† Aalborg Universitet
Institut for Elektroniske Systemer
Afdeling for Datalogi
Fredrik Bajers Vej 7E
9220 Aalborg Ø
http://www.cs.auc.dk

‡ Aalborg Universitet
Afdeling for Computer Vision og Medieteknologi
Niels Jernes vej 14
9220 Aalborg Ø
http://www.cvmt.auc.dk

Interaktiv 3D Modellering
Kunstnerisk formgivning i virtuelle miljøer

Projekt periode: 4. februar - 14. juni 2002

Semester: Dat6

Projekt gruppe:
N4-307b

Deltagere:
Morten Flyger †
flyger@cs.auc.dk

Jim Kynde Meyer †
meyer@cs.auc.dk

Kenneth Pedersen †
kepe@cs.auc.dk

Vejledere:
Claus Brøndgaard Madsen ‡
cbm@cvmt.auc.dk

Peter Bøgh Andersen †
pba@cs.auc.dk

Antal kopier: 9

Antal sider: 128

Synposis:

Denne rapport dokumenterer et projekt som omhandler kunstnerisk formgivning i virtuelle miljøer. Vi har udført en analyse af tegningsprincipper, hvordan kunstnere arbejder i den fysiske verden og ved at læse i relateret litteratur. Kunstnerne vi valgte at observere var en keramiker og en glaspuster. Ydermere valgte vi at observere hvordan en 3D-kunstner modellerede nogle af objekterne, fra de fysiske kunstnere, i Maya. Vi fandt ud af, at vi ikke kunne finde et unikt modelleringsparadigme, som kunne dække de ønskede egenskaber i et system til kunstnerisk formgivning. Dette fik os til at designe en platform som kunne inkludere flere paradigmer, som moduler og plug-ins. Endelig valgte vi at inkludere tredje parts software til at håndtere adgang til hardware og den geometriske matematik. På grund af den begrænsede tid vi havde til rådighed, valgte vi kun at implementere en del af det designede system. Vi valgte et modelleringsparadigme kaldet Surface Drawing, da vi fandt frem til at dette kunne fungere som Proof-of-Concept på kunstnerisk formgivning i vores system.

† Aalborg University
Institute of Electronic Systems
Department of Computer Science
Fredrik Bajers Vej 7E
9220 Aalborg Ø
http://www.cs.auc.dk

‡ Aalborg University
Department of Computer Vision and Media Technology
Niels Jernes vej 14
9220 Aalborg Ø
http://www.cvmt.auc.dk

Interactive 3D Modeling
Artistic Shape Design in Immersive Virtual Environments

Project period: February 4. - June 14. 2002

Semester: Dat6

Project group:
N4-307b

Participants:
Morten Flyger †
flyger@cs.auc.dk

Jim Kynde Meyer †
meyer@cs.auc.dk

Kenneth Pedersen †
kepe@cs.auc.dk

Supervisors:
Claus Brøndgaard Madsen ‡
cbm@cvmt.auc.dk

Peter Bøgh Andersen †
pba@cs.auc.dk

Number of copies: 9

Pages: 128

Abstract:

This report documents a project that concerns artistic shape design in virtual environments. We performed an analysis of drawing principles, how artist model in the real world, and finally by studying related literature. The artists we chose to study, were a ceramic artist and a glassblower. Furthermore, we studied a 3D graphic artist model some of the objects, which we saw modeled by the real world artists, in Maya. We realized that we could not identify one unique paradigm, which could cover the features needed for artistic shape design. Realizing this, we designed a platform, that has the ability to include several paradigms, as modules and plug-ins. Furthermore, we included third party software to handle hardware access and geometric mathematics. Due to the limited time period at hand, we decided to implement only a part of the designed system. We chose a modeling paradigm called Surface Drawing, since we decided that it would function as our Proof-of-Concept of artistic shape design with our system.

Preface

This report is written by project group N4-307b on the 8th. semester at the Department of Computer Science at Aalborg University.

It is primarily written for the members of the project group, the supervisors and the censor. Secondarily, it can be of some use for people with interest in modeling in the area of Virtual Reality (VR) (See Appendix B) and the work process of ceramic artists, glassblowers and 3D modeling experts. All the implemented parts of the project (source and compiled files) and this report can be found at:

`http://www.cs.auc.dk/∼flyger/P8/`

Throughout the report the project group's citations are indicated with [ ] containing a label, that uniquely identifies a citation, which is described in the bibliography. The source code is written in `monospace` and figures are numerated consecutively. Everywhere in the report, where he or his is used, the project group means he/she or his/hers. Everywhere we or our is used it refers to the project group and its activities.

We would like to thank Steven Schkolne at Caltech University for being very helpful and for providing us with source code regarding Surface Drawing. Furthermore, we would like to thank Peter Skotte for producing 3D objects in Maya as 3D modeling expert, and finally Lange Handicraft and Lene Højlund Glassblowing for giving us the necessary knowledge of their workprocesses and their use of tools.

Aalborg University, June 14th 2002.

Morten Flyger   Jim Kynde Meyer   Kenneth Pedersen

Contents

1 Introduction

Computer graphics and geometric modeling is a huge research area within the computer science community, and as such great progress has been made. Computer systems aid in the process of shape design in a number of industries, including the traditional engineering disciplines such as automobile and aeroplane design, and in recent years also in the entertainment industries and for industrial design.

As a result, the state of the art in computerized 3D modeling systems can be used for creating a wide collection of complex surfaces, when in the hands of expert users with extensive experience. Although these systems, that are often based on splines [B86], have proven their worth, they still lack properties such as creative freedom and artistic expression. One of the major problems with spline-based systems, is the fact that users are forced to think in terms of the underlying mathematical structure at the beginning of the modeling process, in order to represent their shape in an affective way. Investigating alternatives for a shape during the modeling process, will require the user to create dramatic changes in the placement of the spline-based patches, and in extreme cases, starting from scratch seems to be the best alternative [SS99].

Seeking creative freedom, artists often turn away from the complex 3D modeling applications, and instead prefer simple tools such as a piece of paper and a pencil as well as physical media like clay and glass for artistic expression. Even though the pencil is a very simple tool, it is an extraordinarily effective way of expressing intricate shapes directly, since it provides a close connection between an artist's perception and action, and the forms the pencil produces [SS99]. The lack of intuitiveness and creative freedom in the traditional computerized modeling tools results in the artists using them for specification rather than creation in the conceptual phase. Our intention with this report is to document a series of studies, that will close the gap between conceptualization and specification using computerized 3D modeling applications, in order to provide artists with new ways of expression and creative freedom.

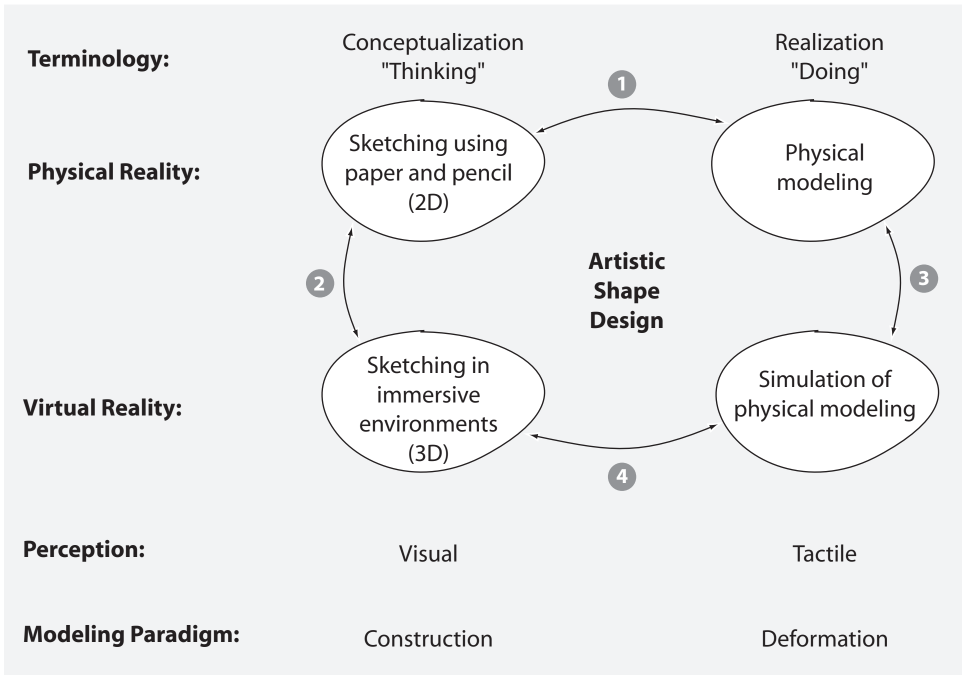


Figure 1.1: A view of artistic shape design, that forms the problem context of this report

Figure 1.1 illustrates the problem context for our studies, namely "Artistic Shape Design" (ASD). Sketching using paper and pencil, and physical modeling with materials such as clay form the basis for the tools used by artists in the real world, which we will refer to as physical reality. When moving towards computerized modeling systems, we find two somewhat competing approaches, with rather different ideas and ways to think about modeling. One, sketching in immersive environments, seeks to expand the traditional paper and pencil activity, which is inherently two-dimensional, to be a three-dimensional tool, that directly links the creation of shapes in their full dimension with the expressive power of the pencil. The second approach, simulation of physical modeling, is based on the idea of recreating the feeling of modeling in the physical world, utilizing mathematical models for material properties such as elasticity and plasticity. We will refer to these two approaches as construction versus deformation, in the sense that they represent two different categories of modeling paradigms.

An interesting observation is that one of the major shortcomings in traditional modeling applications, is their weak usage of perception, that is a very powerful mechanism for learning and understanding. The use of perception is much more visible in the tools chosen by artists, and a crucial part of their design process. To understand this, we need to understand what perception is, and how it affects the artist. According to [Dic02], the definition of perception is as follows:

> *perception n.*
>
> 1. The process, act, or faculty of perceiving.
> 2. The effect or product of perceiving.
> 3. <u>*Psychology.*</u>
>   (a) Recognition and interpretation of **sensory stimuli** based chiefly on memory.
>   (b) The neurological processes by which such **recognition** and interpretation are effected.
> 4. (a) Insight, **intuition**, or knowledge gained by perceiving.
>   (b) The capacity for such insight.

The traditional modeling applications fail to utilize these mechanisms, possibly because of their heritage from earlier systems, that focused mainly on mathematical representation rather than interaction. With regards to the use of sensory stimuli, the most evident shortcomings include the use of two-dimensional visualization and interaction devices with worlds and objects that are inherently three-dimensional. Secondly, users can not recognize shapes for what they are, but are forced to think in terms of control points, patches and polygons, that however clever they may be in the mathematical sense, are counter-intuitive and difficult to understand [SPSa].

With tools such as the paper and pencil, artists can think directly in terms of the shape they are trying to express, and their movements of the pencil is directly linked with their perception of the shape. In this case their primary means of perception is visual information. Physical modeling uses tactile response as an important means of perception, allowing the artist to understand the shape of the object he is working on. Although visual information is also useful, the feeling of the material is essential, since deformation is difficult to control without tactile response.

As powerful and intuitive sketching and physical modeling may be, they still leave things to desire for the artist. This leads us to investigate how the flexibility and advantages of digital representation and modeling, can be combined with the intuitive and powerful sketching and physical modeling, without introducing the shortcomings of the traditional modeling applications. We believe, that an application that exhibits these properties can be useful in many contexts, including the role of a support tool for artists, but also as the primary tool for the creation of virtual art, that creates new possibilities, such as viewing and collaboration of art across physical boundaries. Another use of such an application for artistic shape design, is for entertainment purposes much like people enjoy sketching, painting, and using physical media like clay. Finally, this kind of application has potential as a tool for early conceptualization in the entertainment industries, eg. for game- and character design and movies, as well as in the early phases of industrial design, for rapid evaluation and creation of ideas and shapes.

One of the reasons for moving from physical reality towards virtual reality and immersive environments is the dimensional gap between the paper and pencil, and physical modeling, as illustrated by arrow number 1 in Figure 1.1. Artists can quickly sketch new ideas, but the paper and pencil somehow limits true exploration of three-dimensional form, as it is restricted to two dimensions. Physical modeling is inherently three-dimensional, but the modeling is governed by the laws of physics, that restrict the creative freedom of the artist, setting the boundaries for what is, and is not, possible to do.

As we can control how we model the laws of physics in an immersive environment, we can literally suspend things like gravity, and even create new objects from thin air. It is our belief, that this control can provide artists with previously unseen degrees of creative freedom and artistic expression. However, this vision introduces several challenges, technical as well as non-technical, before this goal is reached. Consider the arrow marked as number 2 in Figure 1.1, which represents the challenges of expanding the traditional two-dimensional sketching to three dimensions. In this case, questions like "How do we include a third dimension without negating the benefits of sketching in two dimensions?", and "Is there an additional need of providing the artist with tools, that help and guide the artist in using the extra degree of freedom?", come to mind.

Moving physical modeling from physical reality to virtual reality, as illustrated by arrow number 3 in Figure 1.1, also poses a number of challenges. Ideally an artist should be unable to differentiate between the simulation running on a computer system, and modeling in physical reality, for him to work in the same intuitive way with digital materials. This gap is no longer a conceptual and dimensional one, as in the case of sketching, but instead a matter of correct representation and behavior of the materials modeled with the system. We believe, that the greatest challenges in this context will be to devise mathematical models that are suitable for interactive real time purposes, combined with the engineering of advanced hardware technologies for providing haptic feedback, eg. the feeling of touching and deforming an object, since this is an important means of perception in physical modeling. Lastly, arrow number 4 in Figure 1.1 denotes a link between the construction and deformation paradigms, in the sense that expanding artistic shape design to immersive environments may reveal completely new hybrid modeling paradigms, that seek to combine the best of both paradigms.

As stated earlier in this chapter, extending artistic shape design to immersive virtual environments sets forward a number of technical challenges within the area of computer science and hardware engineering. As we see it, the primary non-technical challenges include gathering information about the work process of artists as they work in physical reality, be it by sketching or using physical media such as clay. A thorough understanding of these processes will be crucial in the design of interactive modeling applications for artistic shape design, in order to provide the artists with the intuitiveness, creative freedom and artistic expression they have come to expect from their tools. We will acquire this knowledge by a study of drawing principles combined with a series of case studies, in which we will observe artists from different artistic trades and 3D graphic designers as they work with shape design. This learning process, which is illustrated in Figure 1.2 on the next page, forms the basis for the analysis documented in this report.

The remainder of the report is structured as follows. Chapter 2, Drawing Principles, documents the knowledge gained in the study of these principles, that function as a knowledge base that is useful for assessing already existing technical solutions, and for setting forth a number of guidelines that interactive modeling applications for artistic shape design should adhere to. Case Studies, in Chapter 3, describes the case studies we performed, in order to identify how specific groups of people use their hands for creative purposes, and what tools they utilize to achieve certain tasks. Guidelines for Paradigm Selection in Chapter 4 sets forth a number of guidelines for choosing modeling paradigms, that are suitable for artistic shape design. Chapter 5, Related Work, presents an overview of research and literature that is related to the context of our project, and discusses each of these so called modeling paradigms based on the knowledge gained in the previous parts of the analysis. Chapter 6 summarizes upon the Analysis part, and presents

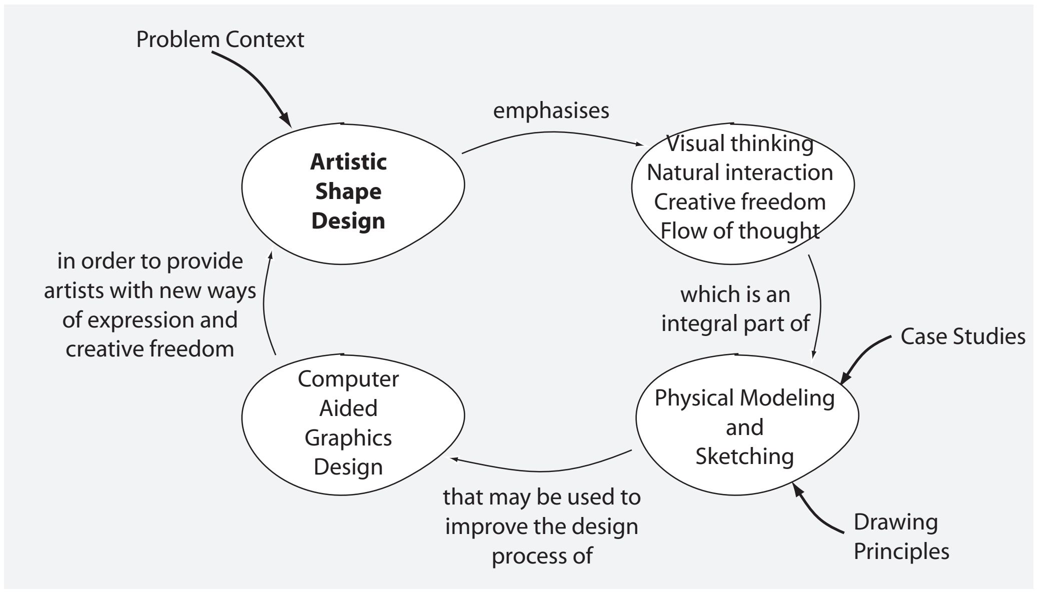

Problem Context
Artistic Shape Design
emphasises
Visual thinking
Natural interaction
Creative freedom
Flow of thought
which is an integral part of
Case Studies
Physical Modeling and Sketching
Drawing Principles
that may be used to improve the design process of
Computer Aided Graphics Design
in order to provide artists with new ways of expression and creative freedom

Figure 1.2: The problem definition for artistic shape design, that forms the foundation for the analysis in this report

further elaborations of the topics discussed in that part.

Design in Chapter 7 describes the design of a platform for an interactive modeling application for artistic shape design, that takes into consideration the lessons learned from the case studies and our investigation of drawing principles, as well as the guidelines for such an application, which we present in this report. The chapter also describes the choice of a modeling paradigm for artistic shape design, that runs on the ASD platform, which we present later in this report. Implementation in Chapter 8 documents the implementation performed during this semester, and discusses the results achieved and the problems encountered. Also, this chapter presents the capabilities of the application we implemented, and the hardware setup it utilizes. Chapter 9 summarize upon the Design & Implementation part, and elaborates upon the topics discussed in that part.

The final part, starting with Future Work in Chapter 10, provides an overview of the future work we identified, followed by Conclusion in Chapter 11, that concludes on the research results in this report.

Part 1 Analysis

This part contains the analysis performed. As mentioned in the Introduction, we found it relevant to study drawing principles to learn how drawing and sketching was carried out in 2D. Furthermore, we did some fieldwork to study how real world artists model. Here we studied both artists working with ceramics and glass, but also a professional 3D computer modeller. These studies are discussed and compared before we set some guidelines for how the remaining part of the project should elapse. Finally, we studied related work to identify existing paradigms which fulfill the guidelines and our intentions with this project.

2 Drawing Principles

To learn about the principles of drawing, we studied literature in that area. The following is a description of some of the principles we learned about in [CJ98]. This book starts with the introduction:

> ”Drawing is the process or technique of representing something - an object, a scene, or an idea - by making lines on a surface.”

We wish to extend this definition to drawing in 3D, and in this context we find it relevant to look closer at the three following topics from this book.

