of 3D object creation - “blobby” inflation in style of Teddy, linear sweeps and surfaces of revolution (Fig. 2.8(b)). Nealen et al.’s *FiberMesh*[40] is yet another extension to Teddy’s inflation system. However, unlike other systems, the user’s original stroke stays on the model to be used as a control curve for further editing (Fig. 2.8(c)). Finally, Das et al.[21] propose another system, where user strokes are interpreted as 3D space curves rather than 2D silhouettes. Using these curves, a 3D solid object is constructed (Fig. 2.8(d)).

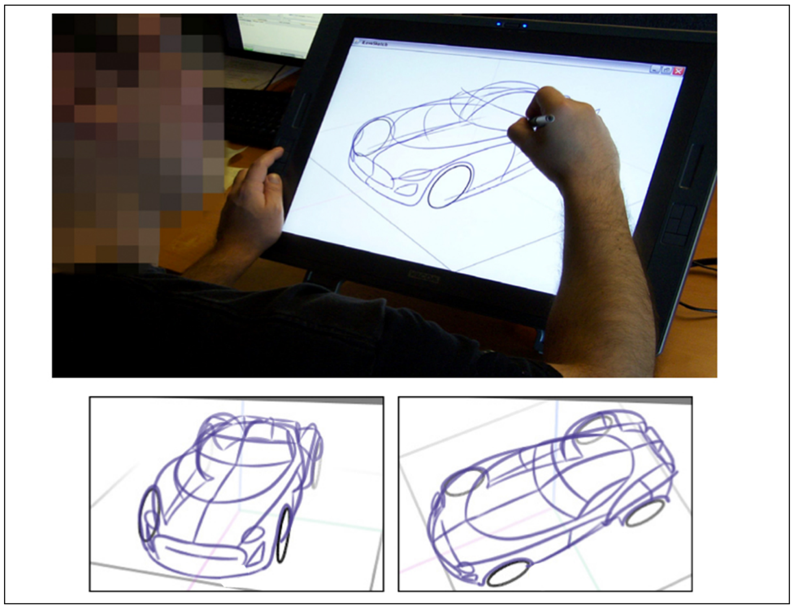


Figure 2.9: Related Work (cont’d): Bae et al.’s ILoveSketch.

The approach we follow for 3D sketching is to ignore *solid* objects, and simply create scenes that only consist of 3D *curves*. The main advantage of such a system over 3D solid object creation is that it is easier to sketch complicated objects with full detail, since there is no concern about creating surfaces. Furthermore, it is easier for users to predict what will be the outcome of any stroke. Bae et al.’s *ILoveSketch*[5], and later extended version *EverybodyLovesSketch*[6], rely on a similar idea and allow users to create 3D sketches consisting of curves with the help of a pen display (Fig. 2.9). Bae et al.’s approach uses several different drawing tools that a user can select from: *ortho plane (span), ortho plane (tick), rotated V plane, oblique plane, extruded surface, freeform surface, 1-view epipolar surface, 2-view epipolar surface.* Similarly, the system has several navigation tools as well: *pan zoom-rotate, dolly-rotate, tumble.* Although it is easy to learn the entry point to 3D sketching ideas such as *orthographic plane sketching* and *single-view symmetric curve*, it takes some time to learn how to use the system in depth[6]. Conversely, in our system we have chosen to use only a single way of drawing and navigating (as explained in Section 3.2.3), which makes it easier to learn.

2.2 Depth Perception

It is important to provide an accurate and informative depth perception during the design of a 3D scene. Depth cues establish the core of depth perception by helping human visual system to perceive the spatial relationships of objects. There are three main categories of depth cues[29]:

1. **Oculomotor:** Cues based on the position of our eyes and the shape of our lens.
2. **Monocular:** Cues that work with one eye.
3. **Binocular:** Cues that depend on two eyes.

2.2.1 Oculomotor Cues

The oculomotor cues consist of *Convergence*, the notion of eyes being converged as we try to look at a nearby object, and *Accommodation*, the change in the tension of eye muscles (hence lens shape) that occurs when the distance of focus changes. The idea is that by feeling these changes at the eyes (the inward convergence and tight muscles), the human visual system can estimate the object distance (Fig. 2.10).

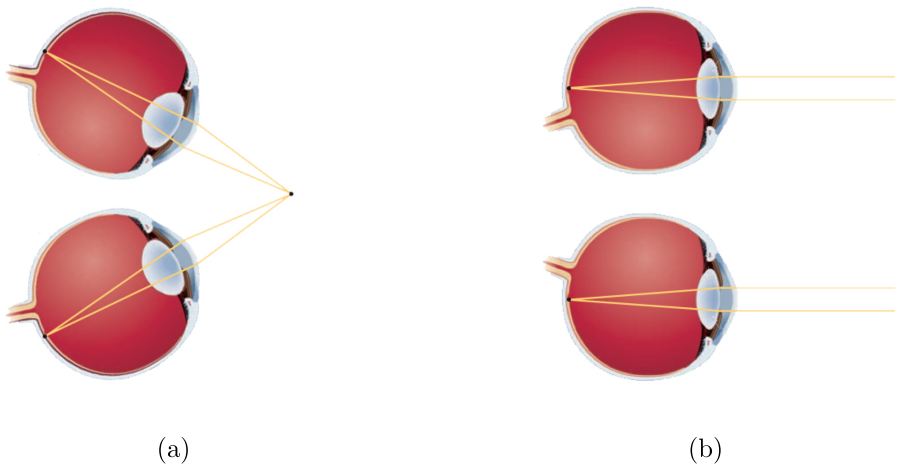


Figure 2.10: Oculomotor Cues: (a) Convergence of the eyes and lens accommodation occurs when a person looks at something that is very close; (b) The eyes look straight ahead and the lens relax when the person observes something that is far away.

2.2.2 Monocular Cues

Monocular cues depend only on a single eye. They can be grouped into *Pictorial Cues*, that can be extracted from a still 2D picture, and *Movement Based Cues*, those related with the movement.

2.2.2.1 Pictorial Cues

Pictorial cues are those that can be perceived even from a static picture (Fig. 2.13). There are a number of different pictorial cues that we list below.

(a) **Occlusion:** Occlusion occurs when a farther away object is fully or partially blocked by another object that is closer to viewpoint (Fig. 2.13(a)).

(b) **Relative Height:** Object that are below the horizon and closer to the viewpoint have their bases lower than those farther away from the viewpoint (Fig. 2.13(b)).

(c) **Relative Size:** According to the cue of relative size, among two equal sized

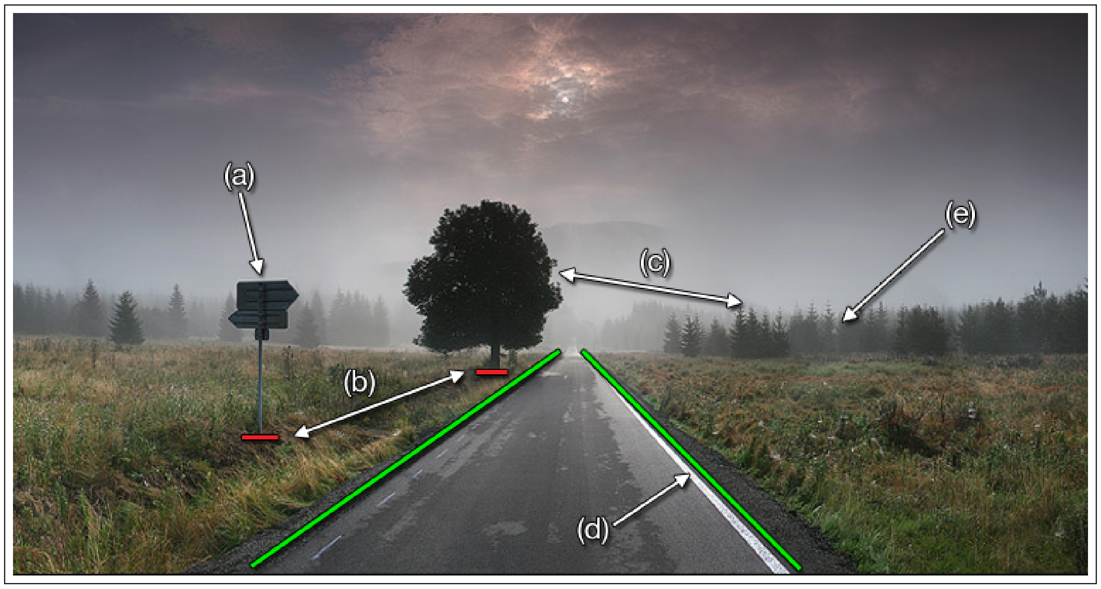


Figure 2.11: Pictorial Cues: (a) occlusion (the road sign occludes the trees behind it); (b) relative height (the tree is higher in the field of view than road sign); (c) relative size (the far trees are smaller than the near one); (d) perspective convergence (the sides of the road converge in the distance); (e) atmospheric perspective (the far trees seem greyed out and less sharp). (Photography courtesy of Robert Mekis)

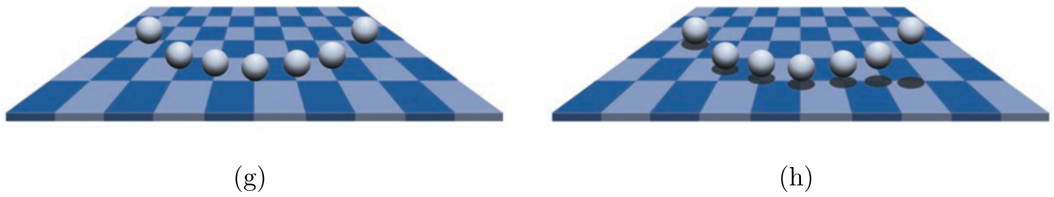


Figure 2.12: Pictorial Cues (cont'd): (g) The location of the spheres are ambiguous; (h) Adding shadows makes their location clear. Notice the texture gradient on the ground as well. (Courtesy of Pascal Mamassion)

objects, the one farther away from the viewpoint will be perceived smaller and take up less of the viewport (Fig. 2.13(c)).

(d) **Perspective Convergence:** Perspective convergence dictates that the parallel lines extend out from the viewpoint will be perceived as converging at distance (Fig. 2.13(d)).

(e) **Atmospheric Perspective:** Atmospheric perspective causes distant objets to be seen more blurry and less saturated with color (Fig. 2.13(e)).

(f) **Familiar Size:** According to this cue, we can judge the distance of objects based on our prior knowledge of their size.

(g) **Shadows:** Objects cast shadows which can provide information about the location of the object (Fig. 2.12).

(h) **Texture Gradient:** As distance increases, the elements get more closely packed with each other, even if they are actually at equal distances with their closer-to-viewpoint relatives (Fig. 2.12).

2.2.2.2 Motion-Produced Cues

Pictorial cues work even if the observer is in a stationary position. Once the observer or the objects start moving around, new cues emerge that further emphasize depth.

(a) **Motion Parallax:** Motion parallax is the phenomenon that as we move, nearby objects move farther across our view volume than distant objects, which moves slower.

(b) **Kinetic Depth Perception:** Even if the observer is stationary, a motion based depth cue is still possible. Kinetic depth perception states that overall shape of an object can be better understand if the object rotates around its axis, eliminating ambiguities of 2D projection.

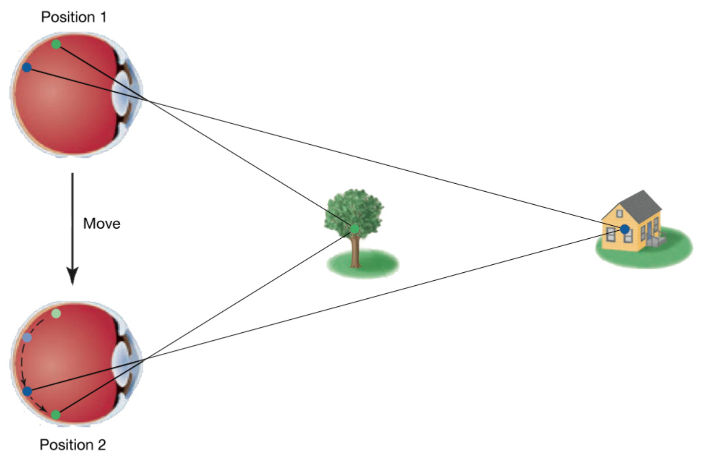


Figure 2.13: Motion Parallax: Notice how the image of the tree moves farther on the retina than the image of the house while eye is moving downwards.

2.2.3 Binocular Cues

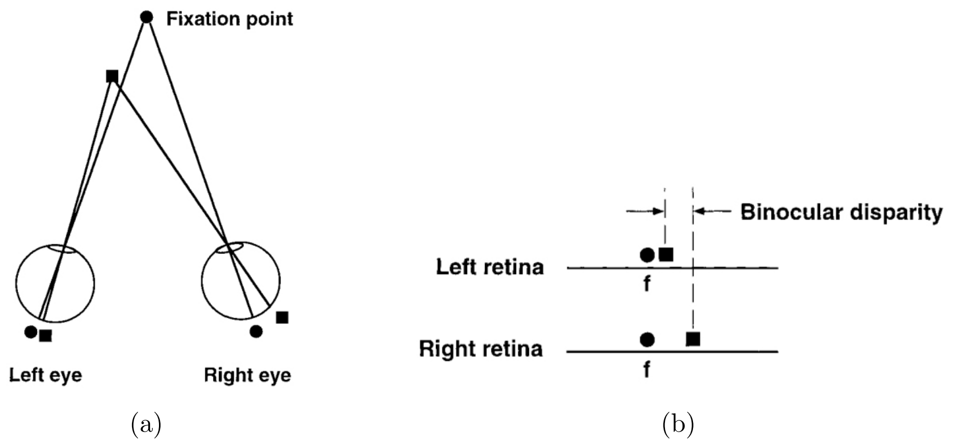


Figure 2.14: Binocular Disparity: (a) Notice the positions of the shapes being observed; (b) Binocular disparity happens between two eye images. Reprinted from [45].

In addition to monocular and oculomotor cues that depend on a single eye, there are also binocular cues, which are created by the differences in the images that is received by left and right eye. *Binocular Disparity* refers to that difference in the location of the object at left and right eye spheres. As object gets closer, disparity increases; and brain uses this information to extract depth information (*Stereopsis*).

2.3 Face Detection and Tracking

As we mention in previous section, motion based cues are fundamental for better perceiving depth information. Such a kinetic depth effect can be achieved by means of tracking the user's face with the help of a camera. The first step that needs to be done for the tracking is Face detection. Face detection is one of the fundamental approaches for a natural human computer interaction. It is the basis algorithm for several facial analysis techniques, including but not limited to facial expression recognition, face verification, face modeling and in our case face tracking.

For a given arbitrary image, the face detection algorithm aims to answer the existence of any faces in the image, and their positions if they do exist[57]. Although the definition of the problem sounds primitive, the solution is not. Therefore, face detection is one of the top studied topics for computer vision. The significance of the problem lies in the several *variable* attributes a face has; including scale, location, pose, expression, lighting and etc. (Fig. 2.15).

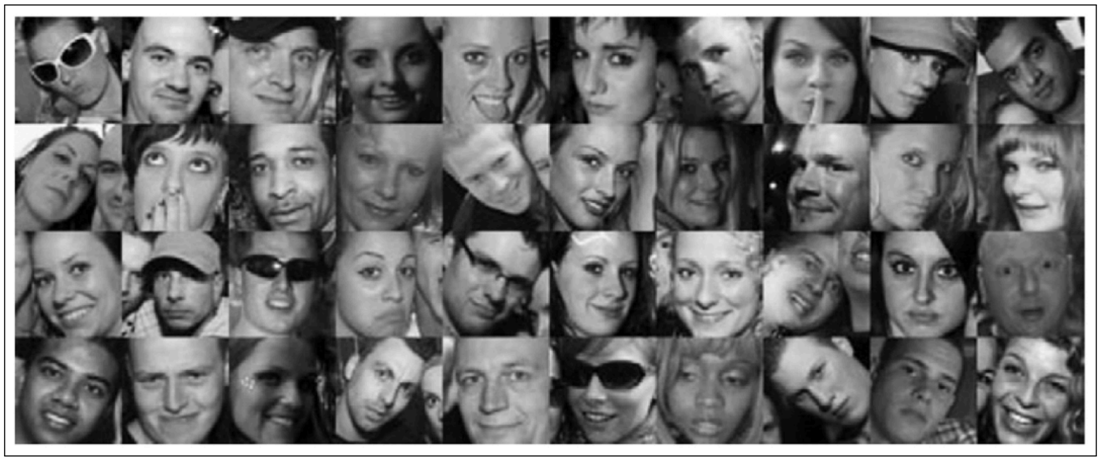


Figure 2.15: Different face poses. Note the variation in pose, facial expression, lighting and etc. Reprinted from [44].