2.1 The Drawing Process

Even with electronic medias and augmented traditional drawing methods, drawing remains a cognitive process that involves perceptive seeing and visual thinking. All drawing is an interactive process of seeing, imagining and representing images, as illustrated on Figure 2.1 on the following page.

Seeing is the use of vision. This sense makes drawing possible, and drawing supports seeing. Imagining is the processing of the data received by the eye. The mind searches for a structure and meaning. The minds eye creates the images we see, which are the images we are trying to draw. The use of visual thought raises drawing beyond a manual skill. Representing is the making of marks, for instance, on a surface to graphically represent the image in the minds eye. This graphically representation then speaks to the eye.

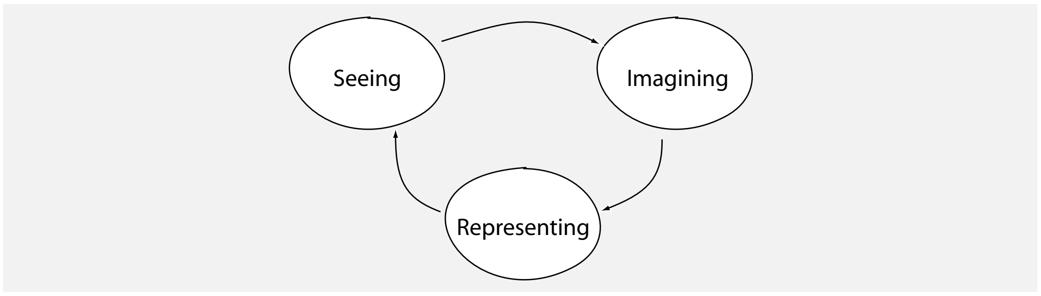


Figure 2.1: The interactive process of seeing, imagining and representing images

2.2 Analytical Drawing and Proportion

Unlike contour drawing, where each area of the drawing is finished before moving on, analytical drawing proceeds from the whole drawing to the subordinate parts and finally the details. To work with analytical drawing is often preferred in preference to contour drawing, because analytical drawing typically results in a better structure and control of proportions, since the whole drawing is taken in consideration already in the first strokes. The first step of analytical drawing is lightly drawing of lines to establish a transparent volumetric framework for a form. This framework helps in drawing the three-dimensional form. Some additional lines can support in finding for instance the center of the drawing, and the framework can be used for drawing the surface lines. Figure 2.2 on the next page illustrates a bottle drawn from a framework.

Regulating lines is used for locating points in the framework. Approximate lines help the eye seek the correct lines, which can then be corrected. The focus on volume as well as surface helps avoiding flat objects, which is typical when the focus is only on the surface. Before a line is drawn, the endpoints are marked with dots, and the line is practiced by moving the hand along the intended line. Furthermore the pencil is always pulled, never pushed. This means that a right-handed always draws lines from left to right and from top to bottom.

Proportion is explained in [CJ98] as follows:

> ”Proportion is the comparative, proper, or harmonious relation of one part to another or to the whole with respect to magnitude, quantity, or degree. Proportional relations are a matter of ratios, and ratio is the relationship between any two parts of a

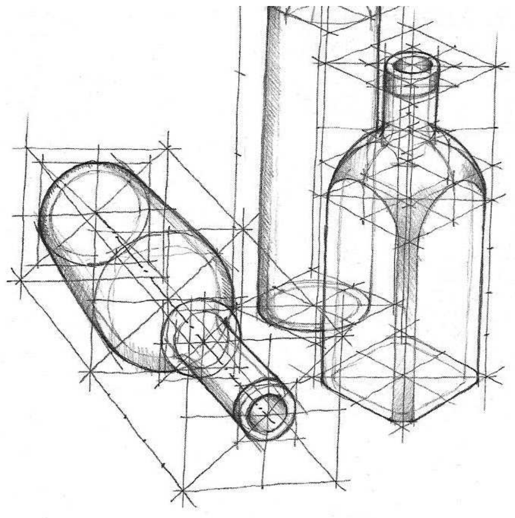


Figure 2.2: Some bottles drawn from frameworks

> whole, or between any part and the whole. In seeing, we should pay attention to the proportional relationships that regulate our perception of size and shape."

In Figure 2.3 on the following page an example of a complex drawing with different ratios is illustrated. The proportions in the drawing are coped with, by finding simple shapes. In this example the simple shapes are squares.

Proportion is also the relative sizes of objects in a drawing, compared to the paper or other objects.

2.3 Building on Geometry

This approach relies on that the object, to be drawn, can be simplified to simple geometry, e.g. cubes, which differs from analytical drawing described in Section 2.2. In analytical drawing, the framework only supports in locating the placement of new lines, the center of objects and keeping the volume proportions right in the object. In the building on geometry the framework is directly used as part of the drawing, and is iteratively adjusted to the final drawing. From the cube other simple geometry models can be derived, as pyramids, cylinders and cones.

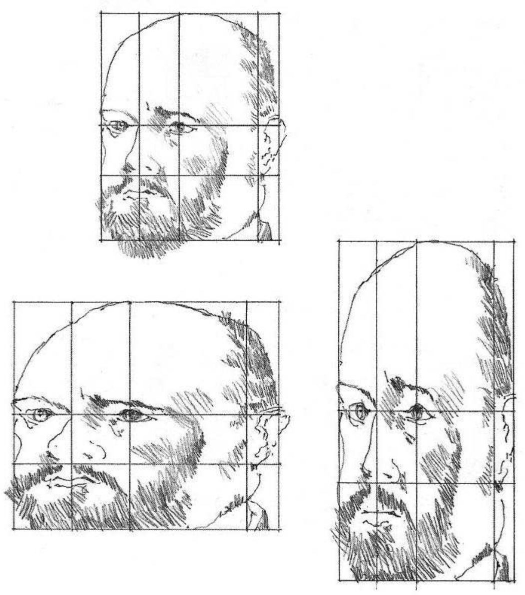


Figure 2.3: A drawing shown with different proportions

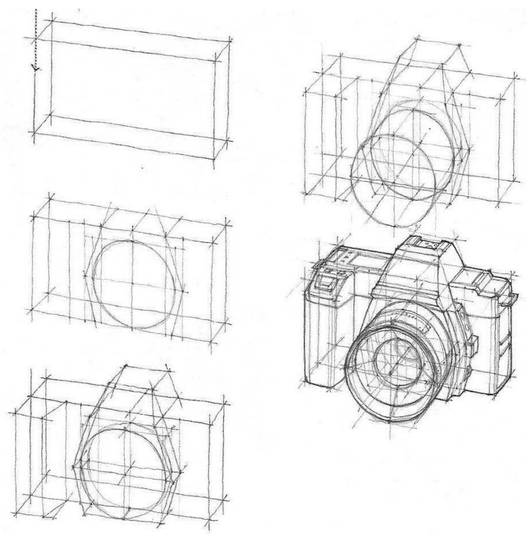


Figure 2.4: A camera built on geometry using additive form

Building on geometry can be done as additive form, subtractive form or complex form. The additive form can be seen in Figure 2.4. A cube can be extended vertically, horizontally and in the depth of the figure. A two-dimensional grid can be added to the figure to subdivide the figure and the extension of the object can be done locally. Dots are reminders of positions, whereas lines represent the vertical and horizontal, and regulate spacing.

The subtractive form is used by starting with a simple form and then carving in finer and finer detail until the desired result is achieved. The complex form is a combination of the additive and the subtractive form.

3 Case Studies

This chapter concerns the case studies we performed, in order to identify how specific groups of people use their hands for creative purposes, and what tools they utilize to achieve certain tasks.

3.1 Overview

To get a broad perspective on the design process of artists, we will contact people that have completely different backgrounds and different ways of working with design. This includes 3D graphic designers that are used to work with computer aided design, e.g. 3D programs like Maya[1], as well as artists without that kind of experience of using computers for design purposes. This selection of people may show us that they have distinctly different ways of transforming their ideas into the final shape.

Interesting results may be gathered by observing how they work with the available tools, in which situations they use what tool and what tasks they have ease or difficulty performing. Something that is easy to do for artists using their hands, may be a very tedious and complex task to perform using a computer system, even for the most experienced users, or vice versa. One of the purposes of observing different people is to identify common and shared methodology and concepts, as well as differences between the different trades.

The real world artists will form the basis for what tasks we will analyze and recreate using a computer system. The main reason for choosing this approach, is that most of these artists, refine and improve the process of creating products such as bowls and candlesticks, until they choose the best

[1]Maya, Alias Wavefront, Silicon Graphics Limited

suited tools and practices for that specific product, based on, among other things, thousands of years traditions in the handicraft. Also, it is more likely that the shape of objects created in the real world, which must adhere to the laws of physics and use real materials, can be recreated using a computer system, than the other way around.

Setting the scope of the interactive modeling application we will design to the artistic trades, enables us to focus on what kind of functionality this group of people require to recreate their products using a computer system, which in turn means that we get a very precise definition of what users actually can do using the application, and what design tasks it is suitable for. Also, setting this scope for the application ensures that the functionality which is included in the application, actually has a purpose, and that the interface is kept as simple as possible. This minimizes a significant problem, which is most evident in systems that provide a very large collection of functionality, namely that selecting an appropriate and suited tool for a specific task is a very complex decision for the user.

We will observe how people from the artistic trades as well as 3D graphic designers work, to gain an understanding of how their design process is structured, and how they use their tools. The questions we wish to answer, which are interesting with regards to modeling in general, navigation in 3D environments, and viewing and understanding the shape of 3D objects, include:

1. Which tools are used? Can they be categorized by type, usage, context and task?
2. What is the role of the hands in different situations? Are they sometimes used as tools themselves, and are different roles assigned to each hand?
3. Does the environment in which they work support them in some way during the design process?
4. What tasks during a design process classify as complex and difficult, and what specific characteristics of the task are the reason for this? Analogous the same could be asked for tasks that classify as easy.

3.2 Selected Cases

We have chosen three areas within arts and crafts, namely ceramics, cold glass work and glassblowing. Together they provide us with a broad perspective

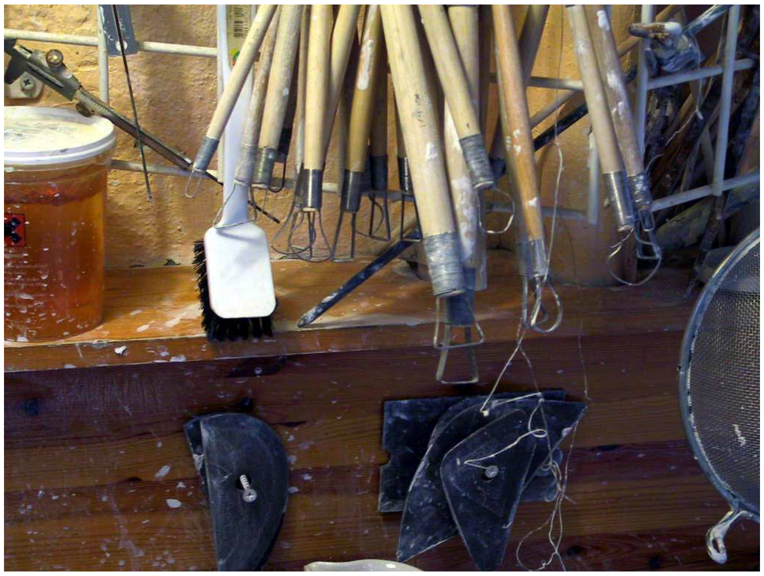


Figure 3.1: The sticks in the upper half of the picture is the trimming tools, and the dark tools at the bottom of the picture are the ribs

of handicraft, since these artists have distinctly different ways of working with their material. Furthermore we look at how a professional 3D computer designer uses the computer to obtain shapes like those the artists created. Technical terms used in this section are based on a Dictionary of Pottery Words[WP98]

3.2.1 Lange Handicraft

We visited Lange Handicraft[2], which is a ceramic and cold glass workshop. The purpose of this visit was to observe the work processes used executing this handicraft. The ceramic work did not involve many tools, since the artist mainly used his hands for the sculpting. He used a rib for carving the clay, a trimming tool for removing the extra material, as shown in Figure 3.1, a cut-off wire for releasing the finished products from the potter's wheel, as illustrated in Figure 3.2 on the next page, and a bucket of water for cleaning his hands, as depicted in Figure 3.3 on page 19. The clay used by the artists is a mixture of French porcelain and German tile mass. The proportion between the two ingredients determines the behavior of the mixture.

[2]Lange Handicraft, Hjelmerstald 15, 9000 Aalborg

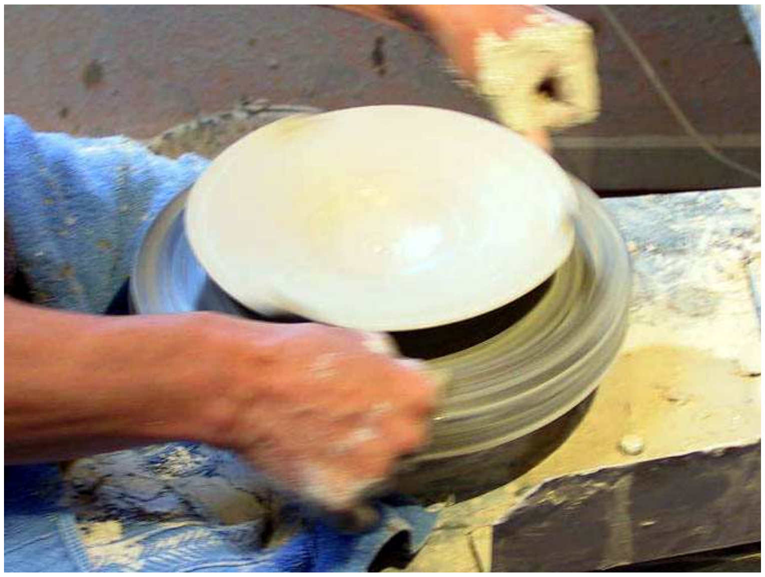


Figure 3.2: The cut-off wire is used for removing the product from the potters wheel

The ashtray on Figure 3.15 on page 27 is created at the potter's wheel. The artist starts using a lump of clay as shown in Figure 3.4 on the facing page. The lump of clay is placed at the potter's wheel and the artist deforms it with his hands as the potter's wheel rotates. The design of the ashtray is done almost with the hands alone. The foot of the astray is shaped with a rib like the ones in Figure 3.1. The turn ups at the edge of the ashtray is simply formed by the artist pressing his thumb against the edge of the ashtray from below. Finally he releases the ashtray from the potter's wheel using a cut-off wire, as shown in Figure 3.2. When the clay has almost dried, the artist finishes the surface of the ashtray with a trimming tool, as depicted in Figure 3.1 on the page before. This action is also performed using a potter's wheel. Finally the astray is glazed and fired.

At Lange Handicraft they do not use computers for sketching and design. This part of the pottery making is done with pen and paper, as illustrated in Figure 3.5 on page 20.

The non-symmetrical products can not be created at the potter's wheel, and are instead created by casting. Figure 3.6 on page 20 shows an example of a mould and a piece of pottery created using this mould.

The other part of the workshop is the cold glass workshop. Here the artists among other things created glass dishes. These were created by placing two

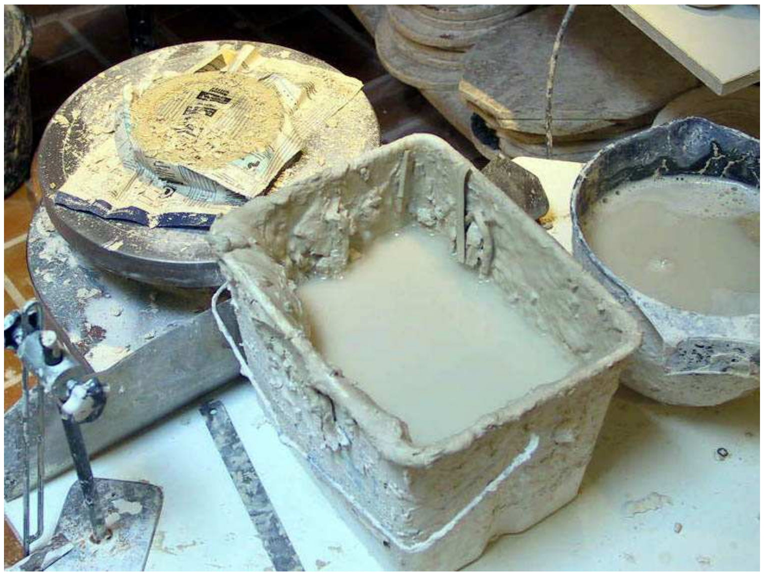


Figure 3.3: A bucket of water for cleaning the hands

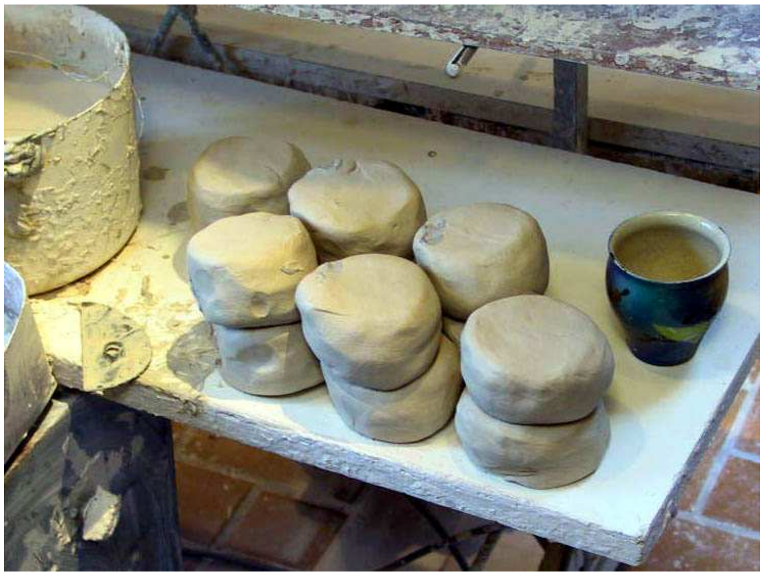


Figure 3.4: The lumps of clay, the artist uses for creating ashtrays

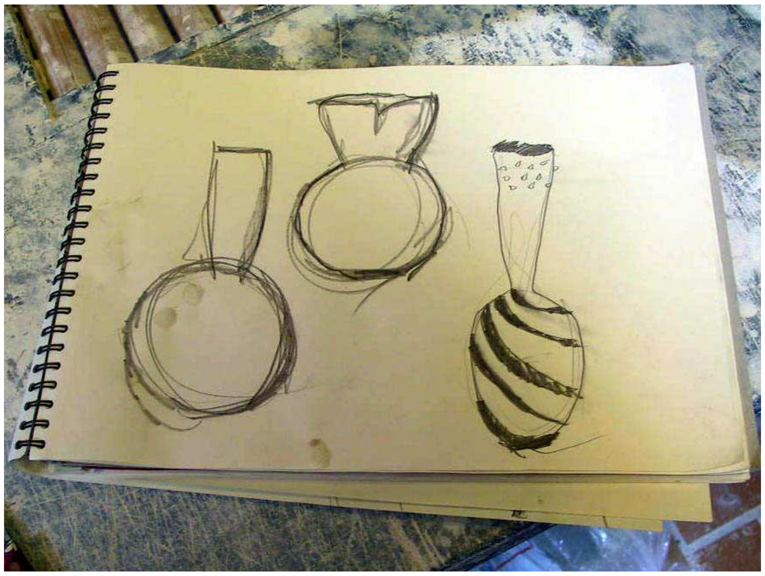


Figure 3.5: The artists at Lange Handicraft design their creations by sketching them with pen and paper