There are several different approaches to face detection. The early approaches (before 2000s) are surveyed in [57] and [30] in detail. Since the face detection is merely a tool for the greater purpose of achieving a depth perception aware sketching tool, there is no need to explain those solutions in depth. Yang et al.[57]

grouped face detection approaches into four main titles: *feature invariant approaches*, *knowledge-based methods*, *template matching methods* and *appearance-based methods*.

Feature invariant approaches aim to match structural face characteristics that are not affected by environmental variables such as pose or lighting. Knowledge-based methods assume predefined rules that are based on several common-sense rules about a human face to detect it. In a template matching approach, there is a face template which is compared against the target image. And finally, appearance based methods use a large dataset of various face images to train a face detector. In general, appearance based methods perform better than all the rest.

The detection algorithm that we use in our system is also an appearance based method. Developed by Viola and Jones[56], the algorithm actually made it practically possible to detect faces in real time. As stated at [60], Viola and Jones' algorithm is the "de-facto standard of face detection".

2.3.1 Viola-Jones Face Detector

The success of Viola-Jones Face Detector depends on three fundamental ideas: *the integral image*, *AdaBoost learning algorithm*, and *the attentional cascade structure*.

2.3.1.1 The Integral Image

Integral image (a.k.a. summed area table) is an algorithm to compute the sum of values in a sub-rectangle of a grid rapidly and efficiently. Crow et al.[20] first proposed the idea of integral image for use in mipmaps. Later, Viola and Jones applied the same principle to quick computation of Haar-like features. The details of the process are below.

The integral image is computed using the equation,

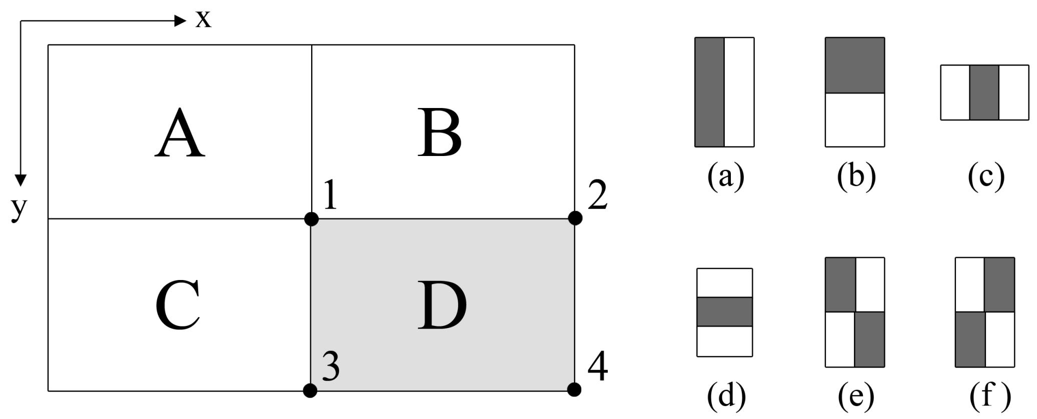


Figure 2.16: Integral image computation and Haar-like rectangle features (a-f). The sum of the pixels within rectangle D computed with four array references. The value of the integral image at *location* 1 is the sum of the pixels in *rectangle* A. The value at location 2 is $A + B$; at location 3, it is $A + C$; and at location 4, it is $A + B + C + D$. Therefore, the sum for rectangle D can be computed as $4 + 1 - (2 + 3)$.

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y') \tag{2.1}$$

where $ii(x, y)$ is the integral image value, and $i(x', y')$ is the original image value for pixel location (x, y). Using the below two recursions,

$$s(x, y) = s(x, y - 1) + i(x, y) \tag{2.2}$$

$$ii(x, y) = ii(x - 1, y) + s(x, y) \tag{2.3}$$

the integral image can be computed in a single pass using dynamic programming. $s(x, y)$ in the equations is the cumulative row sum, where $s(x, -1) = 0$, and $ii(-1, y) = 0$.

The sum of any sub-rectangular area can easily be computed using the integral image (Fig. 2.16). For instance, the sum of pixels in rectangle D is,

$$\sum_{(x,y) \in D} i(x, y) = ii(4) + ii(1) - ii(2) - ii(3) \tag{2.4}$$

which only requires four values that are precomputed and stored in the integral image.

The integral image approach is used to compute Haar-like rectangular features, as shown in Fig. 2.16(a-f). A feature is simply the intensity difference between two separate rectangular regions. For instance, the feature value (a) is computed as the difference between the average pixel value of gray and white rectangles. Since there are two common corners for these two rectangles, only six references are needed to perform the computation. Similarly, features c and d require eight, features e and f require nine array references.

2.3.1.2 AdaBoost Learning Algorithm

There are many ways of learning a classification function, given a training set of positive and negative images and a feature set to derive. Boosting is one of such classification methods. It is the procedure of combining several *weak* classifiers to achieve an accurate hypothesis, hence called *boosting*. For a general introduction on boosting, one can read references [27] and [39]. One of the typical boosting algorithms that is also recognized as the first step of several others is the Adaptive Boosting (*AdaBoost*) algorithm[26]. Viola-Jones uses AdaBoost to select a small set of features and train the classifier.

There are hundreds of thousands of Haar-like rectangular features for each image sub-rectangle, significantly more than the total number of pixels. Although the computation time for a single feature is not that exhaustive, computing the complete set of features is practically not possible. Therefore, the Viola-Jones algorithm aims to compute a fine tuned sub-set of features, which creates an effective classifier once combined. The main challenge is, of course, to find these features.

To achieve this, the single Haar-like feature that best separates positive samples from negatives is selected at the weak learning algorithm. For every feature, the weak learning algorithm adjusts the optimal threshold, such that the minimum number of examples are misclassified. In essence, a weak learner $h_j(x)$

1. Given example images $(x_1, y_1), \ldots, (x_n, y_n)$ where $y_i = 0, 1$ for negative and positive examples respectively.
2. Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively, where m and l are the number of negatives and positives respectively.
3. For $t = 1, \ldots, T$:
   (a) Normalize the weights,
   $$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}} \tag{2.5}$$
   (b) For each feature j, train a classifier h_j which is restricted to using a single feature. The error is evaluated with respect to $w_t, \epsilon_j = \sum_i |h_j(x_i) - y_i|$.
   (c) Choose the classifier h_t, with lowest error ϵ_t.
   (d) Update the weights:
   $$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i} \tag{2.6}$$
   where $e_i = 0$ if example x_i is classified correctly, $e_i = 1$ otherwise, and $\beta_t = \frac{\epsilon t}{1-\epsilon t}$.
4. The final strong classifier is:
$$h(x) = \begin{cases} 1 & \text{if } \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases} \tag{2.7}$$
where $\alpha_t = \log \frac{1}{\beta_t}$

Algorithm 1: The AdaBoost algorithm selects a single feature from the hundreds of thousands of potential features at each iteration. Reprinted from [56].

is,

$$h_j(x) = \begin{cases} 1 & \text{if } p_j f_j(x) < p_j \theta_j \\ 0 & \text{otherwise} \end{cases} \tag{2.8}$$

where f_j is a feature, θ_j is the optimal threshold for that feature, and p_j is the parity to adjust the direction of inequality. See Algorithm 1 for a summary of the complete boosting process.

2.3.1.3 The Attentional Cascade

Attentional cascade plays an important role in the Viola-Jones detector. As stated in [56], *"the key insight is that smaller, and therefore more efficient, boosted classifiers can be constructed which reject many of the negative sub-windows while detecting almost all positive instances"*. In other words, it is possible to adjust the threshold for each boosted classifier so that false negative rate is almost zero. That will lead to the rejection of most sub-windows with false positives in an early stage of the pipeline, making it extremely efficient.

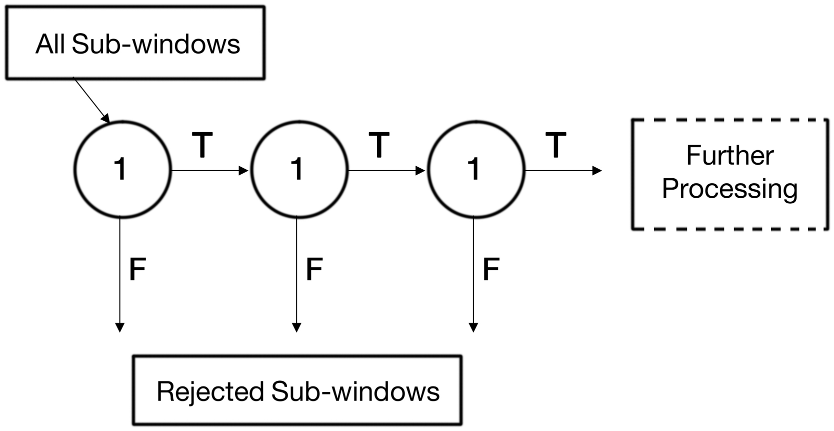


Figure 2.17: Schema of the detection cascade. A pipeline of classifiers are applied to every sub-windows of the image. The classifier get more complex as we proceed at pipeline (i.e. number of weak classifiers that are involved for each node increases for latter nodes). The initial classifier trained to eliminate a massive number of negative examples with very low number of weak classifiers. After several stages of processing the number of candidate sub-windows have been reduced radically.

The process for classification of a sub-window forms a special case of a decision tree, which is referred as a "cascade" in [56]. The input sub-window proceeds through a pipeline of classifier nodes for detection, as shown in Fig. 2.17. Each classifier will make a binary decision on whether a sub-window is a false negative (i.e. reject), or not (i.e. proceed to next node). The number of weak classifiers involved for each node increases as image continues its journey at the pipeline (e.g. the first five nodes include 1, 10, 25, 25, and 50 weak classifiers, respectively [56]). This is an expected schema, since it gets more and more difficult to reject all negative windows while keeping positive samples at later stages. The notion of having fewer number of weak classifiers at first nodes also improves the performance of the Viola-Jones detector.

The training process is also affected by the cascade structure as well. Since a positive sub-window is a rare incidence, usually billions of negative samples are needed to build a successful face detector. Viola and Jones uses a bootstrap process to handle this need. The false positive rate should be reduced at each stage of the attentional cascade. A threshold is *manually* chosen for the minimum reduction in the false positive rate. Then, each stage of the cascade is trained by adding weak classifiers until the threshold of false positive rate is met. More and more stages are added to the final cascade until the overall target for detection rate and false positive rate is achieved.

Viola and Jones[56] trained the attentional cascade *manually* as described above (i.e. the decision thresholds and the number of weak classifiers are selected manually). Given the limited computational power available at that time, it is no surprise that constructing a well performing face detector requires a significant tuning effort..

2.3.2 Continuously Adaptive Mean Shift

The Continuously Adaptive Mean SHIFT (CamShift) algorithm[14], is a derivation of the Mean Shift algorithm[19], a robust non-parametric iterative technique for finding the mode of probability distributions[25].

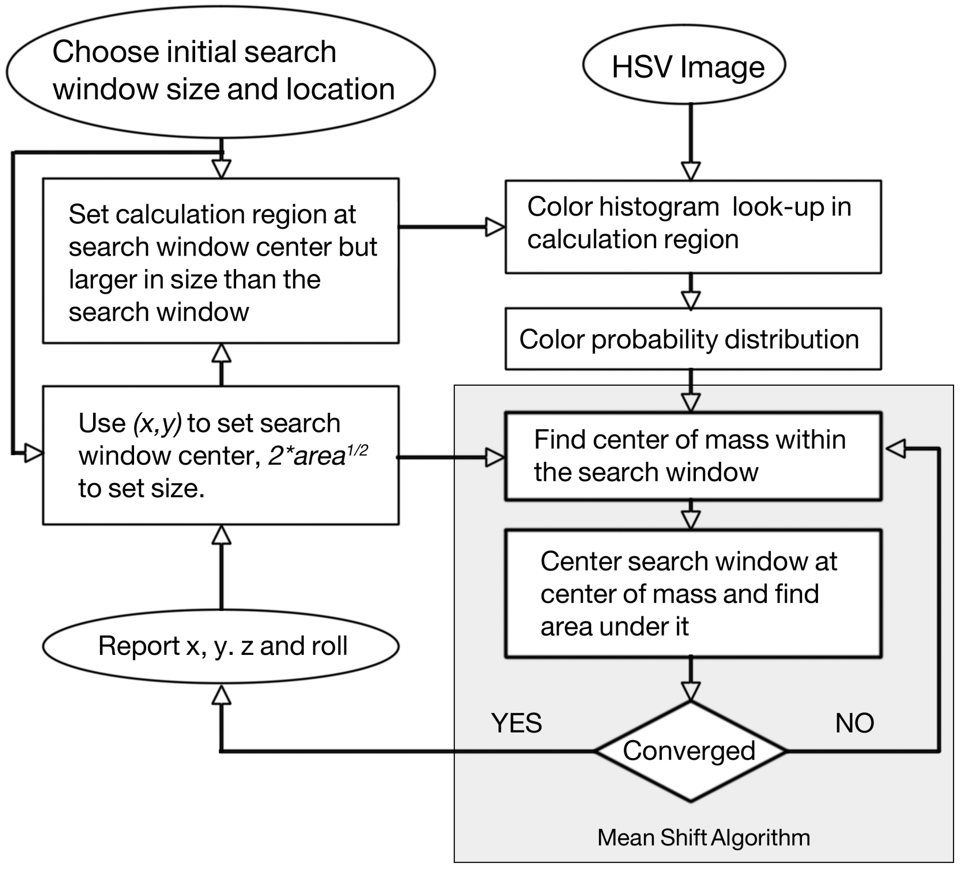

Choose initial search window size and location
HSV Image
Set calculation region at search window center but larger in size than the search window
Color histogram look-up in calculation region
Color probability distribution
Use (x,y) to set search window center, 2*area^1/2 to set size.
Find center of mass within the search window
Center search window at center of mass and find area under it
Report x, y. z and roll
YES
Converged
NO
Mean Shift Algorithm

Figure 2.18: Block diagram of color object tracking.

CamShift algorithm is summarized at Fig. 2.18. At every video frame, a color probability distribution image is created for the actual frame image via a color histogram model of the color being tracked (i.e. skin color for face tracking). CamShift algorithm operates on the probability image to locate the center and the size of the object being tracked. The last size and location is used as the search window for the next video image. That process is repeated for every frame, hence a continuous tracking is performed. As previously noted, the CamShift algorithm is an extension to the Mean Shift algorithm (Fig. 2.18).

2.3.2.1 How the Mean Shift Algorithm Works

The Mean Shift algorithm is the core part of the CamShift algorithm. It is performed on a color probability distribution image that is produced from histogram back-projection:

1. Decide the size of the search window.
2. Decide the initial location for the search window.
3. Compute the location of the mean value in the search window.
4. Adjust the search window's center as the mean location computed in previous step.
5. Repeat Steps 3 and 4 until convergence. (i.e. until the search window's center has moved less than a predefined threshold.)

For a discrete image probability distribution, the location of the mean value in a search window (Steps 3 and 4 above) can be found as follows:

a. Find the zeroth moment,

$$M_{00} = \sum_x \sum_y I(x,y) \tag{2.9}$$

b. Find the first moment for x and y,

$$M_{10} = \sum_x \sum_y xI(x, y); \quad M_{01} = \sum_x \sum_y yI(x, y) \tag{2.10}$$

c. Find the mean search window location as,

$$x_c = \frac{M_{10}}{M_{00}}; \quad y_c = \frac{M_{01}}{M_{00}} \tag{2.11}$$

where $I(x, y)$ is probability at the position (x, y) in the image, and x and y range over the search window.

2.3.2.2 How the Continuously Adaptive Mean Shift Algorithm Works

CamShift is designed for dynamic distributions, unlike the Mean Shift algorithm, which is for static distributions. Change in the distribution occurs when the tracked object moves in a video sequence so that the size and location of the probability distribution changes as well. The goal of the CamShift algorithm is to adjust the search window (both size and location). The initial window size can be set to any reasonable value. In our case, we set the initial location and size of the window using Viola-Jones Face Detector[56]. Using this initial window, CamShift continuously adapts its new window size and location for each video frame, as follows[37]:

1. Set the calculation region of the probability distribution to the whole image.
2. Choose the initial location of the 2D mean shift search window.
3. Calculate the color probability distribution in the 2D region centered at the search window location in an ROI slightly larger than the mean shift window size.
4. Run mean shift algorithm to find the search window center. Store the zeroth moment (area or size) and center location.
5. For the next video frame, center the search window at the mean location stored in Step 4 and set the window size to a function of the zeroth moment found there. Go to Step 3.

For each frame, the Mean Shift algorithm will tend to converge to the mode of the distribution. Therefore, CamShift for video will tend to track the center (mode) of color objects moving in a video scene.