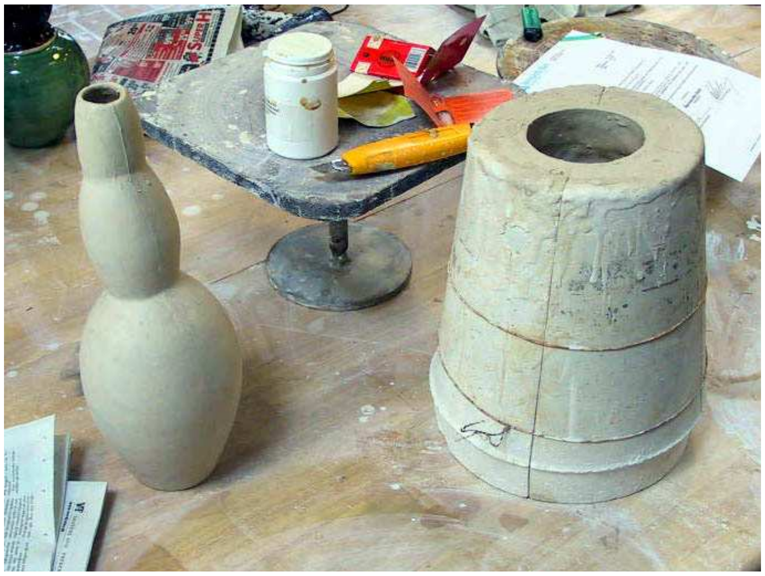


Figure 3.6: At the left the casted product, and at the right the matching mould

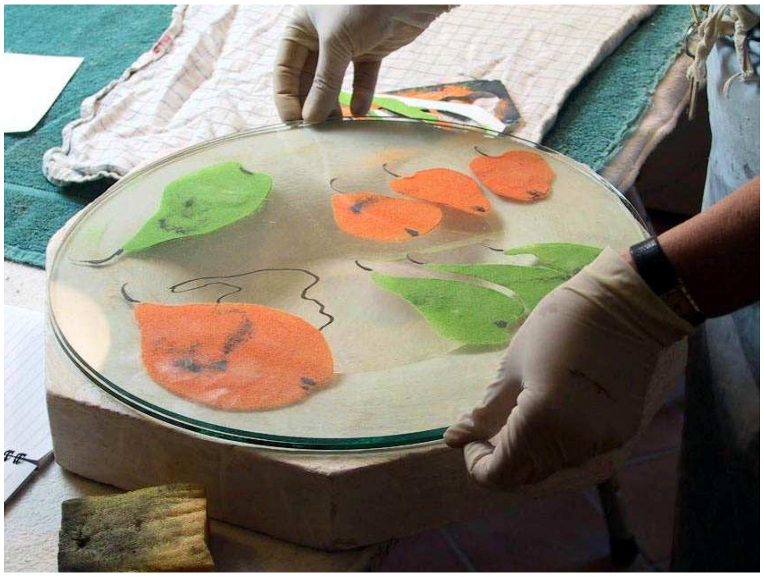


Figure 3.7: Two glass plates with colored powdered glass between on a mould

glass plates on a mould with colored powdered glass between the glass plates, as illustrated in Figure 3.7. The arrangement is then placed in a kiln, where the heated glass is pulled towards the mould by gravity. When it is finished in the kiln, it looks like the dish in Figure 3.8 on the next page.

3.2.2 Lene Højlund Glassblowing

We also visited a glassblower[3] to see another way to work with glass and what differences there are in their choice of tools, if any. The fundamental difference in the way they work with glass at Lange Handicraft and the way it is done at a glassblower, is the temperature of the glass. At Lange Handicraft they perform the work on the glass when it is room temperature, whereas the glassblower warms the glass to about 1000-1100 degrees Celcius. For this reason the glassblowers never touch the glass directly, instead they use different tools. The tool used the most is the blowpipe, as depicted in Figure 3.9 on the following page, which they hold the warm glass with and use to blow air into the glass mass.

When they have placed the warm glass on the tip of the blowpipe they use a big ladle to center the mass on the blowpipe. This is achieved by rotating

[3]Glaspusteriet - Lene Højlund Søndergade 9A 9000 Aalborg

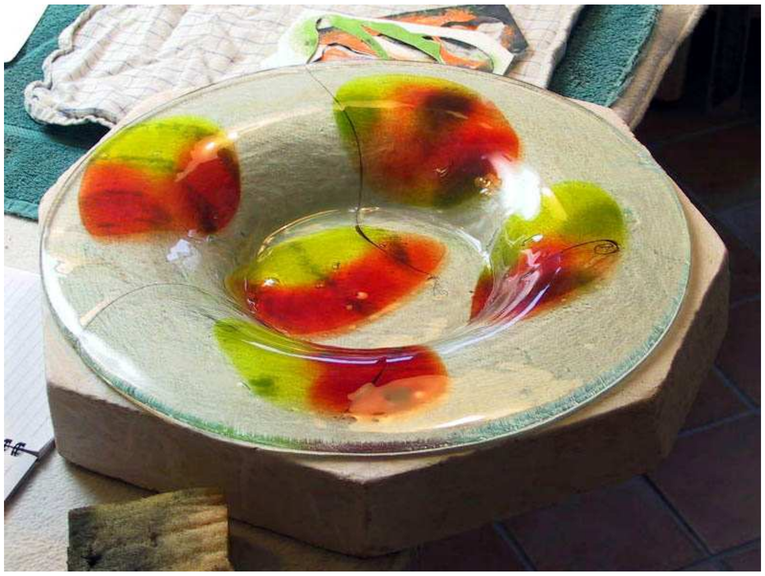


Figure 3.8: The dish created by putting the arrangement from Figure 3.7 in the kiln

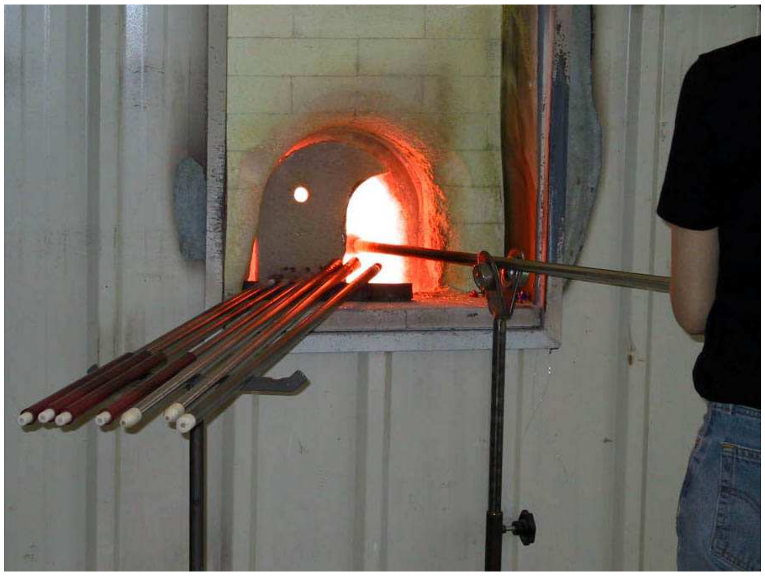


Figure 3.9: The blowpipes used to hold the glass mass and to blow air into the mass

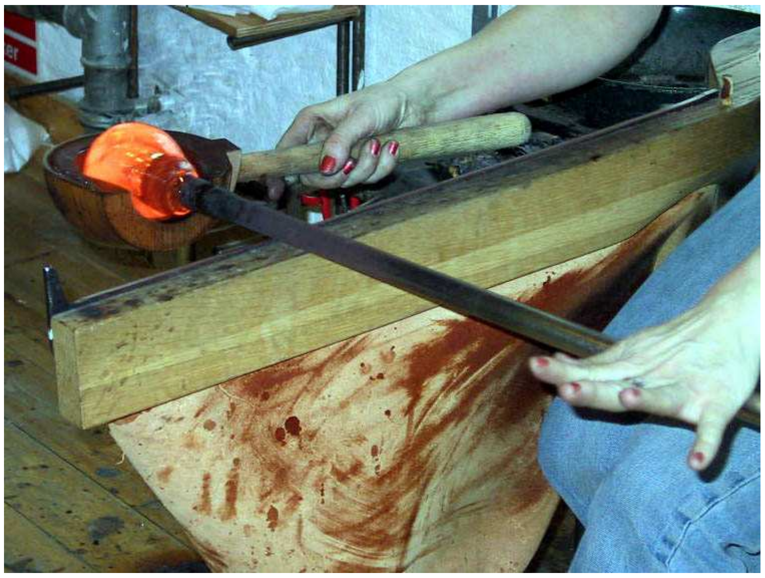


Figure 3.10: The ladle used to center the glass mass on the blowpipe, by rotating the mass in the ladle.

the mass in the ladle, as illustrated in Figure 3.10

As said earlier they can not touch the glass directly due to the temperature. Instead they use a wet newspaper when they have to use the hands to shape the glass, as shown in Figure 3.11 on the following page.

Furthermore, they use a lot of difference pliers for different purposes. Some are for shaping the glass mass, while others are for cutting the mass, as illustrated in Figure 3.12 on the next page. They also use a flat board, for instance to shape the bottom of a glass object. This is illustrated in Figure 3.13 on page 25.

Besides the tools, they utilize the physical properties of the heated glass in combination with how the environment affects their creations. An example of this is the use of gravity to stretch and expand the glass, in ways that can be asymmetric. This is illustrated in Figure 3.14 on page 25, where the glass object is rotated fast resulting in an expansion of the edge and the opening.

3.2.3 3D Graphic Artist

Knowing how artists work with modeling in the physical world, we decided to look at the work process at a computer. For this purpose we persuaded

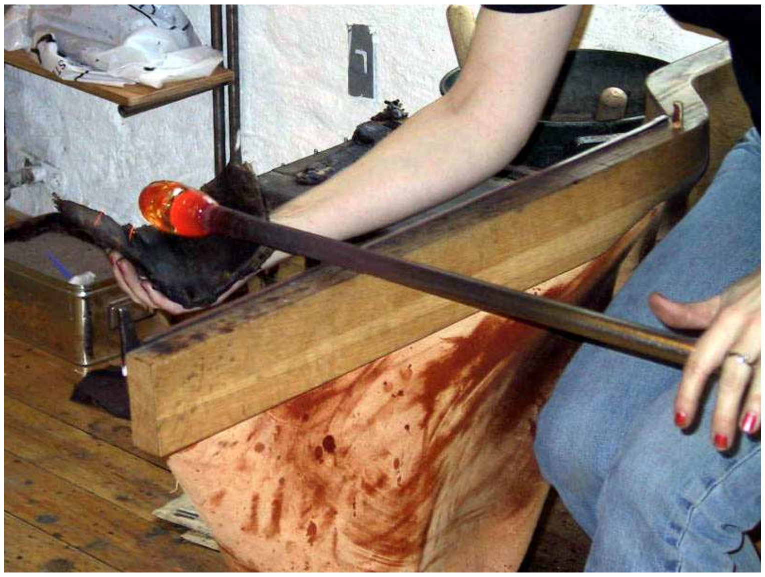


Figure 3.11: A wet newspaper is used to shape the glass when using the hands.

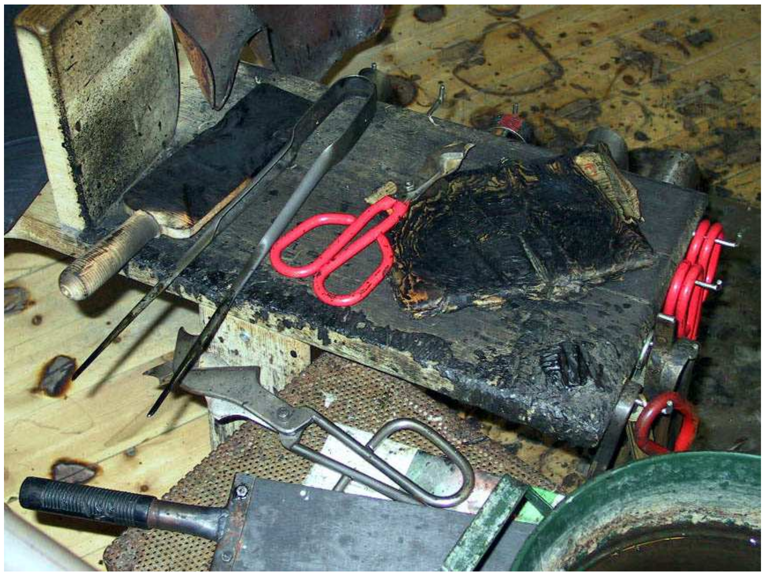


Figure 3.12: A small collection of the tools used in the process of making glass creations.

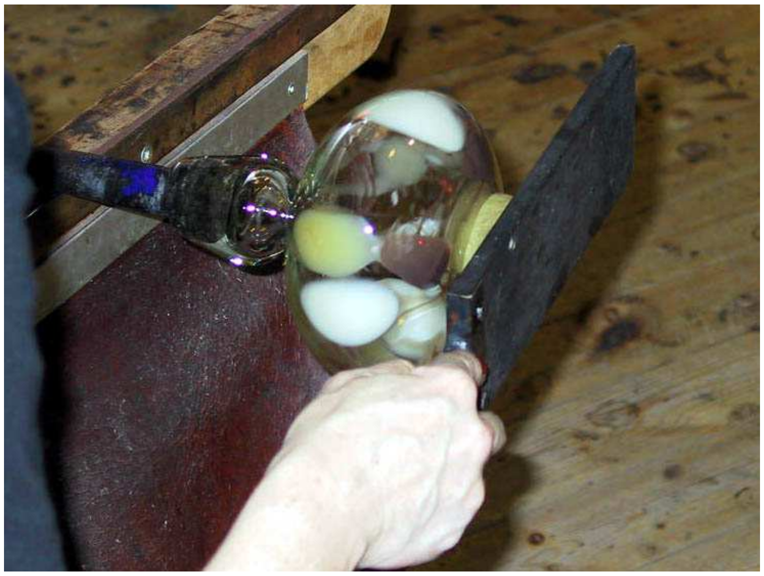


Figure 3.13: A flat board used to shape the bottom of a glass object.

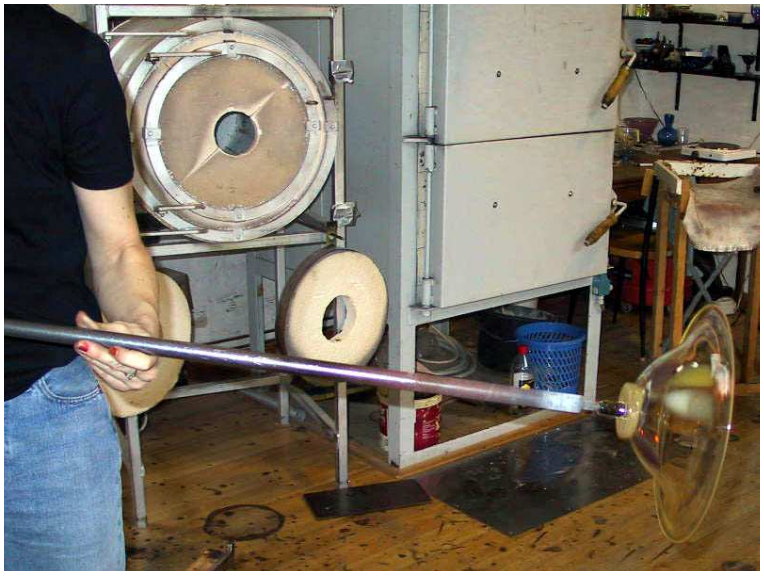


Figure 3.14: A glass object rotated fast to expand it

Peter Skotte[4] to model some of the arts we have seen at Lange Handicraft and at Lene Højlund Glassblowing, in Maya.

We decided to contact Peter Skotte based on his background in 3D graphic design. He has, among other things, been working with the 3D effects in TV commercials. He has a lot of experience using 3D Studio Max[5], Maya[6] and other such applications. Therefore Peter Skotte can be considered an expert in 3D computer artistic graphics, not to be mistaken for a CAD/CAM expert. Furthermore he is currently located close to the project group.

We asked Peter Skotte not to be 100% accurate with size and proportions. The 3D model should just resemble the shape created by the artists. This way we avoided that Peter Skotte felt it was necessary to go into details and use unnecessary time on these.

The tasks we asked him to perform involved creating the shapes shown in Figures 3.15 on the facing page, 3.19 on page 28 and 3.22 on page 30. He succeeded in completing each task with an average of about 10 minutes per figure, resulting in the Figures 3.18 on page 28, 3.21 on page 30 and 3.24 on page 31.

Ashtray

One of the tasks we gave Peter Skotte was to model the ashtray depicted in Figure 3.15. The first thing he did was to draw half of the profile of the ashtray, as shown in Figure 3.16 on the facing page.

Next he revolved the profile to get a shape which resembles the ashtray made by the artist. The result of revolving the profile is shown in Figure 3.17 on the next page.

Finally Peter Skotte had to make the three small deformations along the edge of the ashtray. Those were the hardest thing about this task. He used a different approach for each of the three to illustrate different ways to perform the deformation. Two of them did not give the correct result, whereas the last he made was better, although he made it less wry than the deformations the artist had made. The artist used his thumb which resulted in a deformation that was a little more oblique. The final result of this task can be seen in Figure 3.18 on page 28.