Chapter 3

The System

Users interact with our system using a pressure sensitive pen tablet. In essence, users' pen gestures are captured as time sequenced tablet coordinates and interpreted. The device has several buttons on the tablet that are used for basic non-gestural abilities such as undo, redo, toggle symmetry. The pen also has two buttons, and an eraser at back, that are used as toggles between our gesture modes as detailed in the Section 3.2.3. To be able to give users a real pen and paper experience, our system does not use any of the elements of *WIMP* (windows, icon, pointer, and menu) paradigm, hence creating a natural interface where users complete tasks simply interacting with the system as if it's a real paper.

Depth perception is another aspect that we consider when building the system. As detailed in Section 3.3, we implement several pictorial cues to emphasize the perception of depth throughout the system. We also track the user's head position to manipulate the position and direction of the virtual camera at the scene, creating a motion parallax based kinetic depth effect.

3.1 Overview

Our system consists of three modules: *sketching*, *face tracking*, and *rendering*, as pictured in Fig. 3.1. Every module is responsible for a different aspect of the system.

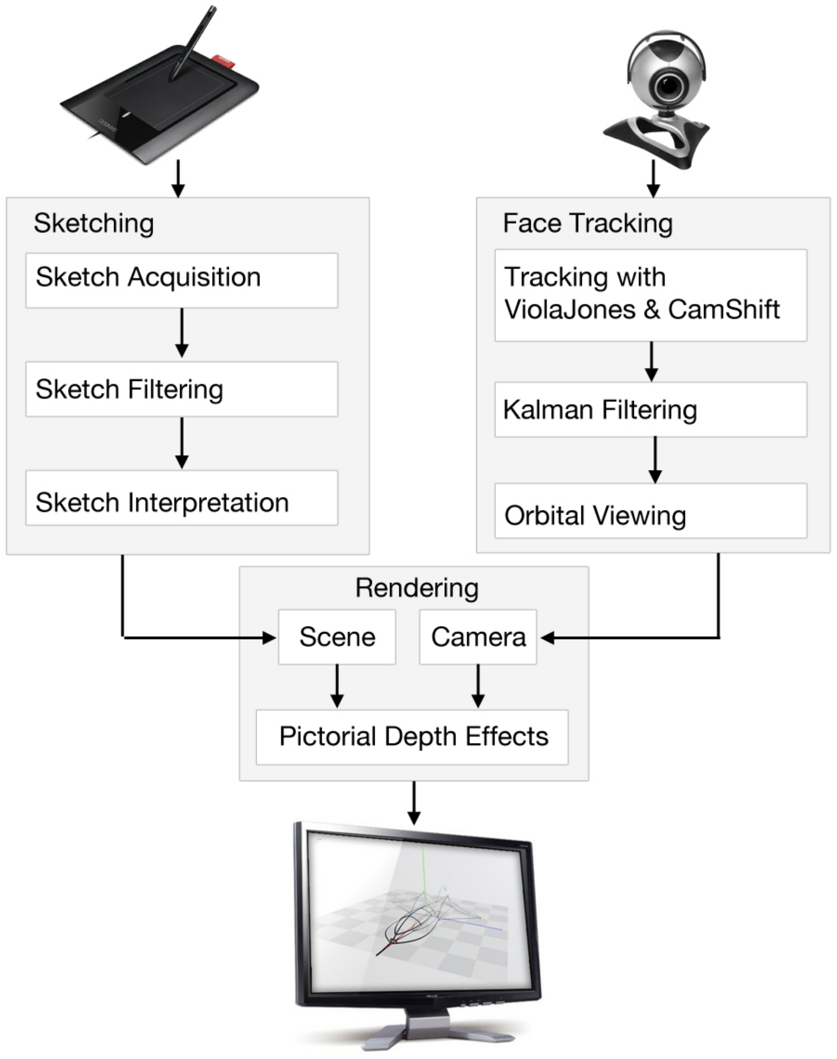


Figure 3.1: Overview of the system. Three modules work together to get sketch input from user, and visualize that sketch emphasizing depth perception with the help of face tracking enabled motion parallax.

The sketching module collects input strokes from user with the help of a pen tablet. The collected input strokes are then filtered and interpreted as explained in Section 3.2. The resulting gestures directly affect scene that is being created. The scene essentially consists of a list of curves that are sketched during a session (Appendix A.3). Meanwhile, the face tracking module fetches video frames from a camera. By applying a hybrid tracking, the face tracking module enables the *virtual camera* to travel orbitally along the scene. Finally, the rendering step uses pictorial depth effects to accurately visualize the current scene for the given viewpoint. Since face tracking and rendering modules are solely responsible for the "visualization" of the system, we are going to discuss the details of these systems in Section 3.3.

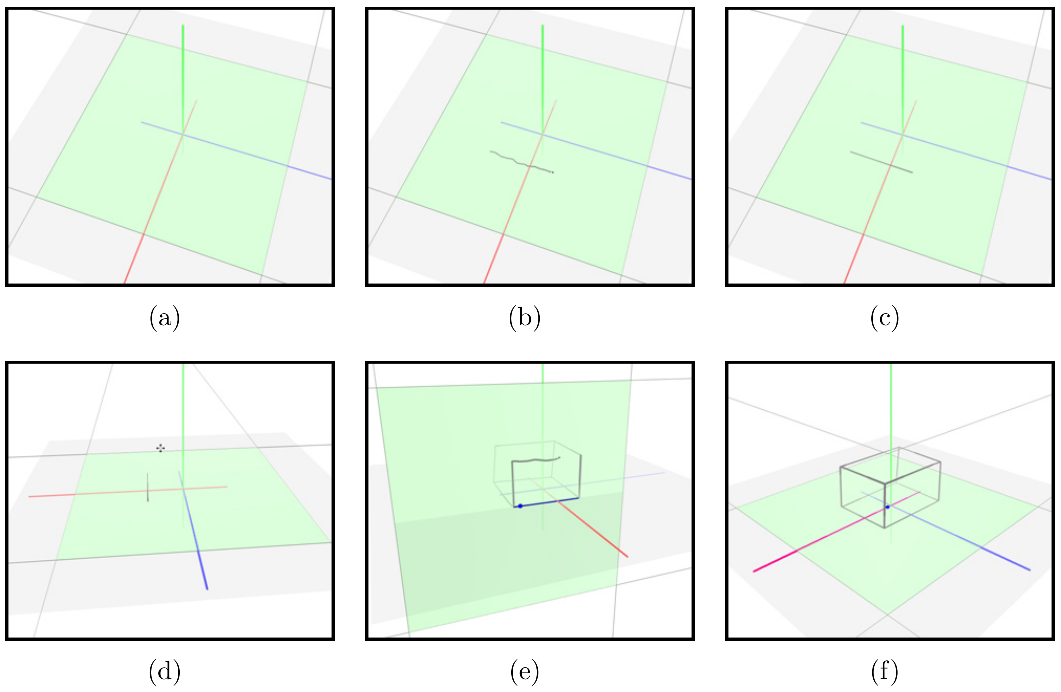


Figure 3.2: Overview of the usage. **(a)** User adjusts the plane that he wants to draw a curve on. **(b)** User draws the curve using pen tablet, which is mapped to the current drawing plane. **(c)** The input curve is then re-sampled and smoothed out. **(d)** User can change the camera position if he needs to. **(e)** This process is repeated until the desired 3D sketch is formed. **(f)** Final result; a cube.

Fig. 3.2 illustrates the overall usage of the system. To be able to draw a curve, the user firsts adjust the *drawing plane* as explained in Section 3.2.3. Any drawing gesture that is made by the user will be reflected on this surface. Once the drawing plane is adjusted, the user can draw a curve with a simple pen gesture on the tablet. The input curve will then be re-sampled and smoothed using the algorithms described at Section 3.2.2. During this process, the user can adjust camera position as well, using the same pen tablet device, if necessary. The user can repeat these steps to complete the 3D object.

3.2 Sketching Pipeline

There is a common pipeline as explained in Section 2.1 for *sketch based interfaces*, which our system also follows. The first step is to acquire input from the user (Section 3.2.1), by means of an input device, a pen tablet in our case. That step is followed by sketch filtering (Section 3.2.2), where the data is re-sampled and smoothed. Finally, the sketch is interpreted appropriately. In our case, interpretation means mapping these curves one of many *gestures* explained in Section 3.2.3.

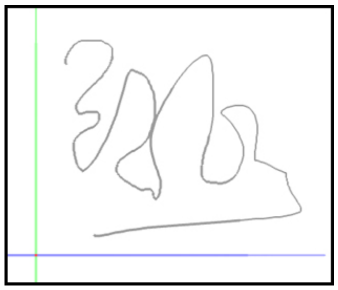
3.2.1 Sketch Acquisition

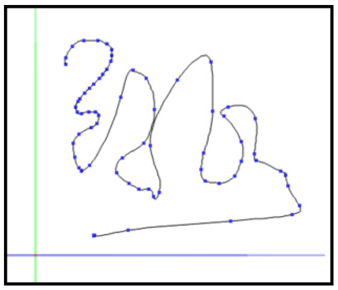
Obtaining a sketch from the user is the first step a sketch based interface should perform. Our system collects free hand sketches from the user using a pen tablet. A tablet display would be even a better choice, since the user will be able to see what he draws just at the drawing surface he is using.

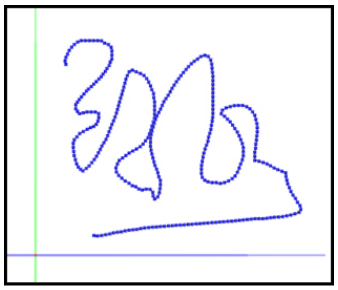
There are basically two different ways of storing a sketch from the user. Either it can be stored as time-ordered sequence of points (i.e. a *stroke*), or approximated to an image based representation[42]. Since we want to preserve temporal information, stroke based approach is better suited than image representation for this purpose.

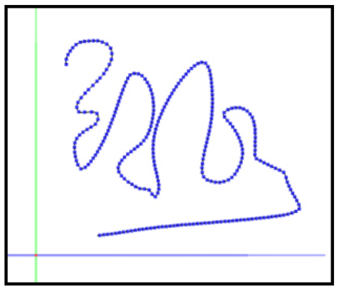
3.2.2 Sketch Filtering

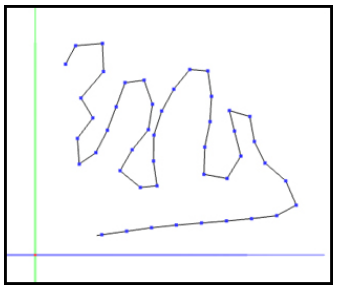
It is important to perform filtering before storing a sketch to the system, since there will be some error caused by both user and the input device itself[50]. The user's hand may shake while drawing, causing curves and lines that are not as smooth and linear as intended. Similarly, the pen tablet can also add some noise to the input while digitizing it. No matter how careful the user is, such errors will always exist. Therefore, the input data should be interpreted knowing that it is imperfect. To overcome this imperfection, our system applies below approaches:


(a)


(b)


(c)


(d)


(e)

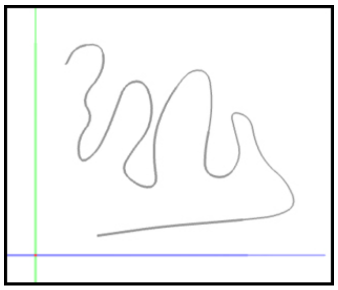

(f)

Figure 3.3: Re-sampling and Smoothing. **(a)** A user input would look like this before any re-sampling and smoothing **(b)** The distance between data points is not equal to each other. (Total of 706 points) **(c)** To make those distances even, the input data is re-sampled (Total of 2877 points) **(d)** A Gaussian filter is applied on the fly as well. **(e)** Reverse Chaikin subdivision is applied to simplify curve representation and further smoothing (Total of 47 points used to *represent* the curve) **(f)** Final result; a smooth B-spline curve (Total of 188 points is used to *render* the curve).

3.2.2.1 Re-sampling and Smoothing

The distance between consecutive data samples that are acquired from the pen tablet is not always the same (Fig. 3.3(b)). Because of this, data points that are sampled closer than a given threshold should be discarded. Similarly, if there is any data point that is sampled too far from the previous point, a interpolation must be performed between these two data points. In our system, this re-sampling is actuated on the fly (Fig. 3.3(c)). However, re-sampling is not sufficient alone. To further smooth out the given input, we use a local Gaussian filter (Fig. 3.3(d)) to any upcoming data point (i.e. the newly acquired data point is adjusted according to a Gaussian filter applied to that point and neighboring points)[54].

1. Calculate the adjusted position p_f of a newly added point P_f as,

$$p_f = \sum_{i=0}^{w_i > \epsilon} w_i p_i \tag{3.1}$$

$$w_i = f(i, \mu, \sigma^2) \tag{3.2}$$

$$f(i, \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \tag{3.3}$$

   where $\mu = 0$ and $\sigma^2 = 1$, hence weight function w_i is a *standard normal distribution*; and p_i is the i^{th} neighboring point's current location.

2. Notice sum iterates as long as w_i is greater than a small number (i.e. It iterates for 4-5 times).

Algorithm 2: The Gaussian filtering step. A standard normal distribution is used as a weighting function for neighboring points to adjust newly added point's final location.

The details of the Gaussian filtering step is explained at Algorithm 2. Basically, we use a standard normal distribution to decide on the weights assigned to each neighboring point of the newly added point. We adjust this new point by the calculated weighted average.

3.2.2.2 Fitting

After re-sampling and smoothing is performed, the resulting curve consists of hundreds of data points. To simplify this representation, we fit a curve onto these data points, using Reverse Chaikin Subdivision[8]. Reverse Chaikin Subdivision is a standard algorithm to produce less number of coarse points that will represent a larger number of fine points. At every iteration of this algorithm, the data size halves. After appropriate number of iterations, these coarse points are used as control points for a B-Spline curve (Fig. 3.3(e)). Running some empirical tests, we concluded that six iterations are suitable for our case. Assuming fine points are denoted as p_1,p_2,...,p_n, and coarse points are denoted as c_1,c_2,...,c_m, a coarse point c_j can be computed as:

$$c_j = -\frac{1}{4}p_{i-1} + \frac{3}{4}p_i + \frac{3}{4}p_{i+1} - \frac{1}{4}p_{i+2} \tag{3.4}$$

where the step size of i variable is two (hence halving the cardinality of the fine points).

3.2.3 Sketch Interpretation

After input sketch is acquired and filtered, the last step of the sketching pipeline is to interpret the given stroke. In our system, the pen tablet acts like a *gestural interface* for users, allowing it to be used for several different tasks. The user can switch between different modes by holding down the buttons on the pen. During a regular session with the system, one will use the pen for camera adjustment, plane selection, drawing, erasing and editing. Hence the detected input stroke will be mapped to one of these gestures, as explained below.

3.2.3.1 Camera Adjustment

When the pen is in this mode, every movement that the user does will be mapped to an invisible *Two-Axis Valuator Trackball*[16]. The horizontal pen movement is mapped to a rotation about the up-vector, whereas a vertical pen movement is mapped to a rotation about the vector perpendicular to view and up, as explained at Algorithm 3. Any diagonal movement is also easily mapped to a combined rotation (Fig. 3.4). As Bade et al. suggest[4], Two-Axis Valuator Trackball is “the best 3D rotation technique” among several other rotational widgets.

1. Calculate normalized x and y offsets Δx_n and Δy_n as,

$$\Delta x_n = \frac{x_{current} - x_{initial}}{screenWidth} \quad , \quad \Delta y_n = \frac{y_{current} - y_{initial}}{screenHeight} \tag{3.5}$$

2. Given $\vec{S_i}$ is the initial view point position at *spherical coordinate system*, current view point position $\vec{S_c} = \langle S_{c,x}, S_{c,y}, S_{c,z} \rangle$ becomes,

$$\vec{S_c} = \vec{S_i} + \langle -\Delta x_n \pi, \Delta y_n \pi, 0 \rangle \tag{3.6}$$

3. Using spherical coordinate $\vec{S_c}$, the current position in *Cartesian coordinate system* $\vec{C_c} = \langle C_{c,x}, C_{c,y}, C_{c,z} \rangle$ can easily be computed as,

$$C_{c,x} = S_{c,z} \times sin(S_{c,y}) \times cos(S_{c,x}) \tag{3.7}$$

$$C_{c,y} = S_{c,z} \times cos(S_{c,y}) \tag{3.8}$$

$$C_{c,z} = -S_{c,z} \times sin(S_{c,y}) \times sin(S_{c,x}) \tag{3.9}$$

4. When camera adjustment gesture is finished (i.e. pen is lifted up from tablet), update initial spherical view point position as,

$$S_i = S_c \tag{3.10}$$

Algorithm 3: Two-Axis Valuator Trackball. The normalized offset of the cursor is used to calculate current spherical coordinates, which is the step before calculating real Cartesian coordinates.

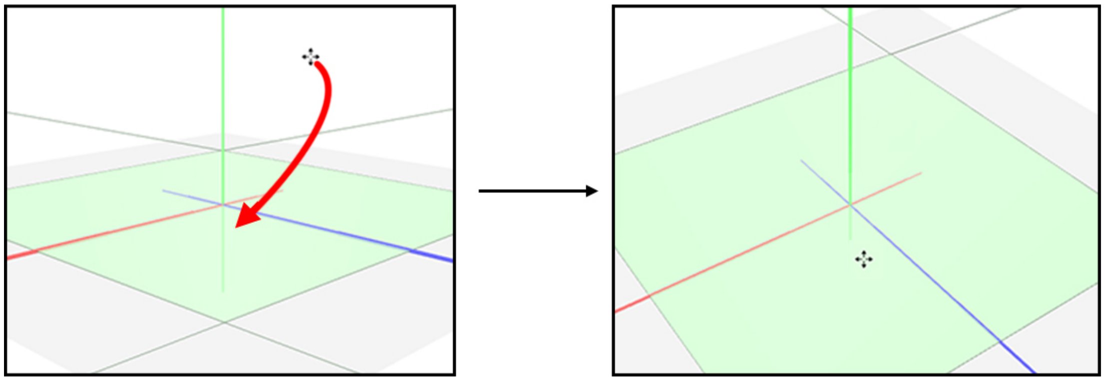


Figure 3.4: Camera adjustment. Any pen movement is mapped to an invisible Two-Axis Valuator Trackball.

3.2.3.2 Plane Selection:

When the user draws curves with the tablet, these curves should be reflected onto a virtual surface in the 3D scene. To enable this effect, the user should select a *drawing plane* beforehand. In our system, there are only two distinct ways of selecting the drawing plane.

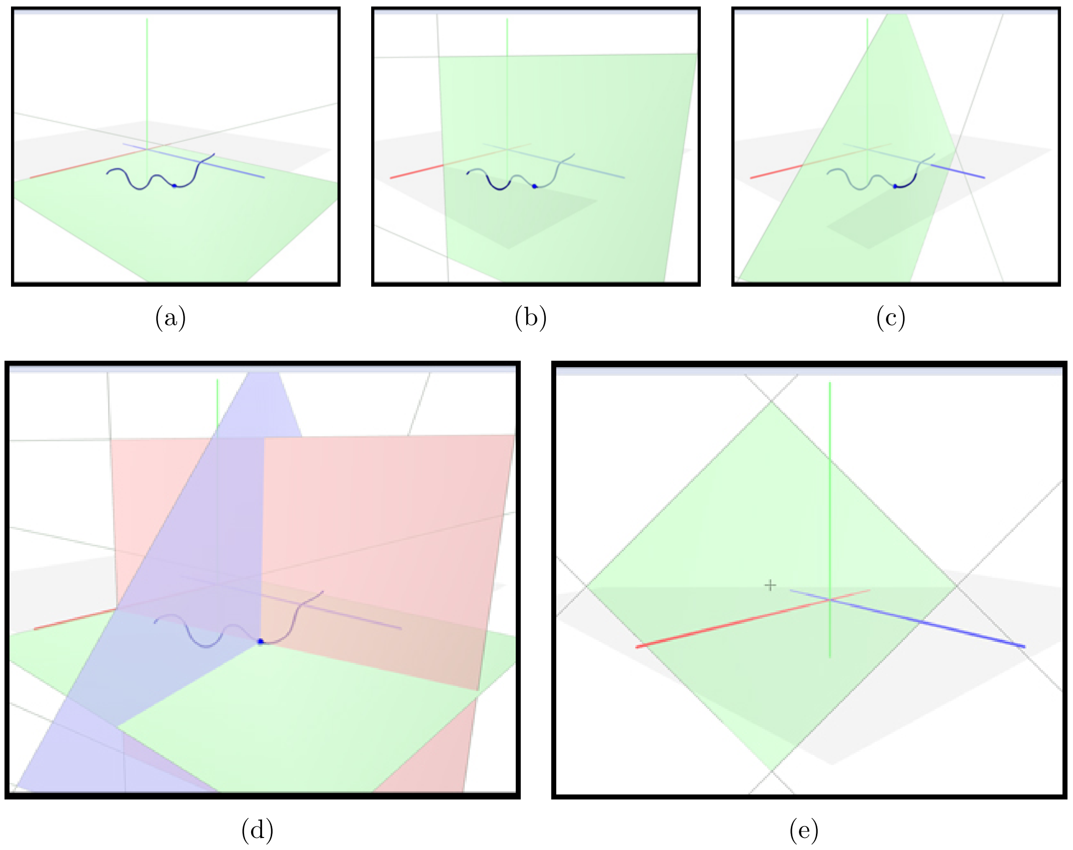


Figure 3.5: Plane selection. **(a)** The plane that selected curve lies. **(b)** The plane that's tangential to the selected curve and perpendicular to its plane. **(c)** The plane that is perpendicular to both (a) and (b). **(d)** The Cartesian coordinate system that is formed by those three planes. **(e)** The plane that is adjusted by extruding a picking ray from the current viewpoint. This plane is parallel to current viewport's near plane.

- In the first available approach, the user takes an assistance from coordinate system lines and current curves on the scene. By selecting any of these curves and lines, the user changes the drawing surface as the plane that selected curve lies in (Fig. 3.5(a)). Further flexibility is enabled with the help of *toggle plane* button on the tablet. Once that button is pushed, the drawing surface will be changed to the plane that's tangential to the selected curve from the selection point, and perpendicular to the plane that the curve lies (Fig. 3.5(b)). Another toggle will change the drawing surface once more, this time as the plane that is perpendicular to both the first plane and the tangential plane (Fig. 3.5(c)). By the help of such three planes, the user can form a mental model of the Cartesian coordinate system at any position and orientation (Fig. 3.5(d)).

- To support even more flexibility, we realized a second approach to plane selection. In this method, the user can adjust the drawing surface to a plane that is parallel to the current near plane of the scene's viewport, and x distant from that near plane, where that x is determined by the current pressure on the pen (Fig. 3.5(e)).

3.2.3.3 Drawing

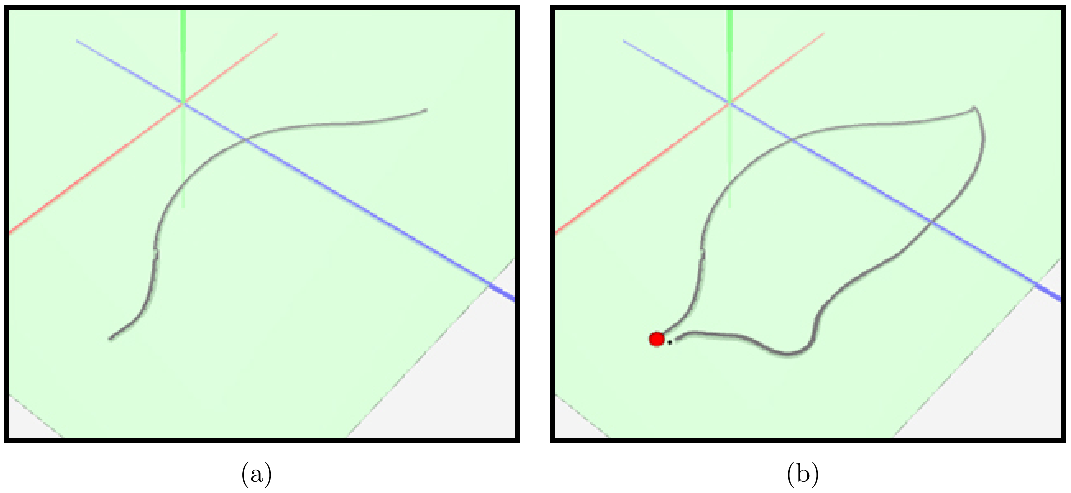

(a) (b)

Figure 3.6: Drawing. **(a)** User can draw arbitrary shaped curves **(b)** Snap points can help to create connected curves.

The main functionality of the system is drawing curves (Fig. 3.6(a)). In this mode, the user can simply draw several curves using the pen tablet. The time sequenced *(x,y)* data that is collected from the pen tablet is then projected to the current drawing plane. After the projection is performed, several re-sampling and smoothing algorithms are used to ensure a plausible curve shape, as detailed in Section 3.2.2. Finally, a B-Spline curve is fitted to the stroke data. While in the drawing mode, the user can take advantage of *snap points* that will appear at the start and end points of existing curves (Fig. 3.6(b)). These snap points make it easier to draw closed or connected shapes.

3.2.3.4 Erasing

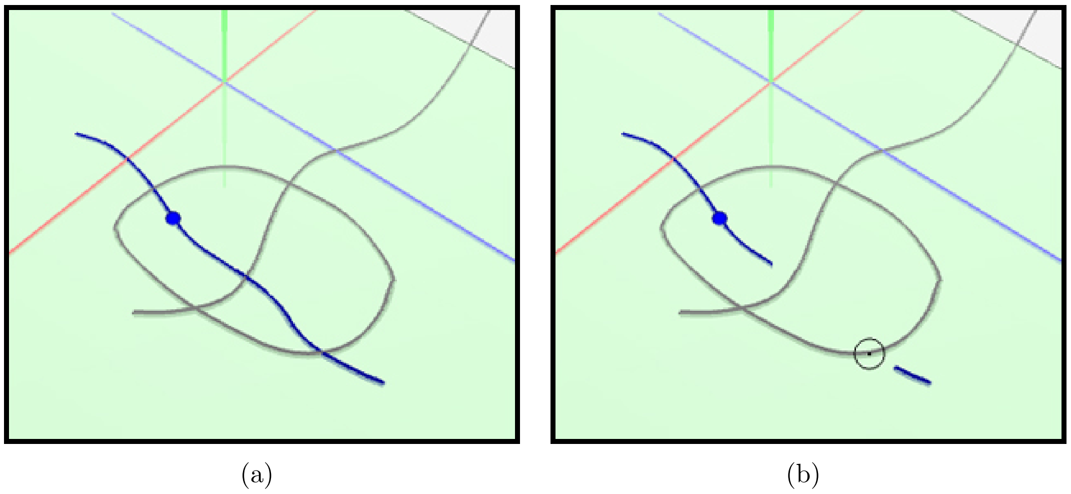


Figure 3.7: Erasing. **(a)** User selects the curve to be erased. **(b)** Performs the erasing with simply turning over the pen and erasing the part he wants.

A paper and pen system cannot be imagined without an eraser. The user can simply turn over his pen device to switch to the eraser mode. Once this is done, the cursor on the screen will get larger to mimic an eraser functionality. Since in a crowded scene, there will be several curves that will lie under eraser's cursor, it will be harder to erase a specific curve's segment. Therefore, erasing can only be performed on the current *selected* curve (Fig. 3.7).

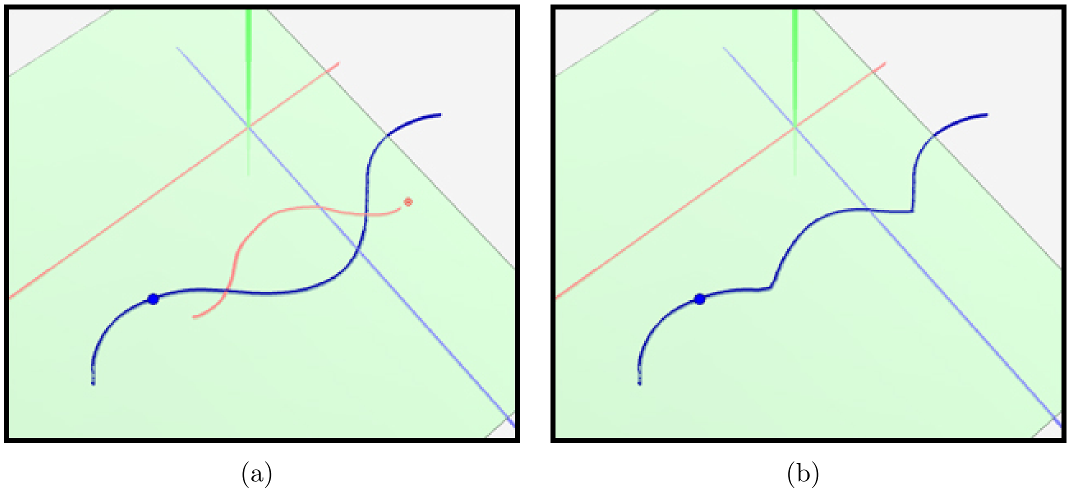


Figure 3.8: Editing. **(a)** User selects the curve to be edited, and draws the edit curve. **(b)** Final result.

3.2.3.5 Editing

Sometimes, the user may want to edit a section of a curve that has a minor flaw. In such a situation, it may be appropriate to use the editing tool instead of erasing that part and re-drawing. Just as in erasing tool, editing is also only performed at the current selected plane (Fig. 3.8).

Over-sketching is a commonly used gesture, when there is a curve region that needs to be corrected. Once a secondary stroke is drawn, the system can update the initial curve by slicing it into segments, and replacing the old segment with the new stroke. A smoothing algorithm is also performed between the transitions of these segments. We use Fleisch et al.[24]'s algorithm to over-sketching that is explained in Algorithm 4.

3.2.3.6 Miscellaneous Operations

As mentioned, there are also several buttons on the tablet, that can be used to achieve miscellaneous operations. When symmetry is toggled, using one of these buttons, any gesture that is performed with the pen will also be reflected to the symmetry of that gesture. For instance, if a curve is drawn when the

1. The algorithm replaces curve segment C_d with the newly sketched segment C_o, resulting in C_r where,

$$\begin{aligned} C_d &= (x_0, x_1, \cdots, x_n), \\ C_o &= (y_0, y_1, \cdots, y_m), \\ C_r &= (r_0, r_1, \cdots, r_o) \end{aligned} \tag{3.11}$$

2. The start x_s and end x_e points of the segment are found by finding the minimum distance between x_i and y_0 and y_m as,

$$\begin{aligned} x_s &= min(|x_i - y_0|), \quad i = 0, \cdots, n, \\ x_e &= min(|x_i - y_m|), \quad i = 0, \cdots, n \end{aligned} \tag{3.12}$$

3. The resulting curve then contains the points,

$$C_r = (x_0, \cdots, x_{s-1}, y_0, \cdots, y_m, x_{e+1}, \cdots, x_n) \tag{3.13}$$

4. The resulting curve is further smoothed to reduce hard breaks. To do that, the smoothing equation below is applied to $\frac{m}{2}$ neighbors from start and end points of over-sketched segment, where m is the length of that segment:

$$x_t = x_t + (y_0 - x_s) \times (0.5 \times sin(t + 0.5\pi) + 0.5) \tag{3.14}$$

where either $\quad s - 1 - \frac{m}{2} < t < s - 1$, or $\quad e + 1 < t < e + 1 + \frac{m}{2}$

Algorithm 4: Constraint Stroke-Based Oversketching for 3D Curves. Reprinted from [24].

symmetry is enabled, a symmetric curve will also be created. Similarly, if a part of a curve will be erased while symmetry is enabled, the same part will be erased for its symmetric counter as well. Symmetry is important to product design, since people prefer objects with symmetry, unity and harmony[10].

To prevent errors that the users might make, the system changes the pen's cursor's image to reflect the current gesture mode[3]. For instance, it's a single dot for drawing, a slightly bigger *red* circle for editing, a cross-hair for plane selection etc. Similarly, our system also supports undo/redo actions using the tablet buttons as well. This functionality is essential for basic error recovery. As can be seen in Section 4, undo is widely used among our users.

3.3 Visualization Pipeline

Correct visualization of a scene is fairly important to make it easier for users to understand the 3D information at the scene. As in real world, computer generated scenes take advantage of depth cues to emphasize 3D. Out of three groups of depth cues (oculomotor, monocular, binocular), monocular depth cues are possible to implement using a standard computer display. One should use stereo-displays or anaglyph rendering if the aim is to benefit binocular cues as well.

Since our system also uses a generic computer display, we used monocular depth cues for 3D visualization. We have a few simple algorithms about *pictorial cues* explained further in the upcoming section (Section 3.3.1). We also tracked user's face to achieve *kinetic depth effect* using *motion parallax*, hence emphasized *motion-based cues.* The details of face tracking approach is explained in Section 3.3.2.