[4]Graphic Designer at VR medialab, Niels Jernes vej 14, 9220 Aalborg Ø
[5]3D Studio Max, Discreet
[6]Maya, Alias Wavefront, Silicon Graphics Limited

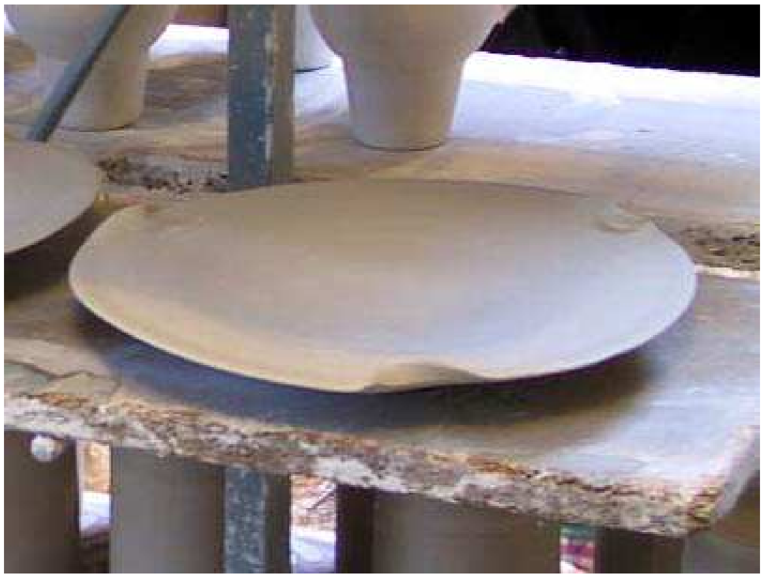


Figure 3.15: Ashtray created at Lange Handicraft

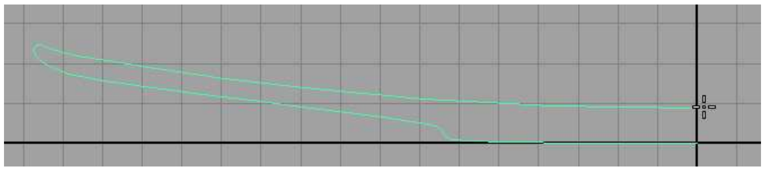


Figure 3.16: Profile of the ashtray created in Maya

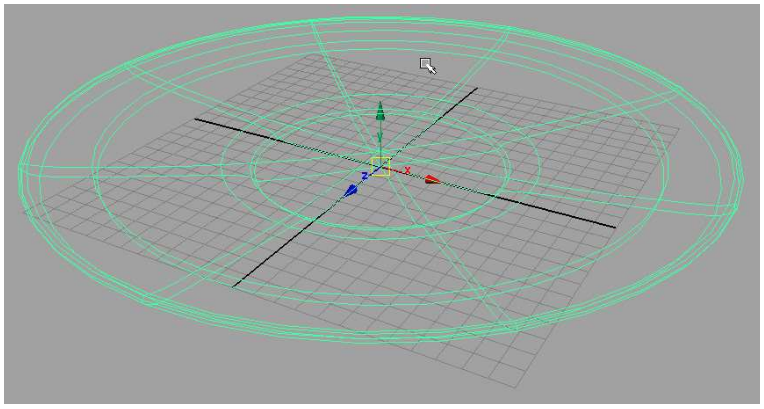


Figure 3.17: The object created by revolving the profile

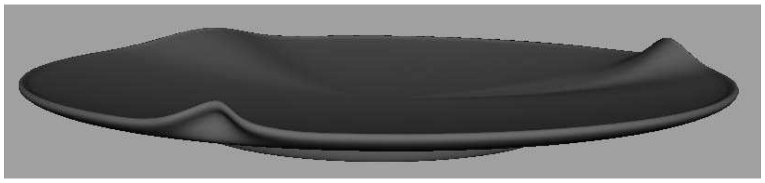


Figure 3.18: The ashtray from Figure 3.15 created in Maya

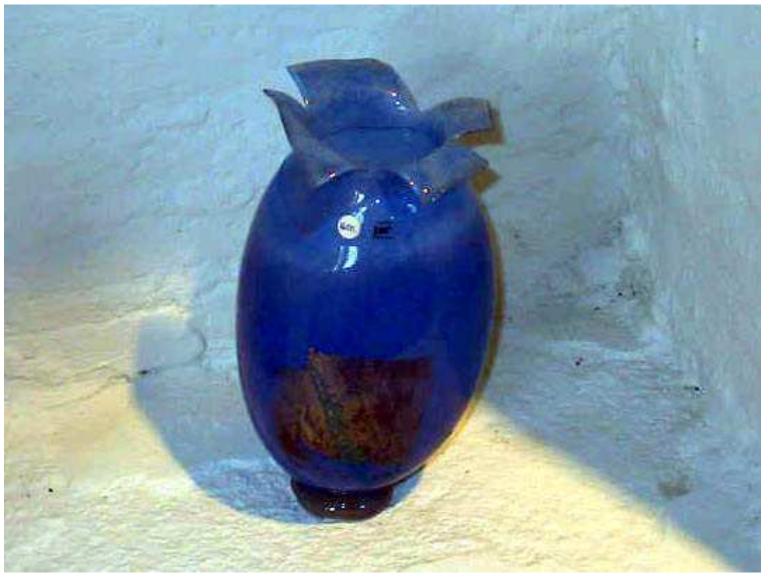


Figure 3.19: A vase created at Lene Højlund Glassblowing

Vase

The second task Peter Skotte solved was to model the vase depicted in Figure 3.19. He also started this task by drawing a profile and revolving it. The foot of the vase was shaped by placing some control points round the edge of the foot, these could now be scaled closer to the center of the vase. This resulted in the flower shaped foot.

The complicated part of creating the vase was the upper part, where the vase is separated in four quarters. This problem was solved by temporarily removing the lower part of the vase, and thereby avoiding the deformation on the upper part in having an effect on the lower part. Furthermore the three of the four parts at the upper part was removed (See Figure 3.20 on the facing page). Then the remaining part of the object were bend outwards from the

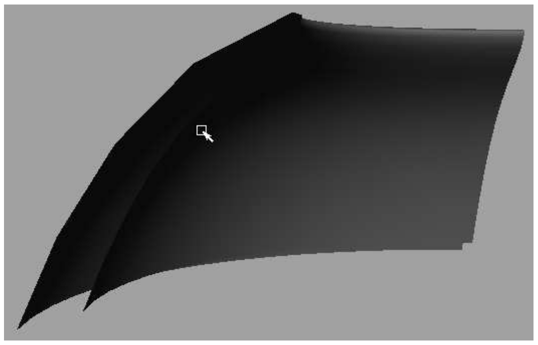


Figure 3.20: A quarter of the upper part of the vase, before the deformation

center of the vase. As the deformation was accomplished satisfactory, the deformed part was copied around the center of the vase and the lower part of the vase was inserted again. This resulted in the object at Figure 3.21 on the next page.

Candlestick

The third task we asked Peter Skotte to make was a candlestick as illustrated in Figure 3.22 on the following page. As in the previous two tasks he first made the profile of the shape, without the bended flat foot, and hereafter he revolved the profile.

After he had revolved the profile he flattened the part of the shape that later would become the foot. Then he inserted a "skeleton" in the center of the candlestick, so it could get the right bend. The first try to make the bend resulted in a distortion of the top part, as shown in Figure 3.23 on page 31.

To cope with this Peter Skotte used "undo" to get back to the point before the insertion of the skeleton and started over. He inserted a new skeleton and bended the candlestick, this time with a better result. The result of the task is shown in Figure 3.24 on page 31.

3.3 Discussion of Concepts

This section discusses the results gained from studying the design process of the 3D graphic artist and the other different artists we selected.

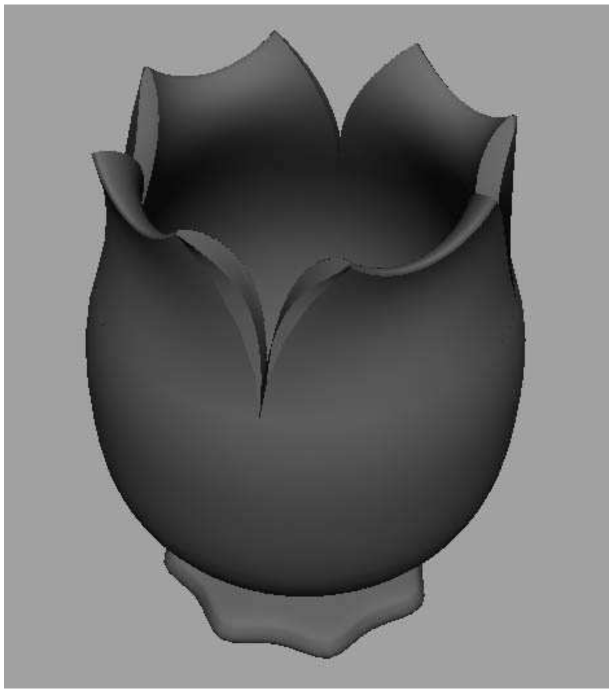


Figure 3.21: The vase from Figure 3.19 created in Maya

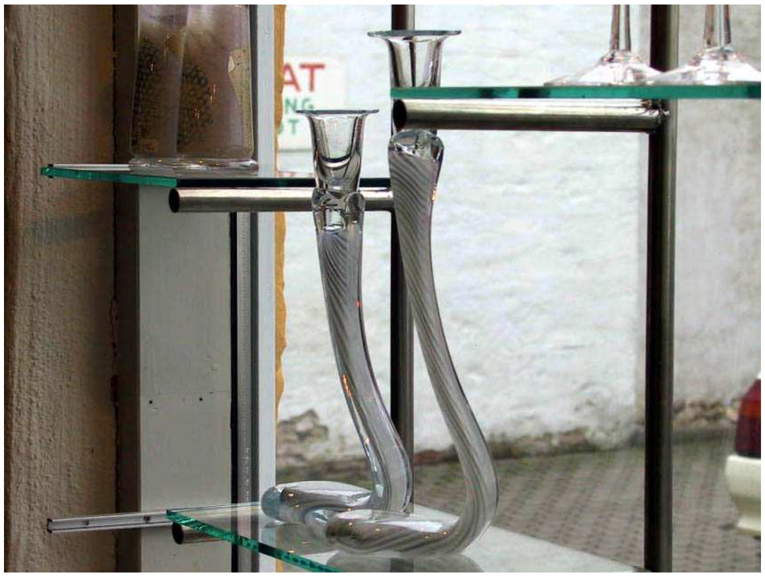


Figure 3.22: Two candlesticks created at Lene Højlund Glassblowing

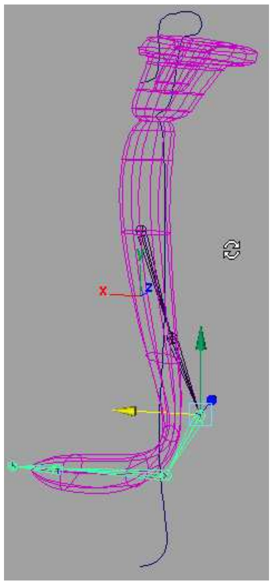


Figure 3.23: A candlestick with distorted top, after an unsuccessful bend of the foot

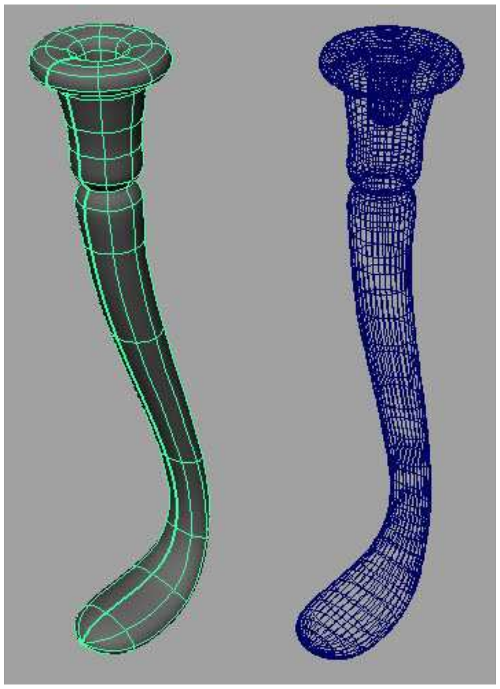


Figure 3.24: The candlesticks from Figure 3.22 created in Maya

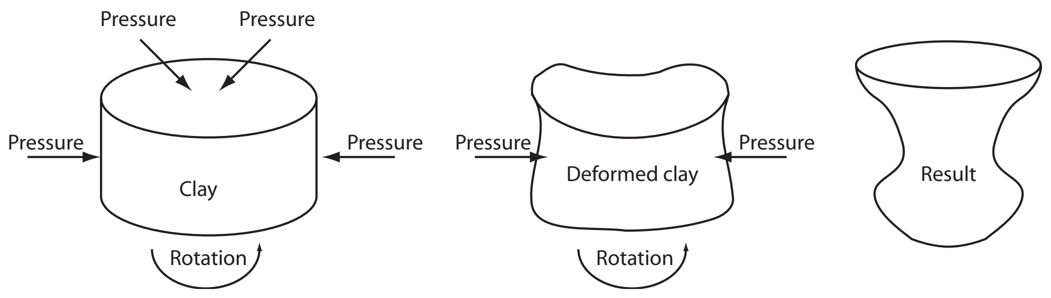


Figure 3.25: Revolve as it is used by ceramic artists and glass blowers is a way of deforming the object

3.3.1 The Revolve Process

In the cases we observed, the most widely used way of working with the initial shape is based on a symmetry axis, and rotating the object to achieve the desired shape. The term for this operation is "Revolve", however the way it is used differs quite a lot from the real world to the computer system. Both the ceramic artist and the glassblower use the revolve operation as a means of deforming already existing objects (Deformation), whereas the 3D graphic artist uses it to create new geometry (Construction).

Figure 3.25 shows how a simple bowl is deformed using a potter's wheel. The original lump of clay is shaped into the final shape, by applying pressure on the sides, while it rotates. Both hands are used to shape the clay, with the thumbs pressing down on top of the clay to create the concave inner shape of the object. At the same time the thumbs are used to counter the pressure applied from the outside by the remaining fingers. Glassblowers work in a somewhat similar way, only they use a collection of tools for shaping the glass while they rotate it using the blowpipe. So in both these cases revolve is achieved by combining pressure and rotation.

This varies a lot from the use of Revolve on a computer system. Here, the final shape is not achieved in steps, by slowly deforming the shape. Instead, the final shape is defined by creating a profile (resembling one half of a cross section of the desired shape) and revolving it around a symmetry axis that specifies the center of the final object (See Figure 3.26 on the facing page). In other words, it is a process that creates new geometry, and not one that modifies and deforms existing geometry.

To answer whether or not the ceramic artists and the glassblowers will find the constructive approach intuitive and useful will most likely require the

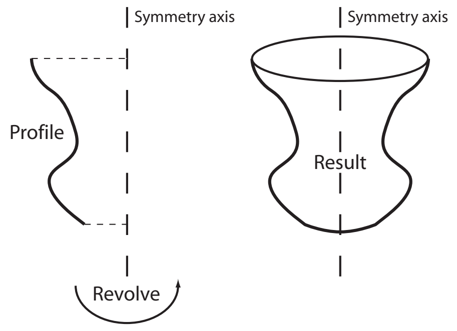


Figure 3.26: Revolve as it is performed using a computer, involves creating a profile that resembles one half of a cross section of the desired shape

design and implementation of a system, followed by a thorough usability test.

3.3.2 The Cutting and Carving Processes

Cutting and carving, as it is understood in the traditional sense, is widely used by ceramic artists as well as glassblowers. In the case of ceramics carving is usually used for creating holes in a larger shape, adding detail to the object by removing and separating material from the main object (See Figure 3.27 on the next page). Carving is also used in cases where the fingers are too coarse for achieving the desired shape. Cutting is not used that often, since the material can be directly shaped into the desired shape using the hands directly.

In a virtual environment the distinction between cutting and carving is less clear, because the difference is only the size of the part removed from the object. The only way to define a clear difference is to define cutting as the action of separating an object into two new objects, and carving as removing a part from the object and discarding the removed part.

With glassblowing carving is a more complicated matter because of the material properties and the working temperature of the glass. In the studies we performed, we did not see the method used in any of the products created during our visit, or exhibited in the shop. The use of regular cutting is more widespread, for a number of tasks. When a new lump of heated glass is attached to the object they are working on, the correct amount is applied by cutting the attached glass in the correct position. Another use for cutting is

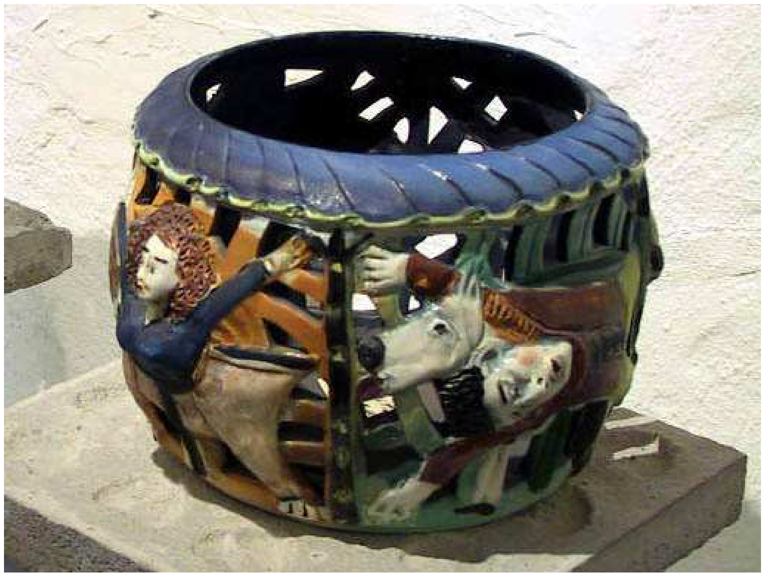


Figure 3.27: Carving is used to add details, such as holes, to objects by removing and separating material

the creation of creases in the objects, for instance to allow bending of only a specific part of the object.

Using a computer, 3D graphic artists are able to perform the same operations as the ceramic artists and the glassblowers. The difference is that cutting may not be the optimal way of achieving the same shape, because of how the geometry is structured. This issue is related to the deformation versus construction issue, in that it is sometimes easier to construct a desired shape in its final form, that it is to start out with a simple shape and gradually apply deformations until the final shape is achieved. Still, there are cases where cutting and carving are the best and most effective ways of working out the shape of an object.

With the computer system, the 3D graphic artist also utilizes the cutting tools for other purposes, that are not even possible in the real world, or would not make any sense to do in the real world. An example of this is when cutting is used to separate a single geometry object into two objects. Although they now function as two geometrical objects, they still function as a whole, while a real object would have been broken by the cutting operation. The main reason for cutting an object into more parts on a computer system is to control the affected areas for other operations that are applied to the object. So to summarize, cutting on a computer system is used both for removing parts of an object (separate) and for controlling what region of an object a geometrical operation is applied to (isolate).

3.3.3 The Deformation Process

Deformation proved to be a very used approach for modeling in the real world. This goes for ceramics as well as glass, where the primary task is to deform a selected lump of material. As previously stated, the way ceramics and heated glass is deformed varies a bit, mainly because of the temperature of the materials.

In both these cases a deformation of the object is affected by the following parameters [Dic02]:

- **rigidity** - the physical property of being stiff and resisting bending
- **cohesiveness** - the intermolecular attraction by which the elements of a body are held together
- **elasticity** - the property of returning to an initial form or state following deformation
- **plasticity** - being capable of undergoing continuous deformation without rupture or relaxation

The ceramic artist and the glassblowers have a long an very extensive experience of working with their materials, which means that they know exactly how the material responds to their interaction. For instance, bending a little turn up on the edge of an ashtray is as simple as flexing the thumb, for a ceramic artist that is familiar with the material. Unfortunately this real world, and physically correct, kind of interaction also limits what the artist is able to do at a time, and to what scale the object is affected by it. An example of this is when the artist would like to shape an area that is larger than the hands of the artist. So even though the artist is in more or less perfect control of deforming small (hand- or tool-sized) regions, deforming larger regions is a more complex task, that is often solved in a number of smaller steps.

For the 3D graphic artist, using a computer, deforming objects is also related to a few issues. Here, changes to the shape of the object may require a rather complex restructuring of the underlying geometric model, in order to correctly accommodate the changes desired by the designer. Expert users, that have an understanding of the underlying model, are able to do almost any thinkable kind of deformation of the object, because they are not limited by the laws of physics. This larger degree of freedom to shape the objects is related to the following properties of the underlying mathematical model:

- **External forces** - The model may or may not be affected by external forces, such as gravity, depending on the selections of the user and the virtual environment that supports the modeling. This means that an object can be deformed into an appealing shape, that would be difficult to recreate in the real world.