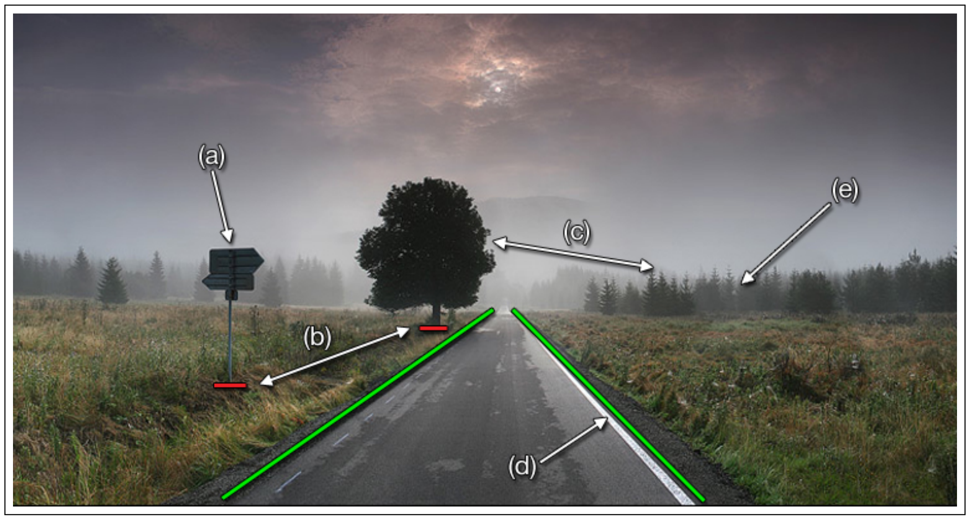
3.3.1 Pictorial Depth Effects

Even with a flat image, it is possible to understand the depth relation of objects with the help of so called pictorial cues. There are several studies that have aimed to explain the interaction of depth cues when several of them are present within a scene. Some of these studies propose that depth cues are combined in an additive fashion[33, 32, 58], others suggest there is a non-linear relation between different depth cues[12, 46]. Whether linear or non-linear, one aspect all these studies seem to agree is that multiple depth cues help disambiguation of the visual stimuli and with the more depth cues, the better the depth perception of the observer[28].

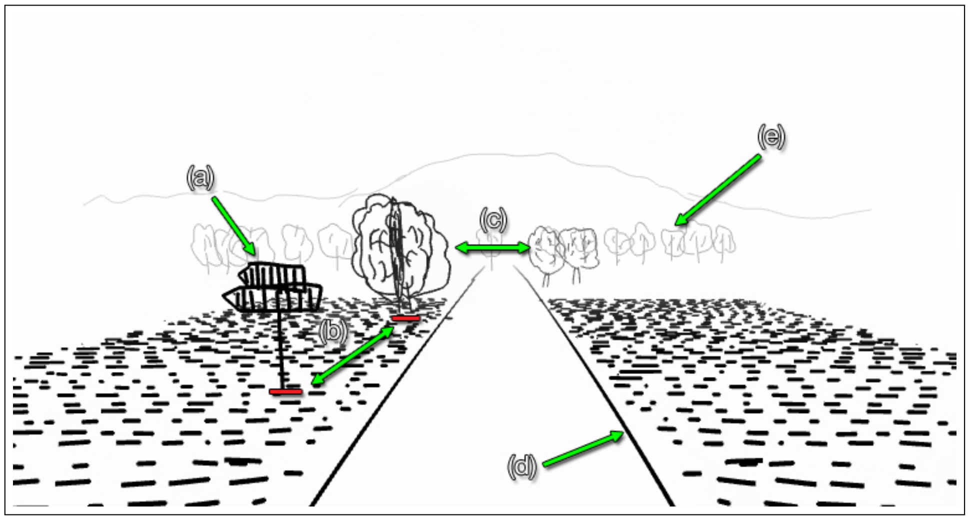
With this reasoning in mind, we have attempted to implement as many non-conflicting depth cues as possible. The resulting effect is almost identical to the one that we expect from a real world scene. As demonstrated at Fig. 3.9, rendered from a similar viewpoint; our system emphasizes the same pictorial cues as does the real world image: occlusion, relative height, relative size, perspective convergence, and atmospheric perspective.

Perspective projection is actually responsible for some of these pictorial depth cues, and we did not have to take any special measures for it. These pictorial cues are **occlusion**, **relative height**, **perspective convergence**, and **texture gradient**. One can expect that relative size should be listed as a perspective-projection-resulted depth cue. But that is not the case for our system. The scenes we create consist of mere line polygons, and OpenGL does not adjust the line width according to depth. Therefore propose a solution, as explained in Algorithm 5. At every render loop, we are calculating the distance of each line from the viewpoint. Then using the normalized distance value, we are setting a proper line width and color for that line segment. The change in width results in **relative size**, while the change in color emphasizes **atmospheric perspective** (Fig. 3.10).

This idea is also supported by technical illustrators. In technical illustration, there are three line conventions suggested by Judy Martin[38]: use single line weight throughout the image; use heavy line weights for out edges, and parts



(a) A real world scene demonstrating pictorial depth cues



(b) A 3D replica of the same scene at our system.

Figure 3.9: A 3D replica of a real world scene at our system. Notice how we preserved pictorial depth cues at rendering; (a) occlusion, (b) relative height, (c) relative size, (d) perspective convergence, (e) atmospheric perspective

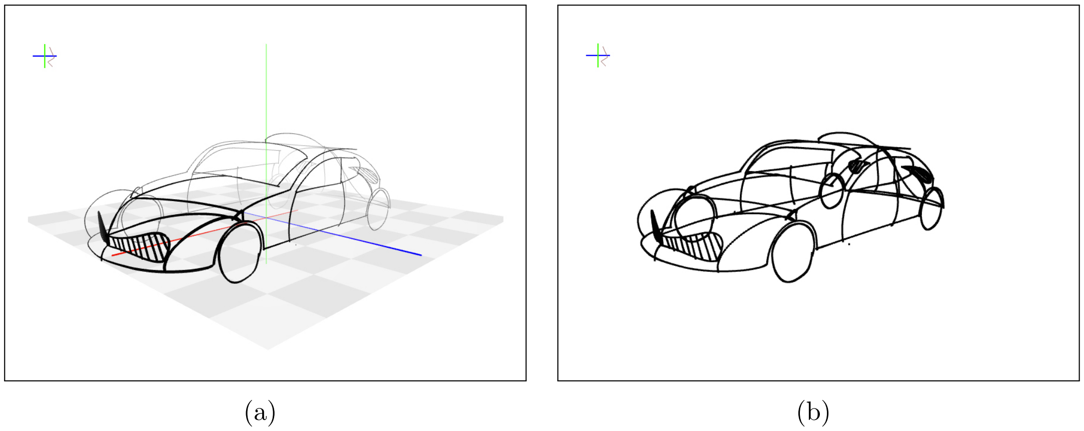


Figure 3.10: Visualization at our system. **(a)** with depth cues. **(b)** without depth cues.

```
1. Define $minDistance$ and $maxDistance$ as global variables.
2. At each render loop, for every line segment;
    (a) Calculate current line segment $l$'s distance $d$ from current viewpoint $v$,
    (b) Update $minDistance$ and $maxDistance$, if current distance $d$ is eligible to replace them.
    (c) Normalize distance $d$ using equation;
            $d = \frac{d - minDistance}{maxDistance - minDistance}$    (3.15)
    (d) Use normalized distance to decide on line thickness and color in a linear relationship fashion.
```

Algorithm 5: The pre-render process of emphasizing relative size and atmospheric perspective by varying line thickness and color. Notice that *minDistance* and *maxDistance* variables converge after first iteration of the render loop.

with open space between them; vary line weight to emphasize perspective (i.e. thicker is closer). Since our concern is to emphasize 3D recognition as much as possible, we find third convention most suitable for the system.

To better emphasize **perspective convergence**, and **texture gradient**, we also used a checker textured ground effect at the drawing scene (Fig. 3.10(a)). Combined with the coordinate axis rendered, that checker effect introduces a scene depth effect.

3.3.2 Face Tracking for Kinetic Depth Effect

Kinetic depth effect is another important depth cue. When an object is rotating around its axis, the three dimensional structural form can be better perceived. In our system, we achieve this kinetic depth effect by motion parallax via tracking user's face position. The first step of face tracking pipeline is to locate face from the captured image by means of a hybrid algorithm of *Viola-Jones Face Detector* and *CamShift Face Tracker*. Then, the location found is filtered using *Kalman Filtering* to reduce noise. Finally the filtered location is fed to the *Spherical Motion Parallax* method, where it is mapped as the virtual camera position for the scene (i.e. *Orbital Viewing*). Upcoming sections explain this pipeline in detail.

3.3.2.1 Tracking with Viola-Jones & CamShift

We combine Viola-Jones and CamShift algorithms (implementation available in OpenCV[13]), to come up with a powerful face tracking module. The pipeline of the hybrid algorithm is shown at Fig. 3.11. The module starts running at initialization state. As soon as a face is detected by Viola-Jones Detector, the search window for CamShift Tracker is set to this detected face's rectangle and the color histogram for the face is calculated from the region. After this step, the state changes from *initialization* to *tracking loop*.

As can be understood from Fig. 3.11, during tracking, our algorithm alternates

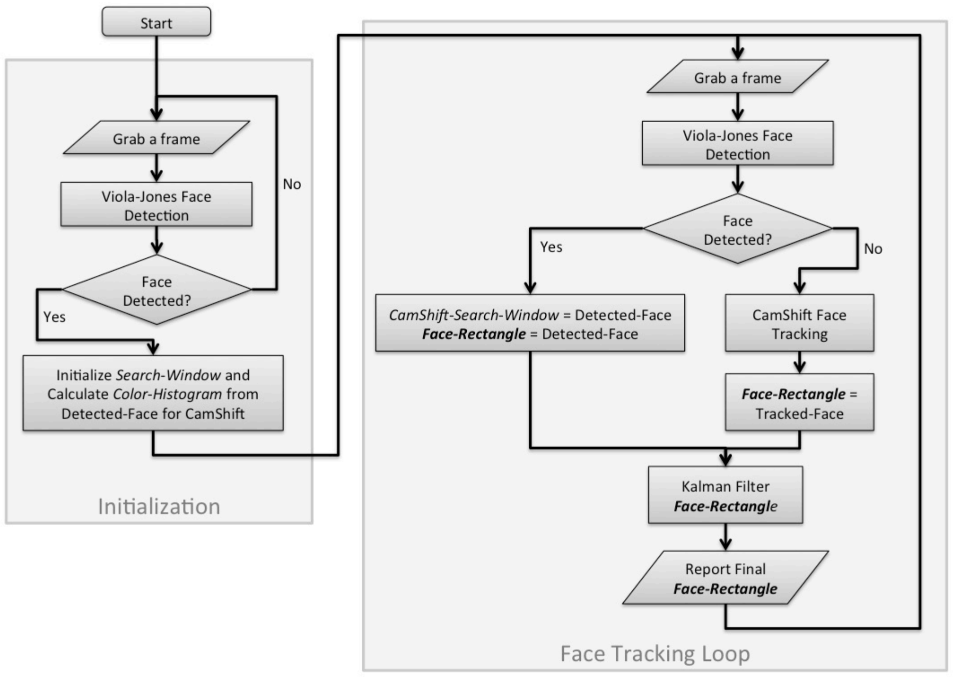

Start
Grab a frame
Viola-Jones Face Detection
Face Detected?
No
Yes
Initialize Search-Window and Calculate Color-Histogram from Detected-Face for CamShift
Initialization
Grab a frame
Viola-Jones Face Detection
Face Detected?
Yes
No
CamShift-Search-Window = Detected-Face
Face-Rectangle = Detected-Face
CamShift Face Tracking
Face-Rectangle = Tracked-Face
Kalman Filter Face-Rectangle
Report Final Face-Rectangle
Face Tracking Loop

Figure 3.11: Face Tracking Pipeline.

between Viola-Jones Detector and CamShift Tracker. As long as there is a face detected by Viola-Jones, we simply do not run CamShift Tracker, and feed this detected face as the real face rectangle to Kalman Filter step. But if Viola-Jones can not detect a face, the CamShift Tracker takes over and tries to track the face by using the last known detected face as initial search window (Fig. 3.12). The resulting tracked region is fed to Kalman Filter as the real face rectangle to reduce noise, as explained in the next section.

To be able to run Viola-Jones Detector in real time for every frame, we take some measures. The detection is pruned as soon as a single face is detected. Therefore, if there are several persons in the scene, the one that is closer to the camera will be detected (Fig. 3.13). This behavior is actually beneficial for our case, since it eliminates the accidental interaction of non-users.

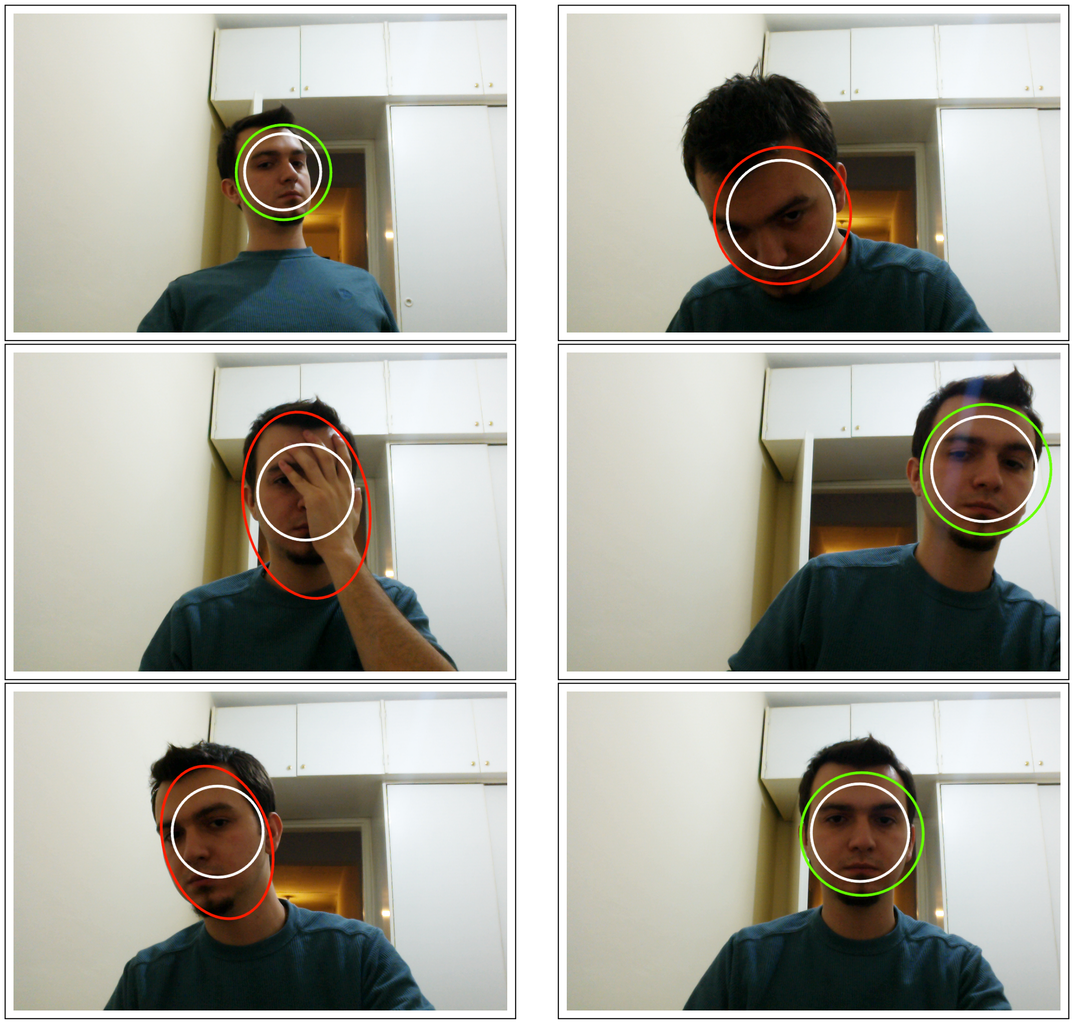


Figure 3.12: Sample Face Tracking results by our system. **Green** ellipses represent Viola-Jones Detection results, while **red** ellipses are for CamShift, that is performed when Viola-Jones fails to detect any face. Finally, the **white** circle is the resulting face circle after normalizing and Kalman Filtering.

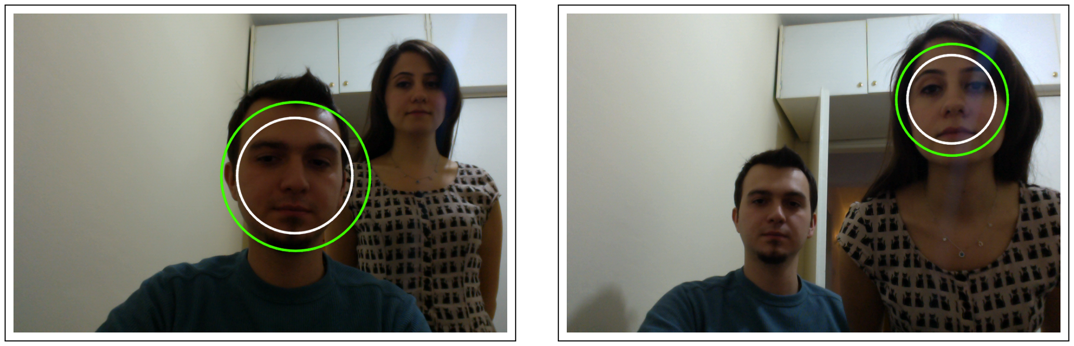


Figure 3.13: The Viola-Jones detector will detect the closer person and stop further computation, increasing detection performance.