- **Non-constant volumes** - An object can be scaled to an arbitrary size without adding new material. For instance, a deformation to an object can be achieved by scaling a part of an object independent of the remaining object, or by pulling the surface of the object outwards, which changes the volume of the object. This provides the user with extra freedom for investigating alternative shapes and sizes without applying new material. Some systems also support constant volume objects, if the designer chooses this.

Unfortunately, deforming objects using a computer system also has disadvantages, especially for non-expert users that have no understanding of how control points can be used to change the shape of an object. Whether or not a deformation is complex depends mainly on how the underlying geometry is structured for the object, regardless of it is achieved by moving control points or applying forces to the object. In our observations, the 3D graphic artist sometimes needed to restructure the geometry and add additional control points in order to accommodate the desired deformations. Even for the expert user, adding the correct amount of control points, in the correct locations, was a rather complex task, that did not always give enough control over the deformation.

3.3.4 Summary of Concepts

This section summarizes the concepts described so far in this chapter. The different topics we discussed, namely revolve, cutting and carving and deformation are outlined in Table 3.1 on the facing page.

Comparison of Concepts		
Ceramics	**Glassblower**	**3D Graphic Artist**
Revolve The ceramic artist uses the revolve operation as a means of deforming an already existing objects (*Deformation*).	**Revolve** The glass blower uses the revolve operation as a means of deforming an already existing objects (*Deformation*).	**Revolve** the 3D graphic artist uses revolve to create new geometry (*Construction*). This affects other parts of his work, as he often thinks more in terms of construction, than in altering existing objects.
Cutting and Carving Used to remove parts of an object (*separate* material). Cutting usually involves larger pieces of material, while carving is for adding detail to the object.	**Cutting and Carving** No use of carving, mainly because of the choice of material. Cutting is used for applying the correct amount of material, when two lumps of glass are merged together (separate material). Another use of cutting is for creating creases in the object, to allow bending of only a specific part of the object.	**Cutting and Carving** Uses cutting in the traditional way (separate material), but also for controlling what region of an object a geometrical operation is applied to (*isolate* affected regions). Often, this cut is temporary, and may be removed when the wanted operation is applied to the given region of the object.
Deformation Deformation proved to be a very used approach for modelling in the real world. This goes for ceramics as well as glass, where the primary task is to deform a selected lump of material. The hands are the primary tool for deformation, since they provide a very direct and powerful way of interaction with the object, although at a very limited scale.	**Deformation** The same widely usage of deformation, although the glassblowers use a more varied selection of tools than the ceramic artists for the process, mainly because of the temperature of the material.	**Deformation** Deformation can be applied at a larger scale, but may require complex restructuring of the underlying geometric model (expert knowledge). Control of external forces and constant or non-constant volumes during deformation. Getting adequate control over some deformation may be very difficult, even for expert users.

Table 3.1: An overview of the discussed concepts for ceramic artists, glassblowers and 3D graphic artists

3.4 Further Observations

This section discusses further observations made during the case studies, including work processes, immersiveness, and the notion of an interaction loop.

3.4.1 Differences in Work Processes

The artist working in the physical environment has a very different approach of modeling the desired objects compared to an 3D artist, and as a result they have ease and difficulties carrying out different tasks. Both kinds of artist can easily create simple organic geometric shapes by their way of revolving. A big difference appears when for instance the turn ups on the ashtray from Figure 3.15 on page 27 are to be created. The ceramic artist creates the turn ups with ease, partly due to the fact that it is a simple task, and partly because of his skills and experience. The 3D artist on the other hand, had some difficulties in achieving the same deformation, even though he likewise has extensive experience with his work. The task for the 3D artist is complicated, because he has to make changes in the underlying geometric structure in the object to make it behave correctly.

The 3D artist had relative ease in creating the candlesticks in Figure 3.22 on page 30, by simply deforming an relatively simple object. The glassblower had some difficulties by creating the candlesticks. The problem is the material properties of the heated glass. According to the glassblower, the glass has "a will of its own" and if it first "decides" to deform in a given direction, the process is irreversible. Also, the glass gets stiffer as it cools down, and continuously changes material property along with the cooling, so there is a time limit in which the glass can be worked with. This restriction does not apply to the 3D artist, that can experiment with the shaping of the candlesticks at his own pace.

3.4.2 Immersive Environments

The work process of the ceramic artist was mainly to deform simple primitives with his hands as he was controlling the rotation of the object with a foot pedal. Furthermore, he used a few tools to manipulate the finer details of the object. The glassblower has another approach for working with his objects due to the extreme temperature of the object. The initial idea of deforming an object is the same as with the ceramic artist, but a difference is that the glassblower uses one hand as a primary hand and the other hand as secondary. The secondary hand is only used as support for the blowpipes and for rotating them. The 3D graphic artist also used a primary and secondary hand, the primary for drawing with the mouse, navigation plus choosing menu items and buttons. The secondary hand is used for choosing how to navigate in the environment, for shortcut keys and for opening a hotbar with fast access to a lot of the program functions.

For all case studies, the environment does support them in the design process. The ceramic artist and the glassblower both organize their working place to support the whole working process. This is achieved by physical arrangements that simplifies or eases the work process of creating their art. The 3D artist also adjusts his workspace to ease and support his work flow. He achieved the adjustment by selecting the layout of the user interface in the application he uses. For instance, the level of detail to show in the hot box in Maya can be adjusted.

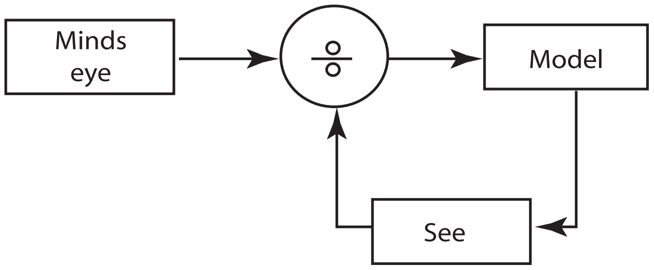


Figure 3.28: The interaction loop a user enters, when modeling

3.4.3 Interaction Loop

As described in Section 2.1 on page 9 drawing is an iterative process, since initial lines guide the eye to more precise lines, and so on. This may also apply for other kinds of artistic shape design, be it construction or deformation, because the eye also gets the feedback from the initial creation in these cases, which enables the user to adjust it to something closer to the desired result. This iterative process is illustrated in Figure 3.28, which is based on the iterative principle in Figure 2.1 on page 10. The figure illustrates that the user models the object from the minds eye. The object then feeds the eye with input. Hereafter, the user compares the result from the modeling process with the expected result from the minds eye. The differences are evaluated and the user corrects the object. This loop continues until the user decides that the creation is finished. In some extreme modeling cases like glass blowing, reshaping of the object in progress is not possible, so the iterative part of this modeling type is to create a new object from a new lump of glass, but with the experience from the failed object in mind. By using the experience from the failed object, the iterativeness of the process is maintained even here. This shows that modeling has an interaction loop, for refining the resulting object.

4 Guidelines for Paradigm Selection

Based on the observations and considerations made in Chapter 3, we are capable of setting forth a number of guidelines for choosing modeling paradigms, that are suitable for artistic shape design.

We describe three major areas of recommendations that, as a whole form the basis for designing the interactive modeling application. These areas include guidelines concerning functionality, interaction and the underlying platform with regards to both software and hardware.

4.1 Functionality

This area covers the following topics:

- What kind of modeling should be supported for the user in using the application?
- What paradigms correspond to the requirements of supported modeling tasks?

The system should, as a minimum, be able to support the user in recreating the objects from the cases we observed. To do this, the following guidelines should be fulfilled:

- Construction of new geometry in the system
- Deletion of geometry, for instance by using an eraser

- Cutting and carving
- Deformation
- Supporting geometric routines - revolve, extrude, scale and mirroring

Based on the analysis performed, a selection of modeling paradigms, that correspond to the tasks described above, should be chosen. To achieve this, a wide selection of related work in the area of interactive modeling should be investigated, in order to identify those best suited for our application.

4.2 Interaction

Interaction covers the following topics:

- User Interface - which of the following should be used to interact with the system and switch between the implemented functions
  - Direct manipulation/Direct interaction
  - Menu
  - Toolbar
  - Gesture
  - Audio
- Input Devices - which kind of hardware input devices should be accessible
  - Data glove
  - Wand
  - Pen

Interaction with the application is of crucial importance, and one of the main reasons for moving the design process from an ordinary desktop computer to a more suited virtual environment. Furthermore, interaction is executed in loops as illustrated in Figure 3.28 on page 39, and therefore this principle must be supported by our interaction method.

With the inexperienced users in mind, the user interface should be as simple and straight-forward as possible. Different users may have different preferences with regards to the usage of gestures and audio feedback for instance, but ideally they should be able to chose the method they prefer. The use of more advanced interaction techniques, such as gestures and audio, will inherently affect the selection of hardware and how the underlying platform for the system will be designed. Therefore, these considerations will be included when the appropriate hardware is selected for interaction devices and the platform that should support the different kinds of interaction paradigms.

The selection of input devices should be made with intuitive interaction as one of the primary goals. Also, they should be comfortable to use and allow the user to express the shapes they imagine, without inducing unnecessary cognitive overhead. In other words, the input devices should support a natural flow of thoughts and the creation of new ideas. This is also stated in the theory of ready-at-hand and present-at-hand, which tells that a tool that is ready-at-hand is transparent to the user, so he can concentrate on the actual work. If the tool only is present-at-hand, the user pays attention to the tool, and will therefore be hindered in an optimal work flow, as the tool takes some of his time [DM93].

The precise selection of whether the best suited input device is a glove, wand or pen depends on the actual use context, the interaction paradigm and certainly also the modeling paradigm, that for instance may require a data glove. Recommendations should be made on the different combinations, and a usability test may be performed to verify the results, with focus on the user's point of view.

4.3 Platform

This area covers the following topics:

- Software - what software should form the basis software platform for the interactive modeling application
- Hardware - which hardware could be compatible with the application

openNURBS [MA00] will be a suitable tool for handling the basics graphics functions, preventing that we need to implement every graphic function from scratch. We have chosen openNURBS partly because its free, and partly because we have seen some impressive implementations based on openNURBS.

VR Juggler [Tea01] has the advantage of being capable of executing on a standard workstation as well as the Cave Automatic Virtual Environment

(CAVE). This gives us the possibility of developing our application at a workstation, and then later execute the application in the CAVE. Furthermore VR Juggler is open source. A more detailed description of VR Juggler can be found in Appendix A.

The visualization should be realized by a Head Mounted Display (HMD) or in the CAVE, because a desktop computer not will be able to provide the user with the correct 3D visualization. The choice of VR Juggler provides us with the freedom to develop on a desktop computer and then move the application to the CAVE or a HMD later. This means that the choice of visualization device can be postponed to later in the development. The choice of interaction is more crucial at this time, because there will be extensive differences in the design of the interaction depending on the choice of a data glove or a wand. Furthermore the choice of haptic feedback or audio feedback will be essential for the design of the interaction.

5 Related Work

This chapter presents an overview of research and literature that is related to the context of our project, namely Interactive 3D Modeling and Artistic Shape Design. The paradigms from these studies are divided into the following groups:

- **Construction** - paradigms concerning creation of new geometry.
- **Deformation** - paradigms that in some way change existing geometry.
- **Hybrid** - paradigms that are mainly a combination of the other two.

Also, each paradigm is discussed and evaluated with regards to the context of our project, and the issues we identified in the analysis in the previous chapters.

Table 5.1 on the next page presents an overview of the different paradigms encountered through our studies of literature. In the table these are divided into the groups described above. Furthermore, they are divided into polygonal or parametric depending on which of these data structures they make use of. This table is made to give an overview of which paradigms can be combined in one application without the need to translate from polygonal to parametric or visa versa.

In [DBW$^+$00] Joachim Deisinger et al. presents the results of a workshop which tested three different prototypes of interactive modeling applications. Since this is not a paradigm it is not placed in one of the previous described groups nor is it a part of the table above. The results are based on the feedback from 36 design professionals. The analysis of these results and their

	Polygonal	**Parametric**
Construction	*Surface Drawing* by Steven Schkolne et al. ([SS99], [SPSa], [SPSb]) *Teddy: A Sketching Interface for 3D freeform design* by Takeo Igarashi et al. ([IMT99])	*3-draw: A Tool for Designing 3D Shapes* by Emmanuel Sachs et al. ([SRS91])
Deformation	*Skin: A Constructive Approach to Modeling Free-Form Shapes* by Lee Markosian et al. ([MCCH99]) *inTouch: Interactive Multiresolution Modeling and 3D Painting with Haptic Interface* by Arthur D. Gregory et al. ([GEL00])	*Direct Manipulation of FFD: Efficient Explicit Solutions and Decomposible Multiple Point Constrains* by Shi-Min Hu et al. ([HZTS01]) *Preventing Self-Intersection Under Free-Form Deformation* by James E. Gain and Neil A. Dodgson ([GD01])
Hybrid	(None found)	*Collaborative Geometrical Modeling in Immersive Virtual Environments* by M. Usoh and T. I. Vassilev ([USV96]) *An Interface for Sketching 3D Curves* by Jonathan M. Cohen et al. ([CMZ+99])

Table 5.1: An overview of the paradigms presented in the studied literature

experience with the systems will be useful in improving interactive modeling applications and further development in this area in general.

The paper states that modeling in virtual environments has the potential to be more powerful than existing modeling applications on desktop systems. However, an effective, user-friendly immersive modeling tool has yet to be developed. Research areas include the exploitation of human fine motor skills and deciding what level of functionality is suited for interactive modeling in immersive environments. During the workshop insight was gained into how participants act during the design process, with regards to work-time, usage of tools and so forth.

The workshop identified three distinct phases during a design:

- conceptual phase: ideas, thoughts and their first visualization,
- elaboration phase: working out alternatives; the quantification and detailing of sketches and models and
- presentation: working out the ultimate shape, character and function of the concept for the general public.

One of the important results from the workshop, is that current interactive modeling tools, only cover the conceptual phase and at most early stages of elaboration. The main reason for this being that no specific tool is able to provide users with the appropriate level of functionality and ease-of-use to span all the phases of a design process. On the topic of ease-of-use, the tools should function in a way that supports a natural flow of thoughts and the creation of new ideas.

Sketching has until now, been an important part of the conceptual phase, but the most widely used tool for this is a regular pencil and a piece of paper. Although it may seem to be a simple kind of interaction, the “paper and pencil” metaphor has proved to be a very powerful tool. However, traditional sketching is an inherently two-dimensional activity. Therefore, the workshop investigated the possibility of extending sketching to three dimensions.

The three phases identified in the workshop are useful for setting the scope of the interactive modeling application that we will design. Most of the results from the workshop will serve as recommendations for how users will interact with the application. Their recommendation of the “paper and pencil” metaphor corresponds well to our point of view (as mentioned in Chapter 1 on page 4) for systems that are intuitive and easy to learn. One of the more promising systems that have extended the pen and paper to 3D, is the Surface Drawing system which we will discuss later in this chapter.

5.1 Construction Paradigms

In [SS99], [SPSa] and [SPSb] Steven Schkolne et al. presents a concept called Surface Drawing. Surface Drawing addresses several issues in creative expression and perceptual thinking by providing a direct link between the motions of the hand and the forging of shapes. Surfaces are created by moving the hand, which is instrumented with a glove, through space. The drawing process is illustrated in Figure 5.1, where a user is drawing in 3D space. According to the articles, this technique allows both novices and experts to create forms without the perceptual constraints of a mathematical structure or a large toolset.

Using this concept no forward planning of the construction is needed. The design space can be freely explored during the modeling process. It supports unconstrained erasing and buildup of new geometry, and allows the user to freely grow, join and erase surfaces based on the hand motions.

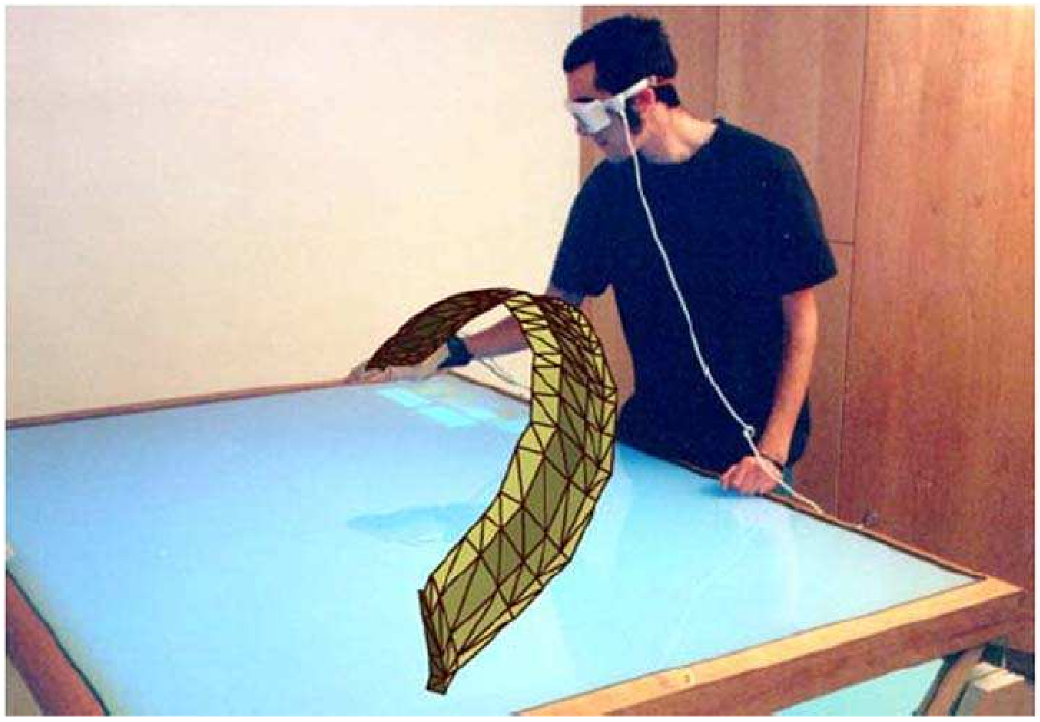


Figure 5.1: A user draws in 3D space using the Surface Drawing system

The Surface Drawing method comes across as a method that is very simple to learn, in that it provides a one-to-one mapping between the gestures of a user, and the resulting geometry. We believe that the method is easy to learn, and that it is suited for experimentation, because users are able to delete geometry the same way they would use an eraser with the lines they created using a pencil. Only, the concept of paper and pencil is extended to 3D in this case.

Surface Drawing is well suited for sketching the initial ideas of a designer, and for quickly creating alternative shapes in the early stages of the design process. This fits perfectly together with the context and scope we set for our modeling application, and the selection and skill of the users that are supposed to use the application. The users do not have to think about the underlying mathematical structures, such as coordinate systems and control points, in order to model interesting shapes using the Surface Drawing method.

But, on the other hand, the simple interaction also has some limitations, most of which are related to the speed of the method, that is influenced by the one-to-one mapping of the hand gestures. For instance, creating a surface that has a large number of S-shaped curves that follow each other, can be a large and somewhat tedious task to perform using only surface drawing. Instead it could be combined with geometric operations such as revolve, sweep and extrusion in order to speedup the creation of some types of objects. Fortunately, this issue does not mean that the least experienced users will be unable to create the desired shapes, but only that they will take more time to get the job done.

In [IMT99] Takeo Igarashi et al. presents another construction paradigm called Teddy. It enables the user to create 3D objects by drawing 2D freeform strokes. The freeform stroke is closed as a silhouette, and from this silhouette the application creates a 3D polygonal surface. A region is inflated by making wide areas fat and narrow areas thin. An example of an object being created in Teddy is shown in Figure 5.2 on the next page. The geometric representation of the polygonal surface is a standard polygonal mesh.

The application can create, cut, erase, paint, extrude, bend, smooth and transform objects.

Teddy provides users with an easy-to-learn way of creating 3D objects, which is a wanted property with regards to the system we will design. Most of the ideas are quite simple, and easy to control using the pen-like interaction device, but the user is quite limited in the shapes that can be created. Shapes that are difficult to create, include rectangular shapes and objects with sharp edges.

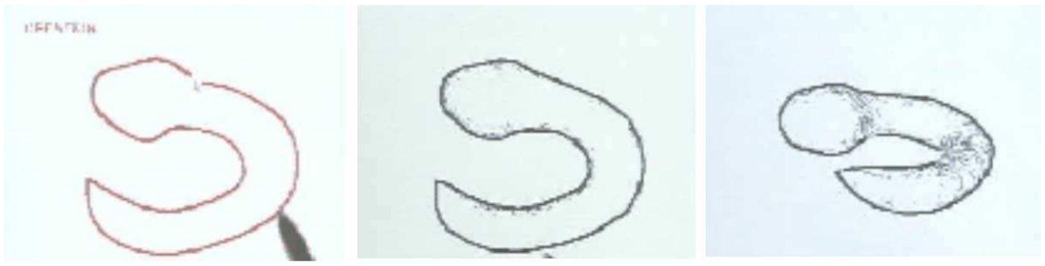


Figure 5.2: An object being created in the Teddy application. In the first picture the initial 2D stroke is drawn. In the second picture, the object has been inflated to an 3D object, and finally the last picture shows the 3D object rotated, to better show the shape of it

This means that an approach that is similar to Teddy can be used for the initial design stages of organic shaped object, mainly by non-expert users that are just learning to use the system. For more complex shapes, that require a larger degree of freedom, geometry should be created using another construction approach, or by deforming the object using some kind of deformation paradigm.

In [SRS91] Emmanuel Sachs et al. presents a way to design 3D shapes by sketching. They present an application that provides a virtual approach to the design sketching, that often is done on paper before a digital drawing is created. The application uses a pair of six-degree-of-freedom input devices, one held in each hand. In the primary hand the user holds a pen, by which he is drawing in free space, and in the other hand he holds a palette which is locked to the global coordinate system for the objects he draws. In that way the user can turn the object he is drawing with the secondary hand.

The shapes are constructed in four steps:

1. The curves are sketched directly in 3D
2. The curves are edited by deformations
3. The surfaces are fitted to groups of linked curves
4. The surfaces are deformed to obtain the desired detail

The interaction should be intuitive to the user, because there is a one-to-one mapping of hand motions to virtual object motions.

The principles in 3-Draw allows the user to draw with a one-to-one mapping directly in 3D with the possibility of moving the drawn object in 3D space using the secondary hand. The most obviously limitation of the method, is the restraint in the size the drawing can have. This size is limited of the reach of the user. The principle of drawing directly in 3D space seems very intuitive, as the user simply draws directly the wanted drawing. Even though the four step approach can perhaps be a little counterintuitive, it should be possible to adapt to.

5.2 Deformation Paradigms

A well known deformation paradigm is Free-Form Deformation (FFD). This paradigm can best be described by using a jelly metaphor. The object to be deformed is wrapped inside a jelly-like substance. To deform the object, pressure is applied on the outside of the jelly and the object is deformed along with the jelly.

To many, this is very counterintuitive and for this reason Direct Manipulation FFD (DMFFD) was introduced. This method provides the users with an extra abstraction layer on top of regular FFD. The difference is that the user does not have to know the mapping from the deformation of the jelly to the deformation of the object. Instead the user directly deforms the object surface and the system then calculates the jelly's deformation to satisfy the desired object deformation.

Without using the metaphor, FFD is a deformation method where the user manipulates control points to make the deformations of the object surface, whereas DMFFD allows the user to work directly on the object surface. When a point is moved on the surface the system calculates the control point positions that fulfill the surface point movement. Then these new control points is applied using standard FFD, like if the user had moved the control points manually.

DMFFD is illustrated in Figure 5.3 on the following page, where a half moon is put inside the "jelly". The deformation is made at the lower tip of the moon, which is pulled downwards. The system then calculates the corresponding jelly deformation to achieve the deformation of the object. The main problem with DMFFD is that it is very computational heavy and not really suited for real time applications.

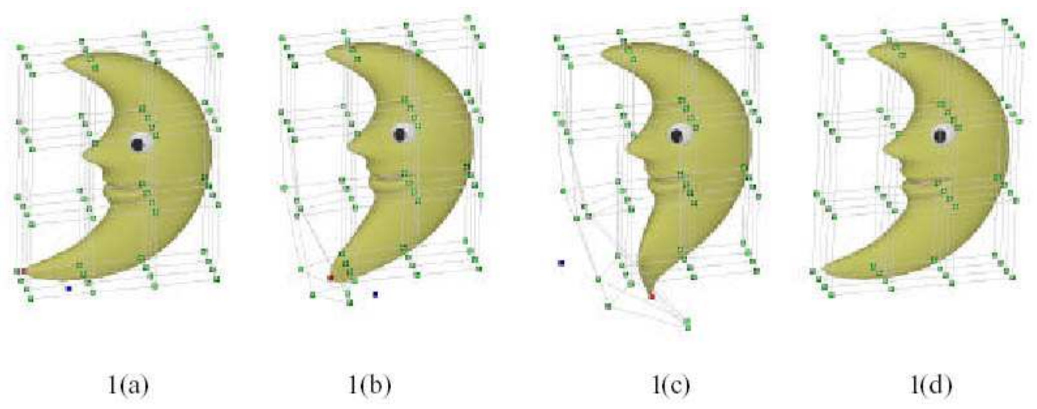


Figure 5.3: An object deformed by DMFFD. Pushing or pulling a point on the surface of an object deforms the surrounding jelly-like shell, that propagates the deformation to the object inside it

This problem is studied by Shi-Min Hu et al. in [HZTS01], where an alternative, and less computational complex, method to perform DMFFD is presented. Other methods are based on computing the pseudo-inverse matrix, which involves complicated calculations. The solution presented in this article solves the direct manipulation problem by using a constrained optimization method and thereby obtains an explicit solution. This only involves, according to the article, simple calculations. In addition they also show that multiple point constraints can be decomposed into separate manipulation of single point constraints.

Unfortunately, FFD and DMFFD only works with parametric surfaces. This means that surfaces, such as those created with Surface Drawing, can not be directly used together with DMFFD, since Surface Drawing is based on a polygonal model. This requires a conversion of the polygonal model to a parametric surface, using either surface fitting or approximation.

Another problem that often arises when using (DM)FFD is one known as self-intersection. Self-intersection is when the object after a deformation intersects itself.

This problem is studied by James E. Gain and Neil A. Dodgson in [GD01]. According to them Self-Intersection, and how to prevent it, is very overlooked in the 3D modeling community. Two approaches for preventing Self-Intersection under both FFD and DMFFD are presented.

The first approach is a precise detection of Self-Intersection, but it is very computational complex and therefore not suited for real time applications.

The other approach is a sufficient detection that is more suited for real time. Both approaches makes use of a injectivity test to detect if a wished deformation will result in Self-Intersection, by checking for one-to-one mapping. The injectivity test checks if all points on the surface before the deformation will map to a unique new position after the deformation, hence one-to-one mapping. This secures that the deformation do not cause self-intersection, since a self-intersection will result in at least two points at the surface sharing the same coordinates in space, and thereby they do not have a one-to-one mapping.

This approach to prevent Self-Intersection, will indirectly limit the user's freedom and to avoid this the article presents a solution called Adaptive Subdivision for DMFFD. In this solution the wished deformation is "chopped" up in half, which then will be carried out individually. If these two new deformations still result in Self-Intersection the process is repeated. This continues until either a maximum number of divisions is achieved or the resulting deformation is not Self-Intersecting.

Another form of deformation paradigm, named Skin, that can make polygonal models smother by refinement of the surface, is presented by Lee Markosian et al. in [MCCH99]. Skin is a particle-based surface representation that enables the user to interactively sculpt free-form surfaces. An important aspect of the Skin approach is the use of subdivision surface theory. Based on a set of rules they define, the particles form triangulations, that make them suitable for applying subdivision. Figure 5.4 illustrates an object which after a deformation would result in Self-Intersection. Therefore the deformation is altered by Adaptive Subdivision so Self-Intersection does not occur.

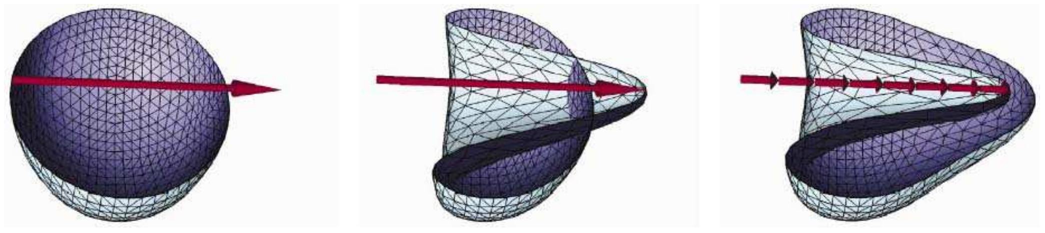


Figure 5.4: An object which after a deformation would result in Self-Intersection. Therefore the deformation is altered by Adaptive Subdivision so Self-Intersection does not occur.

Based on a collection of skeletons, which they refer to as polyhedral elements, a smooth surface is created by growing the particles, or skin, using a subdivision scheme that approximates the underlying skeletal shapes. Figure 5.5 illustrates two different triangulations of a surface. The Skin system is a part of an effort to create a free-form modeling system, that via direct interaction allows a user to define shapes by sketching the shape and its proportions.

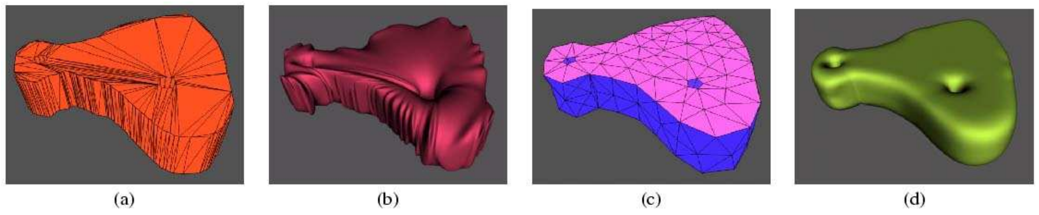


Figure 5.5: The same object triangulated in two different ways. The Skin system can redefine this triangulation for a smoother result. In picture (a) an improper triangulation is used, that results in the object surface in picture (b), for instance the crease between the two lower sections inside the object is unintended. In picture (c) another triangulation is used, which results in the clearly better result in picture (d)

The idea behind Skin is to take a simplified polygonal model, and create complex surfaces based on the polygonal model, resulting in a smooth surface that approximates the underlying polygonal model. The Skin system is well suited for rapid construction of *approximate* free-form surfaces, meaning that it cannot provide exact control of the final shape. This means that it can be used during the creative process of designing and shaping a product, but not in the final production process, where traditional CAD accuracy is required.

As the Skin system takes a simplified polygonal model as basis, it is not suitable for creating new geometry. Instead, Skin is useful combined with paradigms such as Surface Drawing, in order to smoothen a rough polygonal model, and to add details using the notion of creases, that the Skin system introduces.

With regards to ease-of-use, the creases can be drawn using a pen-like device on the surface of the object. Moving the crease to perform the deformation could then be achieved by grabbing the crease with a data glove (See Figure 5.6 on the next page). This should allow new users to utilize this kind of deformation, since the interaction and control of the deformation process is fairly direct.

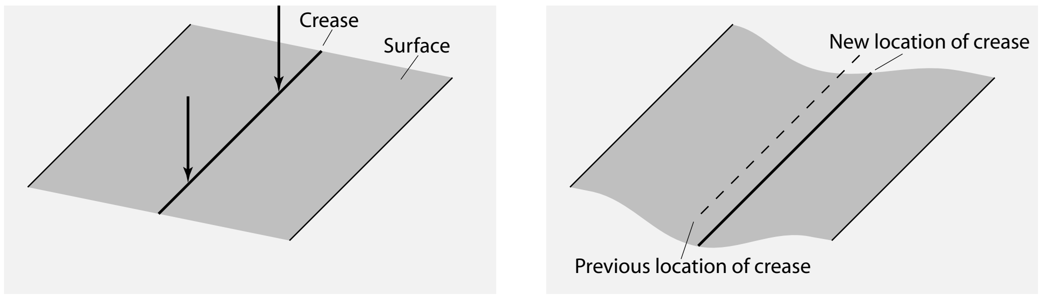


Figure 5.6: To the left the initial line describing the course of the crease is drawn. To the left the initial line is moved, and has thereby drawn the surface down resulting in a crease

Another approach to the refinement concept, called *inTouch*, is presented by Arthur D. Gregory in [GEL00]. *inTouch* can be used as a geometric modeler based on simple loaded primitives, a finishing system by for instance creating sharp features and finer details or as a painting tool where color and textures can be painted directly to the model's surface.

When modeling, the user chooses the resolution and attaches a probe to the surface. The changes are propagated up according to subdivision rules to the highest level of the mesh. At the user interface, actual deformations is accomplished by pressing and releasing a button when in contact with the surface.

inTouch is similar to Skin in that it also allows the user to take a relatively simple polygonal model, and change the shape of the object and add detail by deforming it. Therefore, *inTouch* also combines nicely with systems such as Surface Drawing. However, with *inTouch* it is also possible to load simple geometric shapes as a basis for the deformations. An interesting property of *inTouch* is the use of a haptic interface, that provides feedback to the user from the interaction with the system. Unfortunately their setup, with a pen-like device attached to a robotic arm, has a very limited range. This means that the haptic interface they developed, is not directly suited for larger immersive environments like those that utilize the CAVE system.

5.3 Hybrid Paradigms

[USV96] by M.Usoh and T. I. Vassilev focuses on building an interactive modeling application from a collaborative point a view. An important goal of the application is to provide tools which are intuitive and easy to use, allowing the designer to use natural hand motion to sweep out complex surfaces and interactively deform and reshape them.

The mathematical base of their work is a mixed geometrical-physical approach that combines bi-cubic B-Splines with models of material properties such as rigidity, cohesiveness, elasticity and plasticity. This information is used to ensure that the surfaces will deform in a well defined manner when a user applies a force to the surface.

Although parametric surfaces, such as the selected bi-cubic B-Splines surfaces, are a very powerful tool for geometric modeling, the traditional interaction through movement of control point is neither an elegant nor an intuitive way of interaction with the surface. An example of this is that even with a single bi-cubic patch, the user must deal with 16 control points. Although, after an extended period of time using the system, they believe that a user will become more aware of the relationship between the shape of a surface, and the placement of control points.

With their mixed geometrical-physical model however, the following actions can be performed on a surface, without directly interacting with the control points:

- Deformation through applying a single or a set of forces;
- Deforming a curve embedded on the surface;
- Deforming an area of the surface;
- Moving a single point from the surface to a new position;
- Moving a surface curve to a new curve in space.

New surfaces can be created interactively, by using a wand-like tool, as the user sweeps out a shape in 3D space, a surface is created such that it interpolates the space the user specified. They describe this process as turning the kinetic energy of a user into visible surfaces - turning a sweeping gesture into something concrete.

Unfortunately they were not able to get the kind of performance they were aiming for, since given the current implementation and hardware-setup, the surfaces will jump to their new shape when a force is applied. Instead, they would have liked the surface to deform continuously, in a rubber band fashion, as the force is being applied. However, they are confident that this will be possible in future versions of their application.

The system they designed deals with both construction and deformation topics, which gives the user a great deal of freedom with regards to the way they prefer to model. Although their features are impressive, our main concern is the performance issues they outlined. No matter how clever and intuitive a modeling paradigm may be, we believe that continuous feedback is a very important aspect of interactive modeling, be it visual or haptic. In their case they failed to achieve what they call the “rubber band” effect, where surfaces deform continuously. The ideas behind their system, and the use of parametric surfaces, seem very appealing if the performance can be improved to provide continuously feedback to the user.

In [CMZ$^+$99] Jonathan M. Cohen et al. presents a method for specifying 3D curves with 2D input from a single viewpoint. They claim that even though sketched curves are imprecise by nature, sketching allows a user to quickly create a curve that is close to the desired result. A trained artist should have the ability to produce accurate curves with his existing drawing skills, because the interface matches closely to pencil and paper.

The technique presented in this article falls in the category of direct manipulation of spline curves. The curves are sketched in two strokes: first a estimated curve, and then a stroke on a surface for instance under the curve. The stroke under the curve represents the shadow of the curve and thereby can the orientation of the curves depth be indicated.

Four basis methods for sketching curves are supported:

- Drawing a new curve on some plane
- Overdrawing a section of an existing curve
- Redefining a curve’s entire shadow
- Overdrawing a curve’s shadow

Figure 5.7 on the following page illustrates the drawing of a curve and then a shadow for the curve, and finally an overdraw on the curve. The user chooses between curve mode and shadow mode. The system determine that a stroke is an overdraw if the stroke starts and ends near a existing line, and is nearly parallel with this. The shadow could be projected in another direction, such as a wall plane. Furthermore the projection could be done at a terrain.