3.3.2.2 Kalman Filtering

Similar to An et al.'s work[2], we feed the face rectangle reported by Viola-Jones/CamShift Tracking to a simple linear Kalman Filter. The Kalman Filter smooths out the reported face center position $(x_k,\, y_k)$ and size σ_k. The model we feed Kalman Filter is:

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \mathbf{v}_{k-1} + \mathbf{n}_{x,k-1}, \tag{3.16}$$

$$\mathbf{v}_k = \mathbf{v}_{k-1} + \mathbf{n}_{v,k-1} \tag{3.17}$$

where $\mathbf{x}_k = (x_k, y_k, \sigma_k)$ is the position and size of the face rectangle at time t_k, and $\mathbf{v}_k = (v_{x,k}, v_{y,k}, v_{\sigma,k})$ is velocity. $\mathbf{n}_{x,k-1}$ and $\mathbf{n}_{v,k-1}$ are simply zero-mean Gaussian random functions representing the environment noise. The state transition equation for **estimation** becomes:

$$\mathbf{S}_k = \mathbf{\Phi}_{k-1}\mathbf{S}_{k-1} + \mathbf{N}_{k-1}, \quad \mathbf{\Phi} = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{N}_{e,k-1} = \begin{pmatrix} \mathbf{n}_{x,k-1} \\ \mathbf{n}_{v,k-1} \end{pmatrix} \tag{3.18}$$

where state vector at time t_k is $\mathbf{S} = (x_k, y_k, \sigma_k, v_{x,k}, v_{y,k}, v_{\sigma,k})$, and $\mathbf{\Phi}$ is time-invariant state transition matrix. This leads the**measurement** equation to be:

$$\mathbf{Z}_k = \mathbf{H}\begin{pmatrix} \mathbf{x}_k \\ \mathbf{v}_k \end{pmatrix} + N_{m,k}, \quad \mathbf{H} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \tag{3.19}$$

Given this model, Kalman filter can iterate through the estimates. The algorithm works in a two-step process: **prediction** and **update**. In the prediction step, the Kalman filter produces estimates for the current state variables. Then

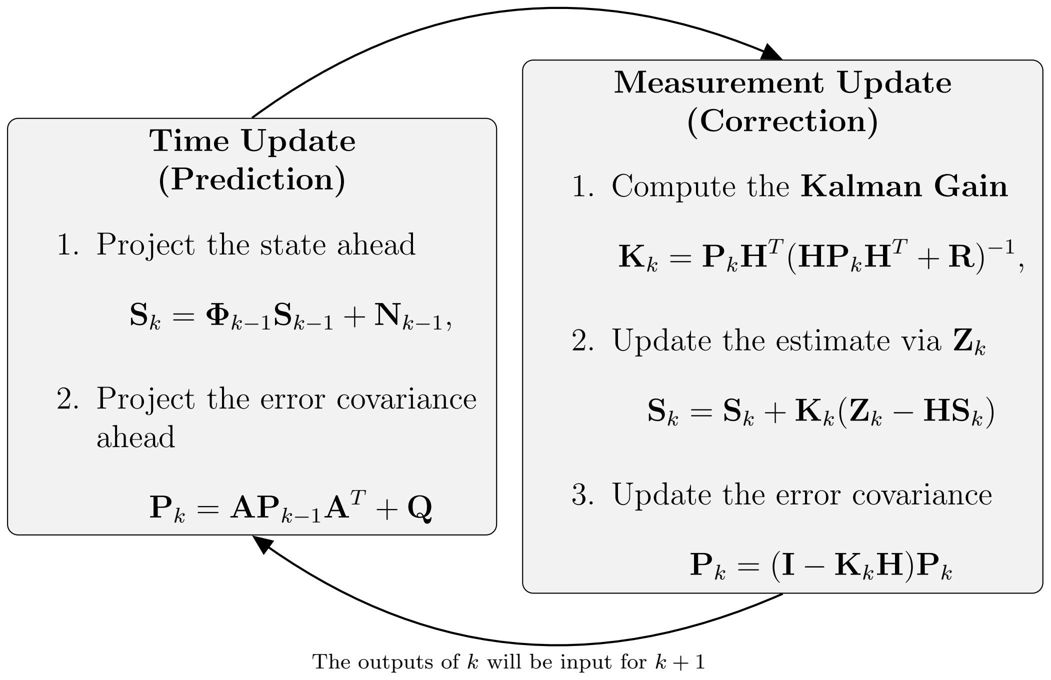

Time Update (Prediction)
1. Project the state ahead
$\mathbf{S}_k = \mathbf{\Phi}_{k-1}\mathbf{S}_{k-1} + \mathbf{N}_{k-1}$,
2. Project the error covariance ahead
$\mathbf{P}_k = \mathbf{A}\mathbf{P}_{k-1}\mathbf{A}^T + \mathbf{Q}$
Measurement Update (Correction)
1. Compute the Kalman Gain
$\mathbf{K}_k = \mathbf{P}_k\mathbf{H}^T(\mathbf{H}\mathbf{P}_k\mathbf{H}^T + \mathbf{R})^{-1}$,
2. Update the estimate via $\mathbf{Z}_k$
$\mathbf{S}_k = \mathbf{S}_k + \mathbf{K}_k(\mathbf{Z}_k - \mathbf{H}\mathbf{S}_k)$
3. Update the error covariance
$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{H})\mathbf{P}_k$
The outputs of k will be input for $k+1$

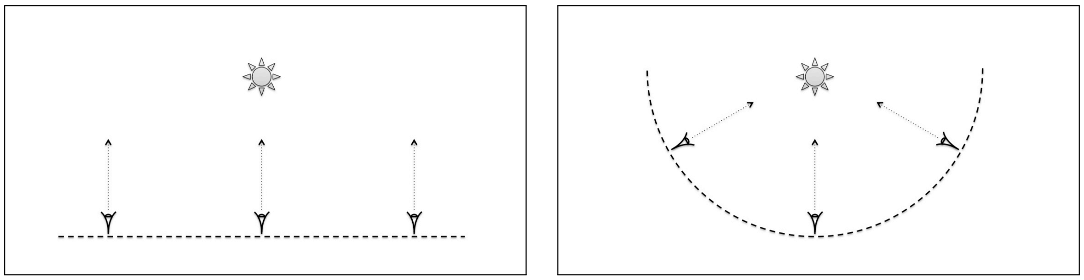
Figure 3.14: Kalman Filtering works in a two-step process: prediction and update.

the current measurement is fed to the algorithm to adjust the model. The process is explained in Fig. 3.14. The updated estimation value $\mathbf{S}_k$ after the correction step is used as the real face rectangle that is given to the *Spherical Motion Parallax* step.

3.3.2.3 Spherical Motion Parallax

The final step of the Face Tracking Pipeline is to map the detected face position so as to move the viewpoint of the user. In a traditional motion parallax system, the gaze direction does not change as user's head position changes. Similarly the viewpoint moves about a planar surface that is parallel to viewport (Fig. 3.15(a)). This camera mode is not suitable for scene editing, since it is not easy to navigate around the object to look at the occluded sides. Instead, we are using a motion parallax paradigm where we map user's head position to a surface of a virtual *sphere*. We further restrict the gaze direction to always be facing to the center of this sphere, rather than straight perpendicular to viewport (Fig. 3.15(b)).

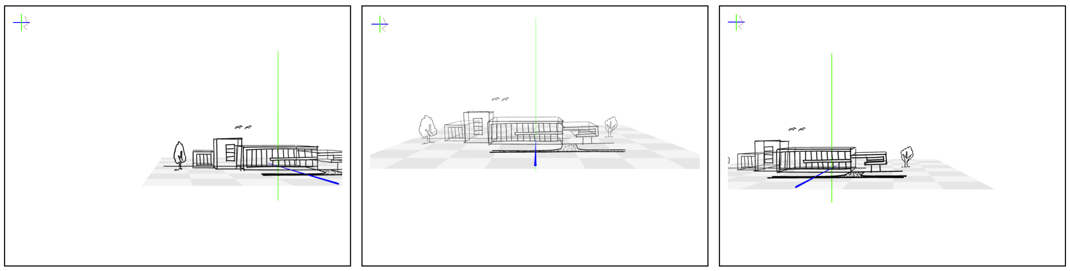
A similar approach, *Orbital Viewing*[36], exists in the literature; where a head mounted display is used to map the rotation of the user's head to the viewpoint.

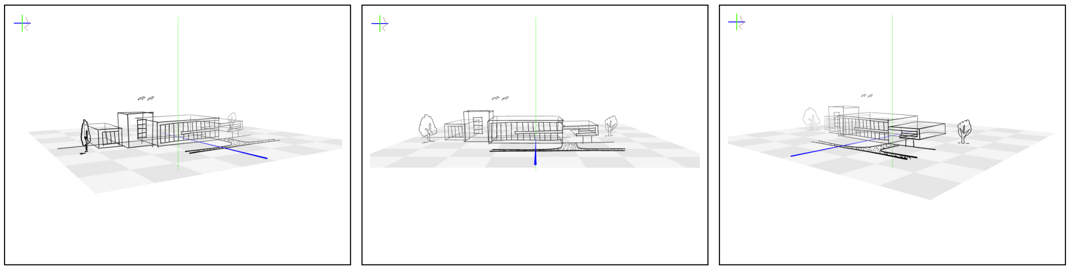


(a) (b)

Figure 3.15: Motion Parallax. **(a)** Planar (Traditional) vs. **(b)** Spherical (Orbital Viewing).

The calculations necessary for this mapping are already discussed at Section 3.2.3.1 Camera Adjustment, since essentially there is no difference between mapping face tracking offset versus mapping pen gesture offset to the virtual camera.


(a) Planar Motion Parallax


(b) Spherical Motion Parallax

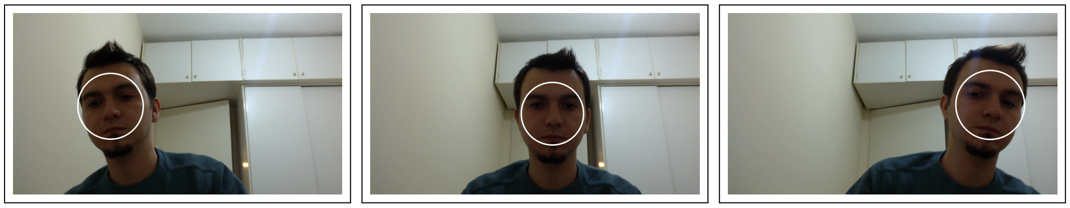

(c) User Position's for Each Case

Figure 3.16: Motion Parallax for Scene Editing. Notice how a rotating spherical motion parallax enables better camera directions for editing.

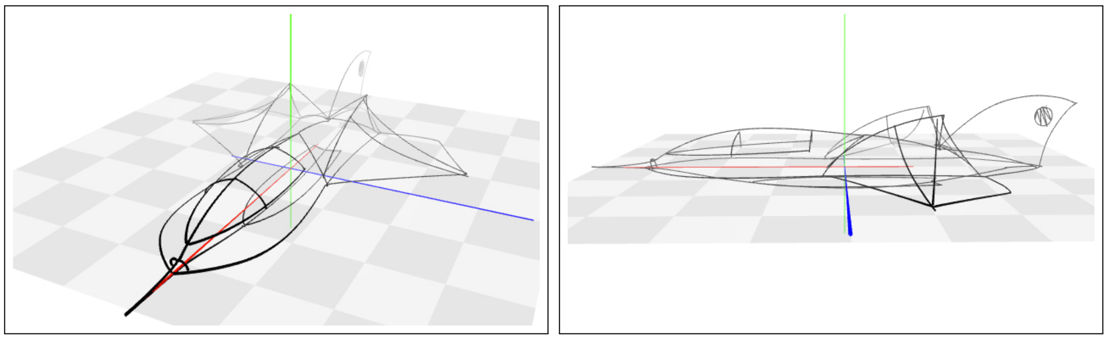
Chapter 4

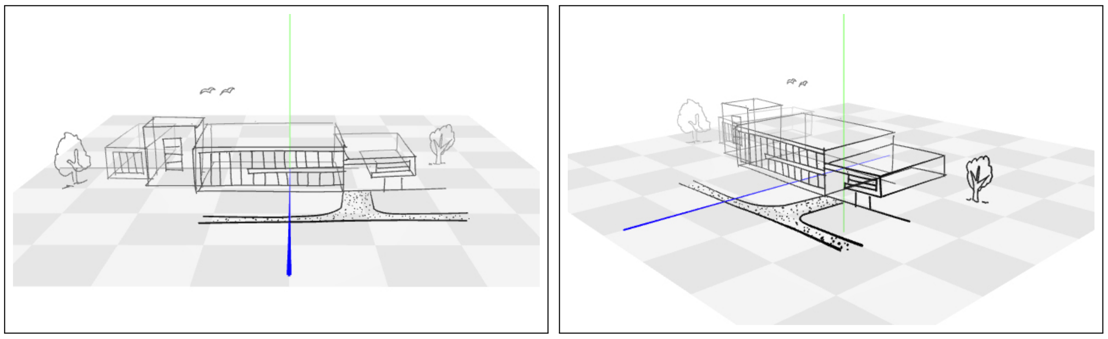
Evaluation, Results & Discussion

We have conducted two separate sets of tests to evaluate our system. First, we invited an architect to use the system for a day, producing few sample objects and giving feedback on the system. This test basically verifies our system is expressive enough to be used in actual product design (Section 4.1). Later, we also conducted a formal experiment, where several non-professional users were expected to complete a twelve-step objective test, detailed in Section 4.2.

4.1 Expert Evaluation

We invited an architect to perform a subjective expert evaluation. After a brief introductory explanation of the system for 15 minutes, the architect is left alone with the system for a full day. The resulting objects of the day can be seen at Fig. 4.1. The architect stated that he did like using the system, but he thought such a system was more suitable for product design than architectural design. We agree on this comment, since our system tries to emphasize the power of free form curves, it is actually more difficult to create regular shapes such as cubes and pyramids. One usability issue that we noted was that the architect preferred using undo function instead of erasing and editing gestures most of the time. Only for some small adjustments, like shortening a curve which is a little too


(a) A jet fighter.


(b) A building complex.

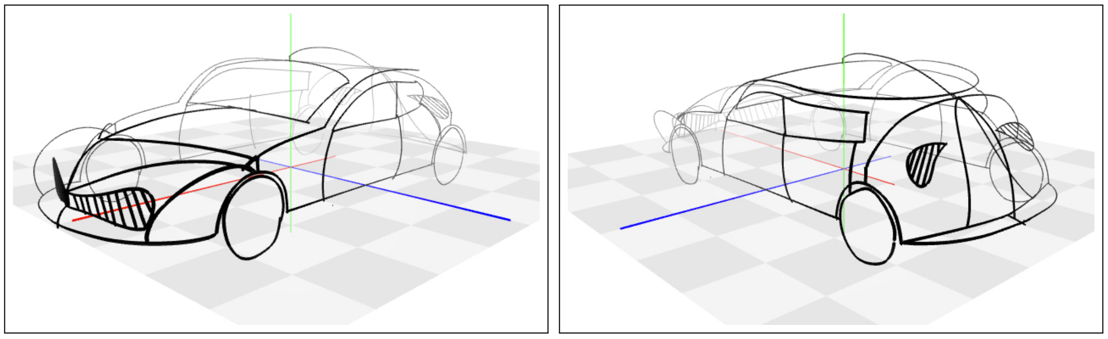

(c) A car.

Figure 4.1: Sample Results created using our system.

long, he used erasing.

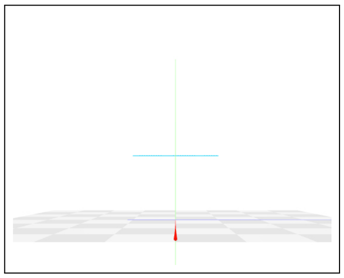
4.2 User Evaluation

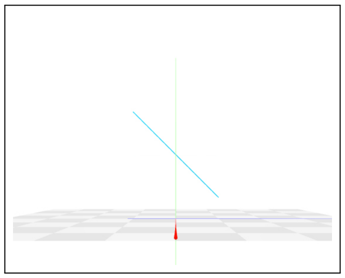
We had also performed objective formal experiment to evaluate the usability of the system. We had selected twelve users that did not have prior experience with technical or artistic drawing, and pen tablets. In a standard test case, we introduced the system to each user briefly within five minutes. Then, we asked them to exactly copy the object they see on the scene. The test consisted of twelve objects, some of which were 2D regular shapes, while others included 3D objects (Fig. 4.2).