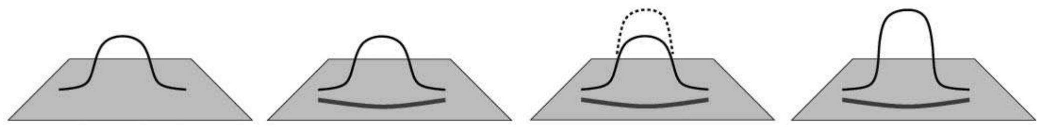
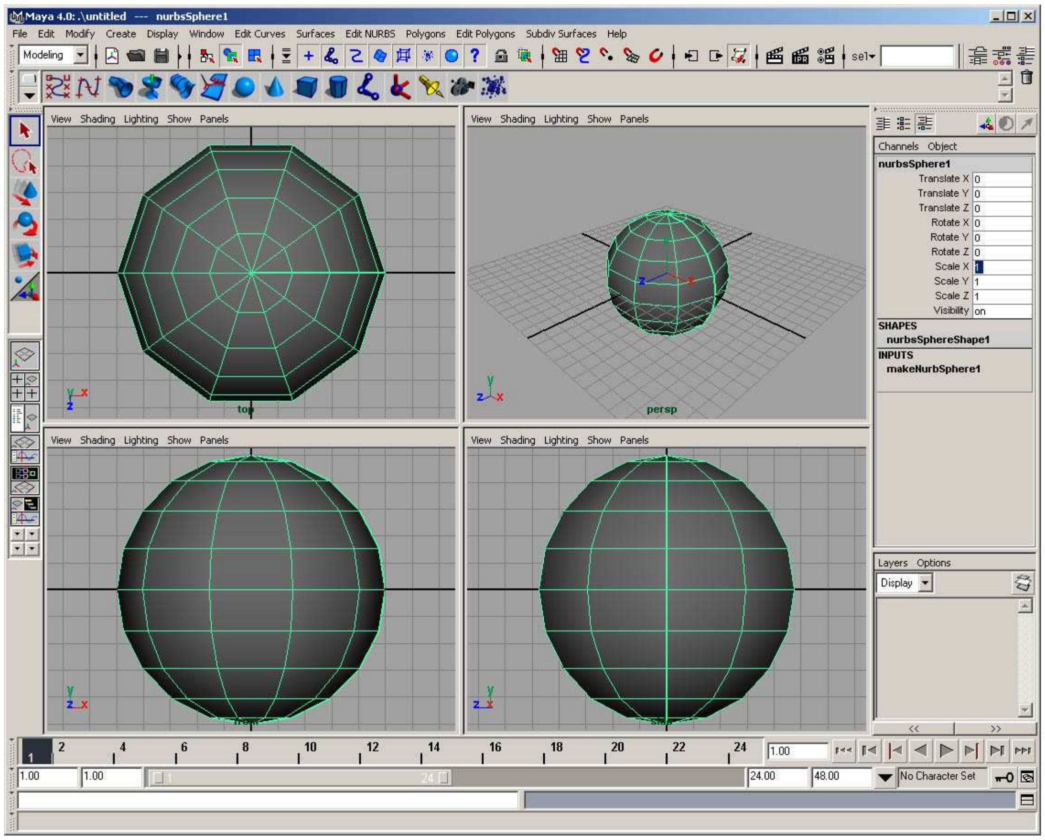


Figure 5.7: A curve is drawn, a shadow is added, and finally the curve is overdrawn.

We have some doubt about the intuitiveness of this approach, since the traditional way to draw something 3D-like on a 2D-plane is to draw in perspective and then add shadows. It seems somehow a bit awkward to define the depth of the drawing based on the shadow, and the shadow would likely be difficult to draw correctly based only on the curve and the picture from the minds eye (See Section 2.1 on page 9).


Maya 4.0: .\untitled --- nurbsSphere1
File Edit Modify Create Display Window Edit Curves Surfaces Edit NURBS Polygons Edit Polygons Subdiv Surfaces Help
Modeling
sel
View Shading Lighting Show Panels
top
persp
front
side
Channels Object
Layers Options
Display

nurbsSphere1

Translate X	0
Translate Y	0
Translate Z	0
Rotate X	0
Rotate Y	0
Rotate Z	0
Scale X	1
Scale Y	1
Scale Z	1
Visibility	on

SHAPES
nurbsSphereShape1
INPUTS
makeNurbSphere1

1.00 1.00 24.00 48.00 No Character Set

Figure 5.8: The four viewpoints from the graphical user interface in Maya

The idea about drawing 3D in 2D with only one viewpoint could be interesting, among other things for making the user interface more well-arranged than the traditional four viewpoints (See Figure 5.8 on the facing page), known from for instance Maya and 3D-Studio Max. The curve-shadow approach seems to have a limitation; if the user for instance wishes to draw two close parallel lines, he will end up with an overdrawn line instead.

6 Analysis Summary

This chapter summarizes upon the Analysis part, and presents further elaborations of the topics discussed in this part.

We did some studies regarding drawing principles which we found really useful, to get a good insight into how a 2D drawing can be achieved. These studies of drawing principles provided us with some basic knowledge in the art of drawing. We concluded that these principles could be mapped to an 3D environment, in which we draw surfaces. This similarity made it likely that the power of the paper and pen paradigm could be utilized in our context.

In the analysis we also did some fieldwork, where we studied how artists from different artistic trades performed their work. We believe this was a very good knowledge base for the project, because we got a good insight into how potential users would use the application, and what kind of tools they might be needing. Furthermore we observed that the ceramic artist and the glassblower worked using the deformation paradigm, and the 3D expert worked using the construction paradigm. The ceramic artist and the glassblower is forced to start with the deformation paradigm, because material does not materialize from thin air. This is not the case for the 3D artist, since he can create objects from nothing. One small drawback in the performed fieldwork was our choice of artists. Without further elaboration, we chose artists who all worked with creating art that in some way was centered around an axis. Because of this, the tasks we asked the 3D graphic designer to perform, indirectly had him designing around an axis too.

We feel that it would have been a good idea to study other artistic trades, which do not work around an axis, such as sculptors, and painters. A painter would probably also give us additional knowledge in drawing principles which we studied only in literature. We believe that sculptors would give us two different approaches for creating objects. Some sculptors would create art by starting out with an object, which is bigger than the size of the desired result, and which often is made of stone or granite. He would then remove parts and pieces of the initial object by cutting and carving until he reaches the result he was aiming for. Other sculptors would go the other way around, and start with an initial object which is smaller than the final result, and which often is made of clay or a material like it. He would then start adding small individual pieces until the desired result is reached.

It would also have been a good idea to study other computer designers, working with eg. CAD/CAM, as a supplement to the 3D graphic designer. This is something that definitely is worth looking at in future projects of this kind.

Furthermore, we studied literature to gain knowledge in the area of interactive 3D modeling. Especially, we needed to identify paradigms related to our project, so we had a better foundation for choosing paradigms, which we believed could fulfill the demands we found through the case studies. We learned that in this area, research is moving forward quite fast. Most of the literature we studied was a maximum of a few years old. So for future work, further and continuous studies of related literature would be a good idea to keep up with new initiatives and paradigms. By studying related work to artistic shape design paradigms, we placed ourselves in a situation, in which we could base our further work on other peoples work. Especially in the case of the Surface Drawing paradigm, we found some related work to base our work on, namely the work of Steven Schkolne et al. ([SS99], [SPSa] and [SPSb]). This is because we believe this paradigm fulfills some of the demands we identified by studying the artists at work, namely the use of the hands as a primary means of interaction, and to give the artists the creative freedom they have come to expect. Furthermore, we did not, through these studies of related work, find one "golden" paradigm which covered all aspects. Therefore, we can expect that our further work must include combinations of paradigms instead of just one paradigm.

Through these studies of related literature we mainly looked at literature that described paradigms which supported "Sketching in immersive environments" rather than "Simulation of physical modeling" (See Figure 1.1 on page 2). We know paradigms exist, like for instance Finite Element Method, which support physical modeling, but these are still not suited for real time applications at larger scale. Another important factor in physical modeling is tactile response, which still is not enough developed to resemble the real world. For these reasons we will continue the work in the area of sketching.

Part

2

Design & Implementation

This part concerns the more practical considerations of the project. After our studies of related work we found that it was hard to find one paradigm which alone fulfilled our wishes. Therefore, as the first task in this part, we designed a platform to support several paradigms, which as a whole could fulfill our intentions with the project. Hereafter, we chose one paradigm which we designed to use the platform. Next, we describe the implementation performed in relation to this project, and the hardware setup used, followed by a presentation of the application. Due to the time period at hand we chose to implement a sys-

7 Design

This chapter describes the design of a platform for an interactive modeling application, that takes into consideration the lessons learned from the case studies as well as the guidelines for such an application, which we presented earlier in this report. The design will focus on the issues we identified during the case studies and the analysis that followed. Furthermore, the design specification presented in this chapter will function as a recommendation for how an interactive modeling application, that utilizes immersive and semi-immersive virtual environments for artistic modeling purposes, should be designed. It is our hope, that this specification can be a stepping stone, which others can use for developing future modeling applications for artistic and rapid prototyping purposes.

Firstly, this chapter presents the system architecture for an interactive modeling platform, that is designed for being scalable and flexible enough to support a number of modeling paradigms, that provide users with maximum artistic freedom and expression. The underlying ideas behind the chosen architecture, and the design issues encountered are described, as well as the primary design goals for the platform. Secondly, the properties of each of the components in the platform are specified, including the purpose and interface for each component.

7.1 System Architecture

This section describes the overall system architecture, which is shown in Figure 7.1 on the following page. The architecture is divided into three layers, namely hardware, third party software and the software we have decided to

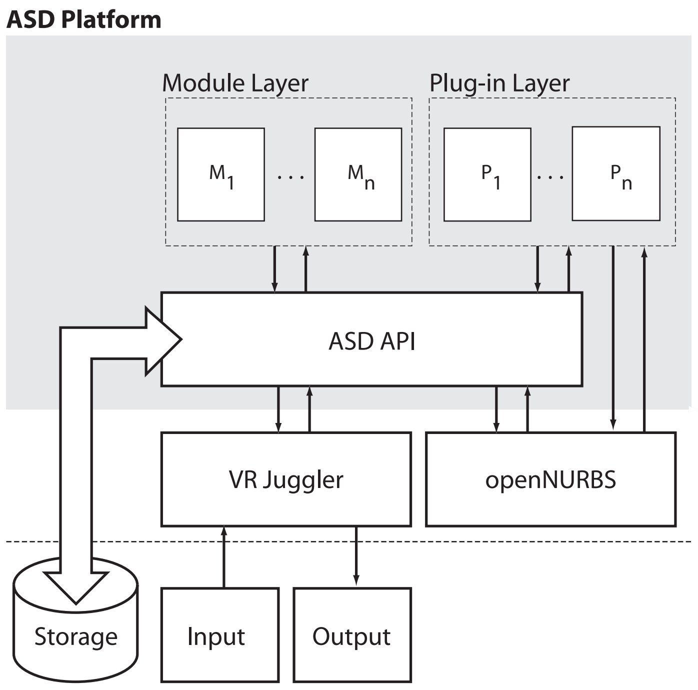


Figure 7.1: The system architecture. The grey area is the parts of the architecture we will design. The parts just below the grey area are third party software, and at the bottom the hardware is shown

design. The software is divided in two to make a clear distinction between what we develop and what others have developed. A detailed description of each component in the architecture is provided later in this chapter.

From the case studies performed, we gained insight into how ceramic artists, glassblowers and 3D graphic artists work. The experiences from these case studies and our analysis form the basis for the design decisions of the ASD Platform we describe in this chapter. The primary lessons learned from the studies include:

- Depending on the background of an artist, deformation paradigms may be preferred over construction paradigms, and vice versa as documented in our case studies in Chapter 3

- Different modeling paradigms may require a specific input device, in order to provide intuitive interaction for the user, as experienced in our case studies, that are documented in Section 3.2

In order to accommodate these issues, one of the design goals for our platform is the ability to support a multitude of modeling paradigms, which should integrate seamlessly with the platform. Ideally, the platform should allow users to work with the system in the way they prefer, regardless of whether their background is in ceramics or 3D graphic design. In our case, the platform will support this through interfaces to the module and the plug-in layers, that connects to a shared API for uniform access to interaction devices and geometric data models. Also, we will provide a solution to a parallel representations of polygonal and parametric based geometric data structures, since experience has shown us, that these two approaches supplement each other in terms of the ability to provide effective visualization, and their ability to effectively represent deformation and shape alterations at varying scales.

With regards to connectivity to hardware devices for interaction, a library that provides simple and effective access to these devices, will be integrated with an shared API, that facilitates the communication between modeling paradigms, and the interaction devices that a specific paradigm utilizes.

Dividing the platform into three distinct layers has a number of implications for the overall characteristics of the architecture. As with other multi-tier applications, a number advantages are gained by this architecture:

- Each layer can be understood without understanding the insides and inner workings of other layers, that communicate with the layer in question.
- Layers are a mechanism for structuring and organizing a complex application into smaller, less complex pieces.
- Layers minimize dependencies between different parts of an application, which makes it possible to replace the implementation of one layer, without affecting neighboring layers.

Different architectures provide varying advantages and disadvantages. The important thing in selecting and designing our architecture is to select a specific architecture, that based on the application context, provides the best weight in the advantages it sets forward, compared to the disadvantages it yields. In the case of a layered architecture, the immediate disadvantages include:

- Too many layers degrade performance - The extra encapsulation inherently affects performance.

- Even though layers are effective for encapsulating changes, "cascading changes", that affects each layer above or below the layer that introduced the change, may occur.

With these issues in mind, we still believe that the advantages clearly outweigh the disadvantages of the layered architecture. Also, the layered architecture is a proven, and well understood way of structuring complex applications, in order to achieve a high degree of encapsulation and maintainability.

The layered architecture provides an abstraction mechanism, that hides the details of the underlying components, enabling developers to focus on implementing a modeling paradigm, without worrying about connectivity to input/output devices, and event handling. Furthermore the combination of plug-ins with modules encourages code reuse and encapsulation, for the developers using the platform. The roles and inner workings of these layers are described in greater detail in the following section.

7.2 Component Specifications

The component specifications outline the components in the ASD platform, including the Module, Plug-in, ASD API, VR Juggler and openNURBS layers, their roles in the system, as well as the interfaces that describe how the components communicate.

7.2.1 The Module Layer

This section presents the design considerations for the Module layer, that is located above the ASD API layer. The purpose of the Module layer is to provide an extensible architecture that accommodates different modeling paradigms, that communicate with connected hardware through an underlying API layer and VR Juggler. In order to ensure that Modules integrate successfully with the remaining system, and the extensibility of the platform as a whole, the following general guidelines for designing this layer are as follows:

- Each module should respond in a well-defined way to communication from the API layer
- The dependencies and requirements of a module has to be satisfied by

the system, through communication with the API layer. This includes dependencies to plug-ins and hardware devices.

In order to accommodate these issues, modules are designed as pieces of functionality that are plugged into the platform, in order to provide the user with a well defined modeling paradigm. The internal structure of a module consists of the following components:

- An implementation of the modeling paradigm itself
- General event handlers for communication with the underlying API layer
- Interaction methods for the implemented modeling paradigm

A module takes the shape of an object, that is registered and instantiated by the platform through the API. This means that the module has to implement an interface, that the API layer uses to communicate with the module. To achieve this, each module inherits from a generic abstract superclass, that describes the module interface. Therefore, the platform can handle the different modules in a uniform way, and that the implementation of new modules are ensured to have the correct functions for the module interface. Based on the case studies, and our investigation of related literature, the following modeling paradigms could be implemented as modules:

- Surface Drawing - a construction modeling paradigm
- DMFFD - a deformation modeling paradigm

The event handlers constitute a collection of member functions or methods, that the platform utilizes for each time-slot that is calculated. A number of events can be triggered by the platform in a time-slot, depending on what the user is currently doing, including:

- Register - Registers this module with the platform. This call succeeds if the dependencies of the module are met
- Unregister - Unregisters this module, and may cause other dependent modules or plug-ins to unregister also
- Update - recalculate the geometry of a specific object

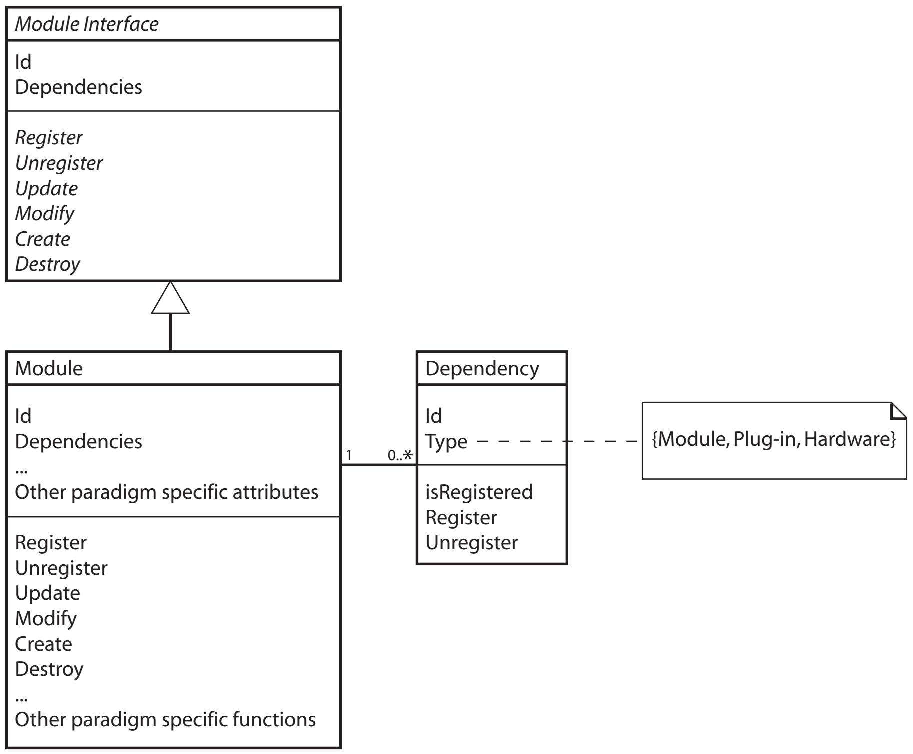


Figure 7.2: The class hierarchy for the Module Layer, including the abstract superclass *Module Interface*, which specifies the interface that modules implement

- Modify - apply a geometric operation to a specific object, based on data from the connected input device
- Create - Instantiate the module and associated data structures
- Destroy - Deallocate associated data structures and free the module

A complete specification of the Module Layer objects is shown in Figure 7.2, that illustrates the abstract superclass that describes the interface a module must implement, and the association of modules and dependencies.

7.2.2 The Plug-in Layer

This section outlines the design considerations for the Plug-in layer, that is located next to the Module layer.

The purpose of the Plug-in layer is to facilitate an extensible communication layer to the geometric routines located in the openNURBS library. It should provide a shared, common code base for utilizing the geometric routines that openNURBS contains, to support higher level geometric functionality that is required by a number of modules in the Module layer. Some of the main reasons for introducing a Plug-in layer include code-sharing and reuse as well as encapsulation of complex functionality and data structures. Also, the plug-ins combined with openNURBS are a means of providing a parallel representation of polygonal and parametric geometric data structures, from which a developer chooses the best suited representation for the specific use context. The most important design issues include the following topics:

- A Plug-in should provide a uniform way of accessing the geometric routines it supports, in order to allow several modules to use its functionality.
- Dependencies between plug-ins may exist, and should be handled in a consistent way. A register mechanism for each plug-in may be required in case of circular dependencies.
- Dependency from modules can be handled by a register mechanism for ensuring that plug-ins required in order to use a specific module are present.
- Dynamic Loading and Deallocation of Plug-ins in order to preserve system resources. The behavior of the system is affected by the dependency issues outlined in this section, as the modeling paradigm in a specific module may no longer be available. Dynamic loading of plug-ins may prove to be difficult on its own, especially if multi-platform issues are taken into consideration. Therefore, a compiled module, eg. ".dll" or ".so" may only run on a specific platform.