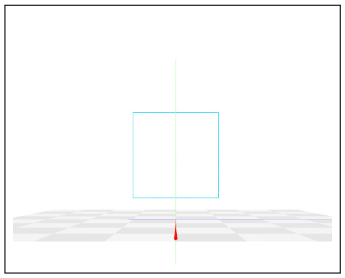
ISO 9241-11 standard defines usability as "extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction"[**?**]. Following this description, Hornbæk[**?**] classifies usability measures into three groups *effectiveness*, *efficiency*, and *satisfaction*. Effectiveness is "the accuracy and completeness with which users achieve specified goals", efficiency is "the resources expended in relation to the accuracy and completeness with which users achieve goals", and satisfaction is "the freedom from discomfort, and positive attitudes towards the use of the product"[**?**]. To be as accurate as possible, we follow the same categorization and position our user evaluations into these three groups, as explained in upcoming sections.

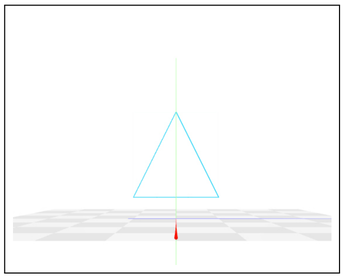
4.2.1 Effectiveness

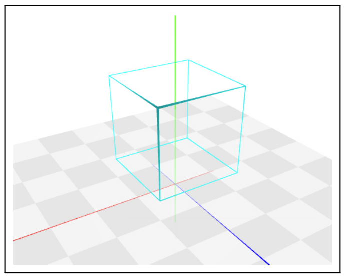
There are several measures about effectiveness that could be performed at a usability test. Hornbæk[**?**] lists few measures, such as *binary task completion*, *accuracy*, *recall*, *completeness*, and so on. For the sake of our user test, we had the goal to measure *accuracy* by quantifying the error made by users during the process of sketching scenes. For twelve test cases demonstrated at Fig. 4.2, the users were asked to copy the exact same objects they see. Then, we compared the

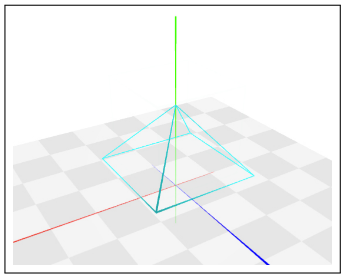

(a) Line

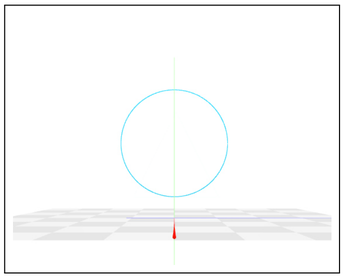

(b) Diagonal Line

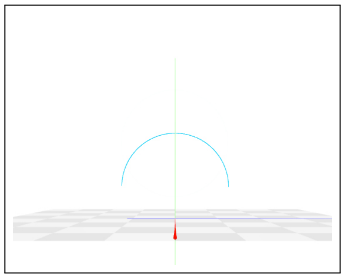

(c) Square

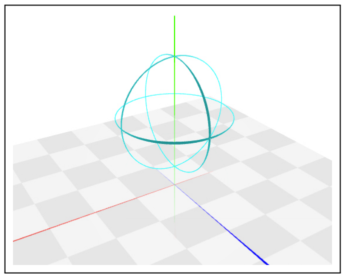

(d) Triangle

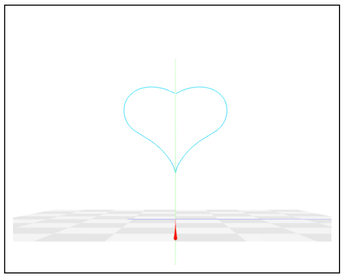

(e) Cube

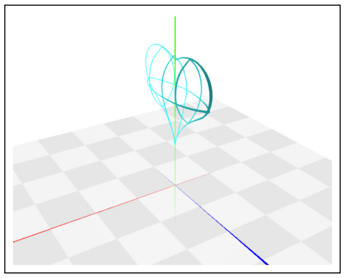

(f) Pyramid


(g) Circle


(h) Semi-Circle


(i) Sphere


(j) Heart


(k) Heart (3D)

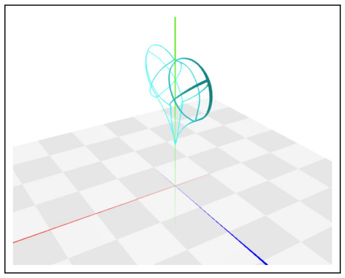

(l) Heart (3D)

Figure 4.2: Twelve test cases in the actual order when test is performed.

resulting scenes with the goal objects using a modified Hausdorff distance measure by Dubuisson et al.[23]. In Dubuisson's work [23], the authors have compared several different versions of Hausdorff distance measures and concluded the one we describe below (Modified Hausdorff Distance a.k.a *MHD*) performs best.

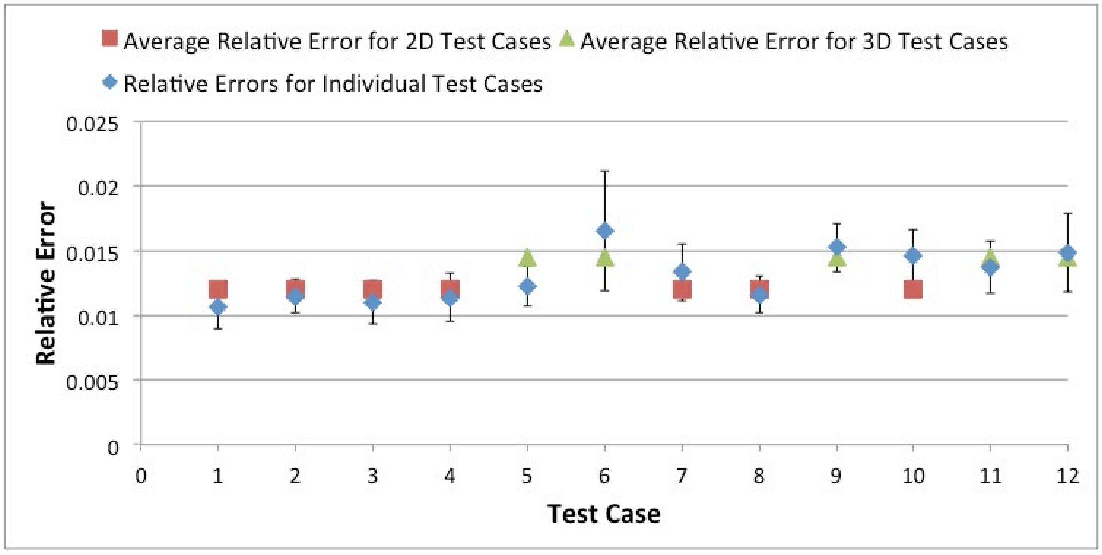


Figure 4.3: Relative Errors. The error is calculated by dividing the Modified Hausdorff Distance measure by 10 (the common test object diameter.

Given the distance between two points a and b is defined as the Euclidean distance $d(a, b) = ||a - b||$, the distance between a point a and a set of points $B = \{b_1, \cdots, b_m\}$ can be defined as $d(a, B) = min_{b \in B}||a - b||$. Whereas, the directed distance between two sets of points $A = \{a_1, \cdots, b_n\}$ and $B = \{b_1, \cdots, b_m\}$ is defined by MHD as,

$$d(A, B) = \frac{1}{n} \sum_{a \in A} d(a, B) \tag{4.1}$$

Two directed distances between sets $d(A, B)$ and $d(B, A)$ can be combined into a single non-directed distance measure f as,

$$f(d(A, B), d(B, A)) = max(d(A, B), d(B, A)) \tag{4.2}$$

Using this f function, we evaluated average Hausdorff distances per test case, as plotted at Fig. 4.3. Evaluation suggests that the error (i.e. Hausdorff distance) for 3D objects is not that different from 2D objects. On average the error is 0.12 units for 2D scenes, and 0.14 units for 3D scenes. Given common object 10 units length, the overall average relative error becomes 0.0128 ($\frac{0.012*7+0.014*5}{12}$), which is not significant. Therefore, we can claim that the system is accurate enough for users to easily represent their ideas in both 2D and 3D.

4.2.2 Efficiency

Similar to effectiveness, the efficiency can also be measured by means of several different aspects, including but not limited to, *task completion time*, *input rate*, *usage patterns*, *communication effort*[?]. To evaluate the efficiency of the system, we collected several task completion time and gesture usage frequency data. While completion times give us an idea of how challenging it is to draw 3D shapes than 2D curves, the gesture usage frequency data give an idea about the existence of irrelevant gestures if there any.

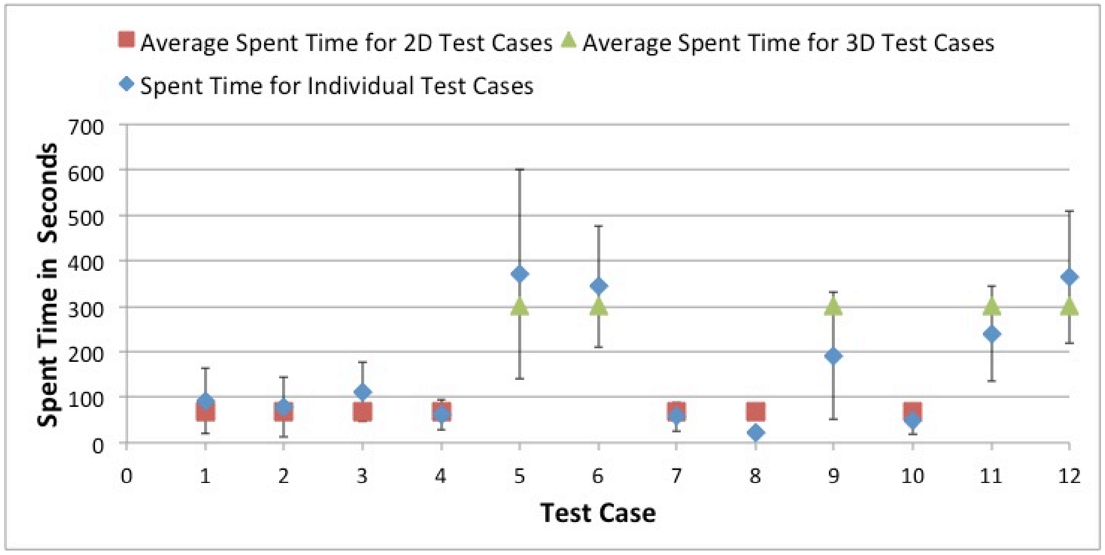


Figure 4.4: Spent time in seconds. Notice the difference in time between 3D test cases (5, 6, 9, 11, 12) vs. 2D test cases (1, 2, 3, 4, 7, 8, 10).

Our *task completion time analysis* revealed that, on average it took 67 seconds to draw a 2D object for all users, whereas it took 301 seconds for a 3D one

(Fig. 4.4). The slight complexity of 3D objects, and the need to adjust drawing plane several times, caused 3D objects to require more time to draw. Even without an analysis of statistical significance, we can easily conclude that 3D object sketching is more difficult than 2D sketches.

The gestural *usage frequency* data we have collected (Appendix A.1) revealed yet another interesting fact. Out of twelve test users, none of them ever used the *editing* gesture to edit a curve. And they seldom used *erasing* gesture as well. Most of the time, users favored simple *undo* button over these two gestures; suggesting that one can at least remove editing gesture from the system. Also the plane selection gesture, where users can extrude a picking ray from current viewpoint was not used often. Creating drawing planes by simply selecting one of the Cartesian planes of an already existing curve outperformed.

4.2.3 Satisfaction

Satisfaction analysis is the last but not least step of our user evaluation. The analysis is usually done by questionnaires used for assessing satisfaction. There are several standard questionnaires in the literature, including *System Usability Scale*(SUS)[15], *Software Usability Measurement Inventory*(SUMI)[**?**], *Questionnaire for User Interaction Satisfaction*(QUIS)[**?**].

Among these several choices, we believe System Usability Scale Survey best suits our intentions (Appendix A.2). System Usability Scale is a simple, ten-item Likert scale giving a global view of subjective assessments of usability[15]. The result scores have a range of 0 to 100. Over twelve test users, our system received 83.75 as a score which can be referred as an “excellent” or “B” grade system, according to Bangor et al.’s work[7]. The individual average scores that the system received for each question can be seen at Table 4.1.

Question	Answer
1. I think that I would like to use this system frequently	3.66
2. I found the system unnecessarily complex	1.16
3. I thought the system was easy to use	4.00
4. I think that I would need the support of a technical person to be able to use this system	1.66
5. I found the various functions in this system were well integrated	4.83
6. I thought there was too much inconsistency in this system	1.16
7. I would imagine that most people would learn to use this system very quickly	4.00
8. I found the system very cumbersome to use	1.66
9. I felt very confident using the system	4.00
10. I needed to learn a lot of things before I could get going with this system	1.33

Table 4.1: System Usability Scale (SUS) survey results. (Strongly Disagree = 1, Strongly Agree = 5)

4.2.3.1 Special Case: Motion Parallax with Face Tracking

Another way of measuring satisfaction is analyzing the *preference* measure that captures which interface users prefer using. For such a measurement, we asked the users to rank the interfaces according to preference. For camera adjustment, we provided two complimentary interfaces: a motion parallax effect with Face Tracking, and a camera adjustment pen gesture with Two-Axis Valuator Trackball. The users had to choose (Appendix A.2) whether they prefer both of them are enabled, or only one. The lowest ranking interface was the choice where only Face Tracking enabled. This is actually an expected phenomena since the pen gesture provides a wider angle of camera movement than motion parallax.

Similarly, people tend to prefer Face Tracking disabled rather than enabled. Since depth cues suggest that motion parallax is an important cue to perceive depth information, we find that this output is surprising. What we believe is that although motion parallax makes it easier to perceive the depth information, it also makes it harder to sketch a curve on the scene as it interferes with the intended sketch gestures. To ease this process, we are pausing the motion parallax effect as soon as the pen gets near the tablet; which was appreciated by our test users.

Chapter 5

Conclusion

We have created a 3D sketching system that can be broadly used by any user, almost like a 3D *paint*. We did push the limits of the system by working with a professional architect to see what the system is capable of, whereas we also tested the system with naive users with a more simplistic way. These evaluations show that our system is an easy to use, yet capable 3D curve sketching interface that requires little learning effort.

While creating such a system, we tried to emphasize the role of depth perception to a great extent; since we believe that for the user to be able to create a pleasing 3D scene, they need to easily visualize the scene. To achieve this goal, we exploited several monocular depth cues, including motion parallax. To our surprise, we concluded that motion parallax is not that effective as we thought it would be.

Throughout this research, we have encountered a number of lessons that will help anyone that may try to develop a similar interface. Several aspects that we implement proved to be essential for such a sketching interface, whereas some of them were not necessary, or even had a negative effect on the overall system. First and foremost, we believe the key of a successful sketching interface is its ease of use. In that regard, we believe getting rid of unnecessary components are important. For instance, our system should drop editing and projectional plane selection gestures.

The emphasis on depth perception has proved to be as important as ease of use. The monocular depth cues are effective on giving the desired depth information and do not interfere with the sketching process of the user. Unfortunately, we can not reach the same conclusion for kinetic depth effect based on motion parallax. The moving nature of the camera with the head position made it harder to keep control of the sketch. Therefore, we believe that new methods to use motion parallax should be investigated in the future.

The hardware specification is also a decisive aspect for the system, since the user experience of sketching is tightly coupled with the input device that is being used. Although we used a pen tablet device rather than a pen display, we can say that a pen display would be suitable for a sketching system. Also it is worth noting that pen devices have several other inputs, such as pressure and the angle of contact, that could be used for augmenting the input further.

Nonetheless, we believe we achieved creating a successful depth perception aware 3D sketching system, which can be used for creating effective 3D curve based models.

Bibliography

[1] A. Alexe, V. Gaildrat, and L. Barthe. Interactive modelling from sketches using spherical implicit functions. In *Proceedings of the 3rd international conference on Computer graphics, virtual reality, visualisation and interaction in Africa*, AFRIGRAPH '04, pages 25–34, New York, NY, USA, 2004. ACM.

[2] K. H. An, D. H. Yoo, S. U. Jung, and M. J. Chung. Robust multi-view face tracking. In *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 1905 – 1910, aug. 2005.

[3] A. Andre and A. Degani. Do you know what mode you're in? an analysis of mode error in everyday things. *Human-automation interaction: Research & Practice, Mahwah, NJ: Lawrence Erlbaum*, pages 19–28, 1997.

[4] R. Bade, F. Ritter, and B. Preim. Usability comparison of mouse-based interaction techniques for predictable 3d rotation. In *Smart Graphics*, pages 924–924. Springer, 2005.

[5] S. Bae, R. Balakrishnan, and K. Singh. Ilovesketch: as-natural-as-possible sketching system for creating 3d curve models. In *Proceedings of the 21st annual ACM symposium on User interface software and technology*, pages 151–160. ACM, 2008.

[6] S. Bae, R. Balakrishnan, and K. Singh. Everybodylovessketch: 3d sketching for a broader audience. In *Proceedings of the 22nd annual ACM symposium on User interface software and technology*, pages 59–68. ACM, 2009.

[7] A. Bangor, J. Miller, et al. Determining what individual sus scores mean: Adding an adjective rating scale. *Journal of usability studies*, 4(3):114–123, 2009.

[8] R. Bartels and F. Samavati. Reversing subdivision rules: Local linear conditions and observations on inner products. *Journal of Computational and Applied Mathematics*, 119(1):29–67, 2000.

[9] T. Baudel. A mark-based interaction paradigm for free-hand drawing. In *Proceedings of the 7th annual ACM symposium on User interface software and technology*, pages 185–192. ACM, 1994.

[10] P. Bloch. Seeking the ideal form: product design and consumer response. *The Journal of Marketing*, pages 16–29, 1995.

[11] D. Bourguignon, M. Cani, and G. Drettakis. Drawing for illustration and annotation in 3d. In *Computer Graphics Forum*, volume 20, pages 114–123. Wiley Online Library, 2001.

[12] M. F. BRADSHAW and B. J. ROGERS. The interaction of binocular disparity and motion parallax in the computation of depth. *Vision Research*, 36(21):3457 – 3468, 1996.

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

[14] G. R. Bradski, S. Clara, and I. Corporation. Computer vision face tracking for use in a perceptual user interface. *Interface*, 2(2):1221, 1998.

[15] J. Brooke. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189:194, 1996.

[16] M. Chen, S. J. Mountford, and A. Sellen. A study in interactive 3-d rotation using 2-d control devices. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '88, pages 121–129, New York, NY, USA, 1988. ACM.

[17] J. J. Cherlin, F. Samavati, M. C. Sousa, and J. A. Jorge. Sketch-based modeling with few strokes. In *Proceedings of the 21st spring conference on Computer graphics*, SCCG '05, pages 137–145, New York, NY, USA, 2005. ACM.

[18] J. Cohen, L. Markosian, R. Zeleznik, J. Hughes, and R. Barzel. An interface for sketching 3d curves. In *Proceedings of the 1999 symposium on Interactive 3D graphics*, pages 17–21. ACM, 1999.

[19] D. Comaniciu and P. Meer. Robust analysis of feature spaces: color image segmentation. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 750 –755, jun 1997.

[20] F. C. Crow. Summed-area tables for texture mapping. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '84, pages 207–212, New York, NY, USA, 1984. ACM.

[21] K. Das, P. Diaz-Gutierrez, and M. Gopi. Sketching free-form surfaces using network of curves. In *Proceedings of eurographics workshop on sketch-based interfaces and modeling (SBIM05)*, 2005.

[22] J. Dorsey, S. Xu, G. Smedresman, H. Rushmeier, and L. McMillan. The mental canvas: A tool for conceptual architectural design and analysis. In *Proceedings of the 15th Pacific Conference on Computer Graphics and Applications*, PG '07, pages 201–210, Washington, DC, USA, 2007. IEEE Computer Society.

[23] M. Dubuisson and A. Jain. A modified hausdorff distance for object matching. In *Pattern Recognition, 1994. Vol. 1-Conference A: Computer Vision & Image Processing., Proceedings of the 12th IAPR International Conference on*, volume 1, pages 566–568. IEEE, 1994.

[24] T. Fleisch, F. Rechel, P. Santos, and A. Stork. Constraint stroke-based oversketching for 3d curves. In *Proceedings of eurographics workshop on sketch-based interfaces and modeling (SBIM04)*. Citeseer, 2004.

[25] A. R. François. Camshift tracker design experiments with intel opencv and sai. Technical Report IRIS-04-423, Institute for Robotics and Intelligent Systems, University of Southern California, July 2004.

[26] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the Second European Conference on Computational Learning Theory*, EuroCOLT '95, pages 23–37, London, UK, UK, 1995. Springer-Verlag.

[27] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting, 1998.

[28] B. FRONER. *Stereoscopic 3D Technologies for Accurate Depth Tasks: A Theoretical and Empirical Study.* PhD thesis, Durham University, 2011.

[29] E. B. Goldstein. *Sensation and Perception (with Virtual Lab Manual CD-ROM).* Wadsworth Publishing, eigth edition, Feb. 2009.

[30] E. Hjelms and B. K. Low. Face detection: A survey. *Computer Vision and Image Understanding*, 83(3):236 – 274, 2001.

[31] T. Igarashi, S. Matsuoka, and H. Tanaka. Teddy: a sketching interface for 3d freeform design. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 409–416. ACM Press/Addison-Wesley Publishing Co., 1999.

[32] E. Johnston, B. Cumming, and A. Parker. Integration of depth modules: Stereopsis and texture. *Vision Research*, 33(56):813 – 826, 1993.

[33] E. B. Johnston, B. G. Cumming, I. Michael, and S. Landyi. Integration of stereopsis and motion shape cues. *Vision Research*, page 22592275, 1994.

[34] L. Kara and K. Shimada. Construction and modification of 3d geometry using a sketch-based interface. In *Proceedings of eurographics workshop on sketch-based interfaces and modeling (SBIM06)*, 2006.

[35] L. Kara and K. Shimada. Sketch-based 3d-shape creation for industrial styling design. *IEEE Computer Graphics and Applications*, pages 60–71, 2007.

[36] D. R. Koller, M. R. Mine, and S. E. Hudson. Head-tracked orbital viewing: an interaction technique for immersive virtual environments. In *Proceedings of the 9th annual ACM symposium on User interface software and technology*, UIST '96, pages 81–82, New York, NY, USA, 1996. ACM.

[37] R. Manual. Open source computer vision library. *Order A Journal On The Theory Of Ordered Sets And Its Applications*, 100(10):236–7, 2001.

[38] J. Martin. *Technical illustration : materials, methods and techniques / Judy Martin.* Child & Associates, Frenchs Forest, N.S.W. :, 1989.

[39] R. Meir and G. Rätsch. Advanced lectures on machine learning. chapter An introduction to boosting and leveraging, pages 118–183. Springer-Verlag New York, Inc., New York, NY, USA, 2003.

[40] A. Nealen, T. Igarashi, O. Sorkine, and M. Alexa. Fibermesh: designing freeform surfaces with 3d curves. In *ACM Transactions on Graphics (TOG)*, volume 26, page 41. ACM, 2007.

[41] J. Nielsen. *Usability engineering.* Morgan Kaufmann, 1994.

[42] L. Olsen, F. Samavati, M. Sousa, and J. Jorge. Sketch-based modeling: A survey. *Computers & Graphics*, 33(1):85–103, 2009.

[43] L. Piegl. Interactive data interpolation by rational bezier curves. *IEEE Comput. Graph. Appl.*, 7(4):45–58, Apr. 1987.

[44] R. Poppe. Facing scalability: Naming faces in an online social network. *Pattern Recognition*, 45(6):2335 – 2347, 2012. ¡ce:title¿Brain Decoding¡/ce:title¿.

[45] N. Qian and D. N. Qian. Binocular disparity and the perception of depth, 1997.

[46] W. F. Reinhart. Depth cueing for visual search and cursor positioning. volume 1457, pages 221–232. SPIE, 1991.

[47] D. F. Rogers. Constrained b-spline curve and surface fitting. *Comput. Aided Des.*, 21(10):641–648, Dec. 1989.

[48] S. Saga. A freehand interface for computer aided drawing systems based on the fuzzy spline curve identifier. In *Systems, Man and Cybernetics, 1995. Intelligent Systems for the 21st Century., IEEE International Conference on*, volume 3, pages 2754 –2759 vol.3, oct 1995.

[49] R. Schmidt, B. Wyvill, M. Sousa, and J. Jorge. Shapeshop: Sketch-based solid modeling with blobtrees. In *ACM SIGGRAPH 2006 Courses*, page 14. ACM, 2006.

[50] T. Sezgin and R. Davis. Scale-space based feature point detection for digital ink. In *ACM SIGGRAPH 2007 courses*, page 36. ACM, 2007.

[51] T. M. Sezgin. Sketch based interfaces: Early processing for sketch understanding. In *Proceedings of PUI-2001. NY.* ACM Press, 2001.

[52] T. M. Sezgin and R. Davis. Scale-space based feature point detection for digital ink. In *ACM SIGGRAPH 2007 courses*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[53] G. Taubin. Curve and surface smoothing without shrinkage. In *Proceedings of the Fifth International Conference on Computer Vision*, ICCV '95, pages 852–, Washington, DC, USA, 1995. IEEE Computer Society.

[54] G. Taubin. Curve and surface smoothing without shrinkage. In *Computer Vision, 1995. Proceedings., Fifth International Conference on*, pages 852–857. IEEE, 1995.

[55] S. Tsang, R. Balakrishnan, K. Singh, and A. Ranjan. A suggestive interface for image guided 3d sketching. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 591–598. ACM, 2004.

[56] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511 – I–518 vol.1, 2001.

[57] M.-H. Yang, D. Kriegman, and N. Ahuja. Detecting faces in images: a survey. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(1):34 –58, jan 2002.

[58] M. J. Young, M. S. Landy, and L. T. Maloney. A perturbation analysis of depth perception from combinations of texture and motion cues. *VISION RESEARCH*, 33(18):2685–2696, 1993.

[59] R. Zeleznik, K. Herndon, and J. Hughes. Sketch: An interface for sketching 3d scenes. 1996.

[60] C. Zhang and Z. Zhang. A survey of recent advances in face detection. *Learning*, (June):17, 2010.

Appendix A

Data

A.1 Sample Usage Data Collected for Objective User Evaluation

```
[    mHausdorffDistance: 0.224159,
     mTotalTime: 47324, mIdleTime: 23489,
     mNumGestures: 45 (
          [ mType: Hover ,mDuration: 873 ]
          [ mType: Navigation ,mDuration: 363 ]
          [ mType: Hover ,mDuration: 770 ]
          [ mType: Picking ,mDuration: 135 ]
          [ mType: Hover ,mDuration: 475 ]
          [ mType: Navigation ,mDuration: 558 ]
          [ mType: Hover ,mDuration: 1659 ]
          [ mType: Drawing ,mDuration: 1276 ]
          [ mType: Hover ,mDuration: 786 ]
          [ mType: Undo ,mDuration: 0 ]
          [ mType: Hover ,mDuration: 1362 ]
          [ mType: Drawing ,mDuration: 2072 ]
          [ mType: Hover ,mDuration: 378 ]
```

```
        [ mType: Undo ,mDuration: 0 ]
        [ mType: Hover ,mDuration: 738 ]
        [ mType: ToggleSymmetry ,mDuration: 0 ]
        [ mType: Hover ,mDuration: 1373 ]
        [ mType: Drawing ,mDuration: 4462 ]
        [ mType: Hover ,mDuration: 1036 ]
        [ mType: Drawing ,mDuration: 2207 ]
        [ mType: Hover ,mDuration: 1278 ]
        [ mType: Undo ,mDuration: 0 ]
        [ mType: Hover ,mDuration: 494 ]
        [ mType: Undo ,mDuration: 0 ]
        [ mType: Hover ,mDuration: 1421 ]
        [ mType: Drawing ,mDuration: 920 ]
        [ mType: Hover ,mDuration: 110 ]
        [ mType: Undo ,mDuration: 0 ]
        [ mType: Hover ,mDuration: 489 ]
        [ mType: Undo ,mDuration: 0 ]
        [ mType: Hover ,mDuration: 470 ]
        [ mType: Drawing ,mDuration: 3333 ]
        [ mType: Hover ,mDuration: 1383 ]
        [ mType: Drawing ,mDuration: 1876 ]
        [ mType: Hover ,mDuration: 544 ]
        [ mType: Undo ,mDuration: 0 ]
        [ mType: Hover ,mDuration: 533 ]
        [ mType: Undo ,mDuration: 0 ]
        [ mType: Hover ,mDuration: 999 ]
        [ mType: Drawing ,mDuration: 3032 ]
        [ mType: Hover ,mDuration: 3816 ]
        [ mType: Picking ,mDuration: 169 ]
        [ mType: Hover ,mDuration: 2848 ]
        [ mType: Eraser ,mDuration: 3086 ]
        [ mType: Hover ,mDuration: 0 ]
    )
```

]

A.2 Survey Questions for Subjective User Evaluation

System Usability Scale

		Strongly Disagree				Strongly Agree
1.	I think that I would like to use this system frequently	1	2	3	4	5
2.	I found the system unnecessarily complex	1	2	3	4	5
3.	I thought the system was easy to use	1	2	3	4	5
4.	I think that I would need the support of a technical person to be able to use this system	1	2	3	4	5
5.	I found the various functions in this system were well integrated	1	2	3	4	5
6.	I thought there was too much inconsistency in this system	1	2	3	4	5
7.	I would imagine that most people would learn to use this system very quickly	1	2	3	4	5
8.	I found the system very cumbersome to use	1	2	3	4	5
9.	I felt very confident using the system	1	2	3	4	5
10.	I needed to learn a lot of things before I could get going with this system	1	2	3	4	5

System Component Ranking

Rank (1-3) below set of components according to their success for the task.

1. Skecth Correction:

Undo Button ☐ Erasing Gesture ☐ Editing Gesture ☐

2. Scene Navigation:

Motion Parallax Disabled ☐ Motion Parallax w/ Pause on Draw ☐ Motion Parallax Enabled ☐

A.3 Sample Sketch Data that Represents a Scene

```
[   mDrawingPlane: [mNormal: [0 0 1], mPoint: [0 4.5 0]],
    mMax: 4,
    mCurves: 2
    (
        [   mPlane: [mNormal: [0 0 1], mPoint: [0 4.5 0]],
            mColor: [0 0 0],
            mId: 3,
            mSymmetricCurveId: −1,
            mPoints: 28
            (
                [−3.81767 5.96285 0]
                [−3.73841 5.95997 0]
                [−3.57988 5.95421 0]
                [−3.3421 5.94557 0]
                [−3.02504 5.93404 0]
                [−2.6767 5.92864 0]
                [−2.29706 5.92935 0]
                [−1.88613 5.93618 0]
                [−1.44391 5.94914 0]
                [−1.00124 5.96193 0]
                [−0.558102 5.97457 0]
                [−0.114512 5.98706 0]
                [0.329536 5.99938 0]
                [0.774033 6.00597 − 7.70191e−11]
                [1.21898 6.00682 − 2.31057e−10]
                [1.66438 6.00193 − 4.62114e−10]
                [2.11022 5.9913 − 7.70191e−10]
                [2.51806 5.98467 − 9.24246e−10]
                [2.8879 5.98202 − 9.24279e−10]
                [3.21973 5.98337 − 7.70292e−10]
                [3.51355  5.98871 - 4.62283e-10]
                [3.73619  5.99257 - 2.31259e-10]
                [3.88764  5.99495 - 7.72209e-11]
                [3.96791  5.99584 - 1.68212e-13]
                [3.977  5.99526 - 1.00927e-13]
                [3.98381  5.99482 - 5.04637e-14]
                [3.98836  5.99453 - 1.68212e-14]
                [3.99063  5.99438  0]
        )
    ]
    [   mPlane: [mNormal: [0  0  1], mPoint: [0  4.5  0]],
        mColor: [0  0  0], mId: 4,
        mSymmetricCurveId: -1,
        mPoints: 28
        (
                [-3.92855  6.0178  0]
                [-3.84772  6.00654  9.4739e-11]
                [-3.68607  5.98401  2.84217e-10]
                [-3.44359  5.95021  5.68434e-10]
                [-3.12028  5.90515  9.4739e-10]
                [-2.76407  5.87987  1.13699e-09]
                [-2.37496  5.87438  1.13722e-09]
                [-1.95295  5.88867  9.48104e-10]
                [-1.49803  5.92275  5.69623e-10]
                [-1.03983  5.95055  2.40457e-10]
                [-0.578361  5.97208 - 3.93961e-11]
                [-0.113611  5.98733 - 2.69935e-10]
                [0.354416  5.9963 - 4.51161e-10]
                [0.81712  6.00819 - 6.18936e-10]
                [1.2745  6.02301 - 7.73261e-10]
                [1.72656  6.04075 - 9.14135e-10]
                [2.17329  6.0614 - 1.04156e-09]
                [2.57811  6.06736 - 1.0601e-09]
```

```
                    [2.94101  6.05863 − 9.69758e−10]
                    [3.262  6.03519 − 7.70532e−10]
                    [3.54107  5.99706 − 4.62424e−10]
                    [3.75234  5.96836 − 2.31326e−10]
                    [3.89581  5.94907 − 7.72392e−11]
                    [3.97149  5.9392 − 1.63097e−13]
                    [3.97937  5.93876 − 9.78582e−14]
                    [3.98528  5.93843 − 4.89291e−14]
                    [3.98922  5.93821 − 1.63097e−14]
                    [3.99119  5.9381  0]
               )
          ]
     )
]
```