A complete specification of the Plug-in Layer objects is shown in Figure 7.3 on the following page, that illustrates the abstract superclass that describes the interface a plug-in must implement, and the association of plug-ins and dependencies. The methods specified in the plug-in interface include:

- Register - Registers this plug-in with the platform. This call succeeds if the dependencies of the plug-in are met
- Unregister - Unregisters this plug-in, and may cause other dependent plug-ins to unregister also

- ImplementsFunction - Queries a plug-in about whether it implements the specified function, optionally given a version requirement

- getFunctionAddr - Returns the address of the specified function or method as a function pointer, that provides access to the functionality in the plug-in

- getFunctionTable - Returns a hash map containing a description of each function implemented in the plug-in

- Public Functions - A place holder that represents the functions in the plug-in that can be accessed using the getFunctionAddr method

- Private Function - A place holder for any private utility or internal functions used in the plug-in

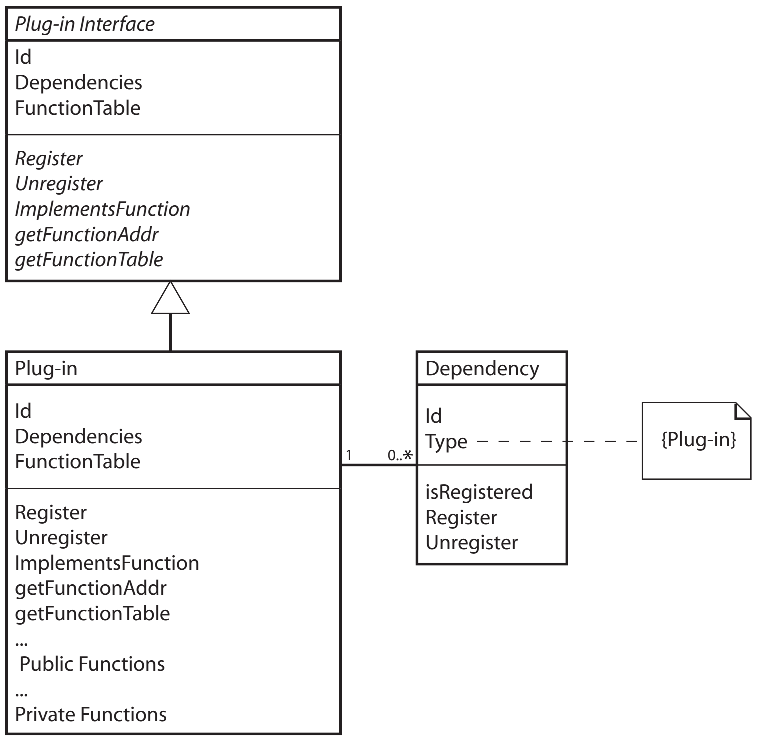


Figure 7.3: The class hierarchy for the Plug-in Layer, including the abstract superclass *Plug-in Interface*, which specifies the interface that plug-ins implement

The dependencies of a plug-in are limited to other plug-ins, and does not include modules and hardware devices, that are a part of the dependencies of a module. This ensures that the implementation of a module may be replaced with another, for instance one that uses additional or alternative plug-ins, without affecting the previously used plug-ins in the system. Furthermore, this approach also enforces the use of plug-ins as classes containing utility functions used internally by modules, that control how the information gathered from hardware devices is utilized.

7.2.3 The ASD API

This section presents the design considerations for the ASD API layer, that is located below the Module and Plug-in layers. The purpose of the ASD API Layer is to facilitate communication between the Module and Plug-in layers, that cooperate in responding to interaction from the user of the system, handle hard disk access, and manage communication with VR Juggler and openNURBS.

Since this layer is the core component of the ASD Platform, it facilitates communication between both the Module and the Plug-in layer and to the underlying layers, that provide connectivity to hardware devices and geometric routines. Plug-ins are however also allowed to utilize some of the geometric routines directly, without involving the API layer, to ensure that performance critical calculations can be carried out with a minimum of overhead. The plug-ins that use this direct form of communication with openNURBS, is typically pieces of reusable and encapsulated code, that the user has no interaction with. Instead plug-ins provide functions such as geometric conversion routines, that are utilized by the modules in the system.

The primary task of the API layer is to run the interactive application that the user interacts with, based on what modules and plug-ins are present, and what input/output devices are available to the user. This means, that besides being an API that facilitates communication between the different layers, it also has the role of an interactive modeling application. As with other interactive VR applications, a number of routines have to be present in order to accommodate the wishes of the user that interacts with the system, including:

- **Responding to input from the users** - further processing of input data received from the VR Juggler layer is handled by the Input component.

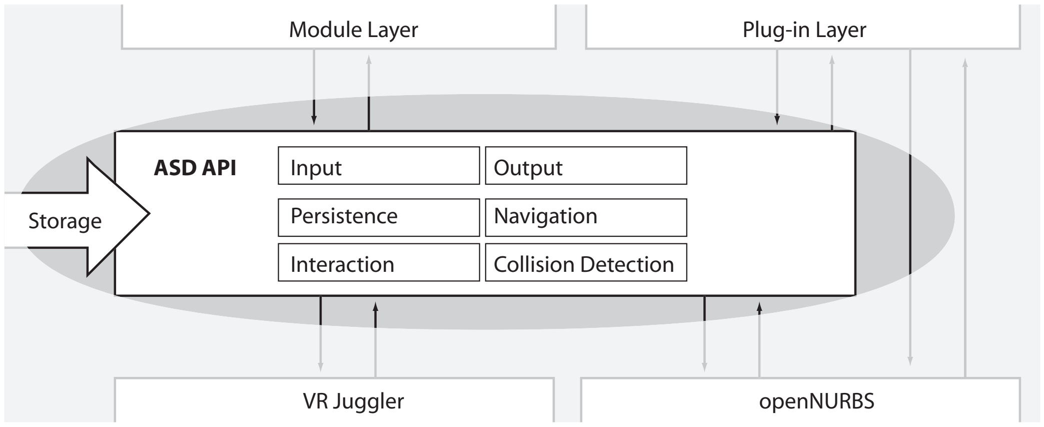


Figure 7.4: A detailed specification of the ASD API Layer, and its internal components

- **Providing the correct output for a given situation and input data** - this includes visualization of the geometric objects in the system and the virtual environment, as well as other kinds of audio or haptic feedback to the user. Handled by the Output component.

- **Interaction interfaces** - routines for the modules and plug-ins to utilize for creating interaction elements, including toolbars, menus etc. is handled by the Interaction component.

- **Navigation** - navigation in the virtual environment, based on the connected input devices is implemented in the Navigation component.

- **Basic geometric routines** - such as the ability to load and save geometry for use with the platform is handled by the Persistence component.

- **Collision detection** - detection of which, if any, objects the user is interacting with is implemented in the Collision Detection component. Collision detection is involved as an important aspect of the modeling process, as well as for selecting the elements of the user interface.

A complete specification of the ASD API Layer is shown in Figure 7.4, that illustrates the internal components of the layer.

The remaining aspects of the interactive modeling application are handled by VR Juggler, that provides functionality for accessing the hardware which is connected to the system. As mentioned earlier we decided to include implementations developed by others, also known as third party modules. The implementations included in our architecture are:

- VR Juggler
- openNURBS

The following sections outline the considerations involved in choosing the third party software that is used in conjunction with our platform.

7.2.4 VR Juggler

We choose to include VR Juggler, mainly to have the advantage of the I/O-handling in VR Juggler which covers a wide range of supported hardware and platforms. This is accomplished by libraries in VR Juggler for the supported hardware. It thereby gives the opportunity to run the 3D application in a simulation mode that can be executed on a desktop computer without special 3D hardware and later in the development process move the application to for instance the CAVE. This is an advantage since it gives the possibility of developing the application almost anywhere. This means that VR Juggler helps us in being able to concentrate on the Module and the Plug-in Layer, which concern the modeling concept we focus on in this project. Furthermore, VR Juggler is fairly well documented, even though it still is work in progress, but the most important things are already documented.

VR Juggler handles the different hardware through configuration files, that tells VR Juggler how to interact with the specific hardware. These configuration files can be edited using a Java based configuration tool, that comes with VR Juggler. The configuration files are loaded into the system, when it is run, as arguments in the command line. This way it is not necessary to recompile the application just because hardware has been exchanged or reconfigured.

Due to the choice of VR Juggler we can, to a certain point, ignore some hardware issues when choosing which hardware that should be compatible with the system. VR Juggler provides an interface to a broad variety of different hardware, such as gloves and wands. It also supports different methods for visualization, like a HMD, the CAVE and simulation on a desktop computer.

When using VR Juggler it is a manageable task to change or add hardware input- and output devices, because of the way VR Juggler handles hardware. Since hardware is relatively easy to change, we will make the considerations about this issue in each Module and Plug-in we design. We do it this way, because the modules might use different devices, and all we have to take into consideration is what VR Juggler outputs when using different hardware.

Event Handling Using VR Juggler

This section describes the utilization and behavior of VR Juggler. In our system VR Juggler will be utilized for:

- fetching input from interaction devices (gloves, wand, pen) and sending it to the API
- providing connectivity to output devices, such as visualization hardware, for displaying the geometry specified by the the API Layer

To control the application VR Juggler has a very strict control loop which controls when the different tasks should be performed. This control loop has the following steps:

1. `init()`
2. `initAPI()`
3. `apiinit()`
4. `preFrame()`
5. `draw()`
6. `intraFrame()`
7. `sync()`
8. `postFrame()`
9. `updateAllData()`

The first three enumerations (1 - 3) are only run at the beginning of execution. These initializes application data and start the VR Juggler API. The last six enumerations (4 - 9) is the loop VR Juggler enters after initialization. One pass through the loop corresponds to one frame visualized. Here it is important to place the tasks, wished to be executed, in the correct steps in the control loop. If misplaced the application may lag[1] more than necessary, because some of the steps are implemented to run in parallel and others are not. In our design we will mainly concentrate on the four steps:

`preFrame()`:
Here tasks that needs to be executed before a frame is visualized should be placed. Heavy calculation should never be in this step, since it will make the execution lag. In our system fetching the input from the interaction devices will be performed in this step. This is not a task with heavy computation and therefore it will not lag the system when performed in this step.

`draw()`:
This step handles the visualization, so we will follow that and perform all the visualization in this step.

`intraFrame()`:
In this step the heavy computations should be placed. This is because this step is executed in parallel with the visualization (`draw()`), hence it will not lag the system. In this step we will calculate all the updates based on the fetched input from `preFrame()`. These updates must not be performed directly on the data used for the visualization, which runs in parallel with this step. It has to be performed on temporary data, which will later be used for updating the data used in the visualization process.

`postFrame()`:
This step will be performed after the visualization is complete. Like in `preFrame()` there should never be heavy calculations in this step. If placed here, the heavy calculations might result in a system that lags more than necessary. Since this step is performed after the visualization it is suitable for updates which had to wait, to avoid intervening in the visualization. In this step we can make the updates on the data which we calculated in `intraFrame()`.

VR Juggler is described more thoroughly in Appendix A.

7.2.5 openNURBS

openNURBS is included to handle the mathematical details of the geometric models. openNURBS is a collection of libraries which concern these mathematical details. This is, like in the case with VR Juggler, a way to concentrate

[1]Experienced by humans when the latency between visualized frames exceeds some threshold, and the illusion of motion graphics can no longer be achieved. This can be caused by less powerful hardware.

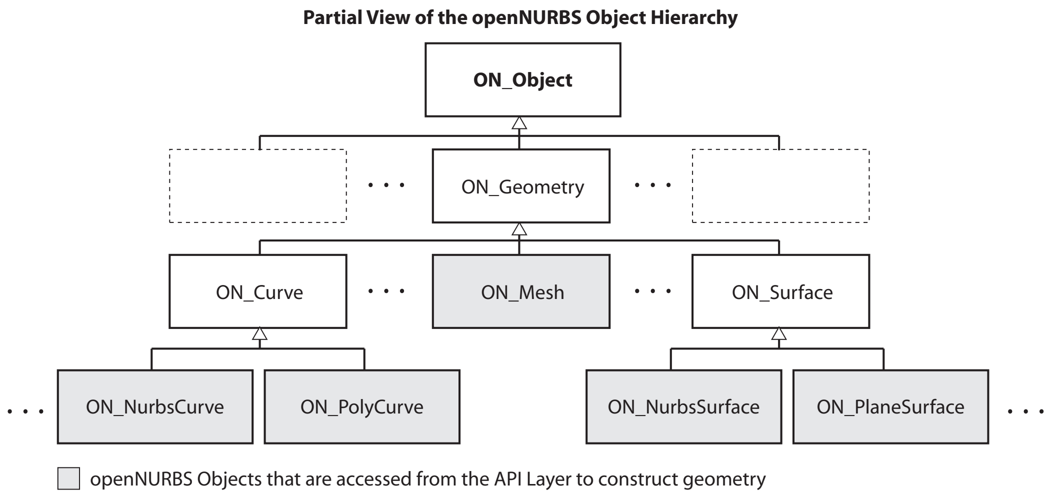

Partial View of the openNURBS Object Hierarchy
ON_Object
...
ON_Geometry
...
ON_Curve
...
ON_Mesh
...
ON_Surface
...
ON_NurbsCurve
ON_PolyCurve
ON_NurbsSurface
ON_PlaneSurface
...
openNURBS Objects that are accessed from the API Layer to construct geometry

Figure 7.5: The hierarchical structure of the openNURBS classes utilized in the ASD platform

on the context we wish to focus on. openNURBS can be downloaded for free from [MA00].

The openNURBS part of Figure 7.1 on page 66 provides us with geometric functions, that can be accessed from the ASD API and from the plug-ins.

The interface between openNURBS and the Plug-in component enables plug-ins to reach more specific functions in openNURBS, precisely as the ASD API. The hierarchical structure of the classes included in the description above is illustrated in figure 7.5.

7.3 Interaction Considerations

An approach for making our application easy to use, is to look closer at the immersiveness of the user. A closer integration of the user with the system, will improve the ease of use, because the user will have the possibility of interacting with the system in a way that resembles interaction with physical objects in real life, compared to how he interacts with a looser integration. This is not always true, for instance a lot of people prefer to play their driving games from a viewpoint placed outside the car instead of the more immersive inside-the-car cam or bumper cam [How], but in our case a one-to-one mapping and a high integration is desired, because we want to allow users with a background in handicrafts to use the system. Therefore, it is in our interest to have a high degree of immersiveness.

A data glove is a way to integrate the user closer with the system, than for instance a mouse and keyboard will do. This is based on the fact that the data glove allows the user to have his hand in the same environment as the object he models. This is not possible with a mouse and keyboard, because these are not integrated in the graphical application.

In a virtual environment the use of the hands directly could be accomplished with data gloves, and even the concept of a primary and a secondary hand could be included. The secondary hand could be used for navigation and to hold the object of interest. The ceramic artist is the only one in our use cases that uses his hands directly for deforming the objects, although the glass-blower comes very close, when deforming the hot glass with a wet newspaper. The approach of deforming the object with the hands seems like a reasonable approach in an virtual environment, because the data gloves give the possibility of using the hands directly, and the fact that we have all used our hands for creating and deforming things at some point in our lives. For the visualization part the desktop monitor offers only a low immersiveness compared to a HMD or the CAVE, this is the case because the use of a desktop monitor isolates the environment the user is placed in from the environment which contains the objects the user interacts with. Both a HMD and the CAVE places the user in the same environment as these objects, which is preferable in our case when focusing on a high degree of immersiveness.

7.3.1 Interaction Levels

Another usability consideration to deal with, is the interaction level in which the user interacts with the system. An example of two different levels of interaction in the same application is AutoCAD[2], where the user can create a drawing by clicking with the mouse directly on the drawing or he can, as an alternative write the numerical values for the wanted positions directly. No matter which of the two approaches the user chooses, the system will receive the numerical input. This means that the program translates the mouse input to numerical values, if the user interacts in this way. If the drawing is created from numerical input the drawing is still depicted in the program to give the user feedback on his work, however the program receive the numerical inputs directly for processing.

The idea of different interaction levels can be seen in figure 7.6. As shown

[2]AutoCAD, Autodesk, Inc.

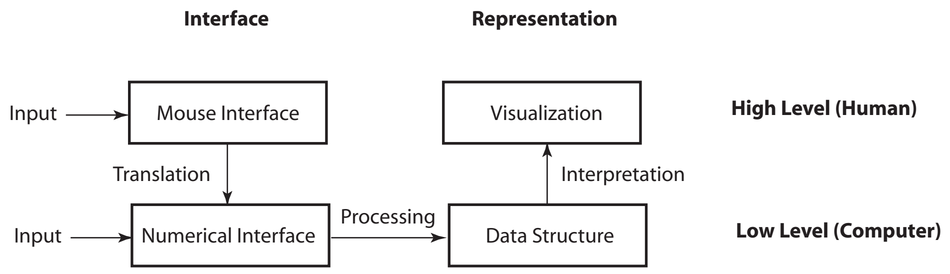


Figure 7.6: Two different approaches for giving an application input and the different representations

in the figure can input be given to the application in two different ways, either through high level interaction (mouse), or through low level interaction (numerical). If the user chooses to use the high level interaction, the input will be translated to numerical before the input is processed. The processing will add the input to the low level representation. This representation is usually not understood by humans, therefore the representation is interpreted and visualized at a high level, which can be understood by humans.

7.3.2 Selecting the Right Kind of Tools

The tools used by the ceramic artist, the glassblower and the 3D artist, can be categorized by usage. All tools are used for construction, deformation, cutting and carving, or navigation no matter which use case we study. This means that we can focus on these overall paradigms when selecting the different interaction approaches in the application.

In a VR application an important consideration is whether potential tools should be represented by a real physical object or by a virtual object. For instance, the physical objects would require the user to have see through HMDs, so the tools can be seen. If this is not the case it somehow seems awkward to represent the tool virtually when it is actually there, because the only consequences of this, is that the virtual and the real placement has to be coordinated very precisely. Also, the user is subject to the physical boundaries of a real tool. The crucial boundary of a physical tool, is that the user is forced to return to the same physical place to pick up the wanted tool. With a virtual tool this problem is non existent, since the tool could simply be in a toolbar from where it could be selected when it is wanted. An advantage of using a physical tool is the familiar feeling of the tool in the hand, but to actually make use of this physical feedback, the system needs to utilize force feedback to provide the user with the feeling of using the tool.

Virtual tools can be more handy to change between than the physical tools, since the change of virtual tool is just a change of mode in the application. This will be simpler than being forced to locate the physical tool, provided that the change of virtual tools is implemented in an appropriate way.

Multiple interaction levels could be used for providing users of varying experience with intuitive interaction methods, which allows users to utilize the interaction method, that best matches their experience and skills with the system. For example, shortcuts could be implemented for choosing a virtual tool, thereby speeding up the design process for experienced users. Shortcuts can be implemented in many ways, from key combinations on the keyboard over icons to gestures by a data glove. Common for all types of shortcuts is that they are an alternative approach for executing a command, and is often preferred by the experienced user. In some cases it can be an advantage to have the option of disabling the shortcuts. An example of this is in the case where the data glove, among other things, is used for shortcuts. In this case the inexperienced user is likely to be annoyed by the shortcuts, if they are accidentally triggered, which can result in unwanted tool shifts. This problem is even more evident if the glove is not precisely calibrated, since this can result in difficulties with recognizing gestures.

7.4 An Artistic Modeling Paradigm on the ASD Platform

This section describes the choice and design of a paradigm for the ASD platform. Even though the platform is designed in a way so it supports several simultaneous paradigms, because of the layered architecture with modules and plug-ins, we will design one paradigm for the platform.

This paradigm will be selected based on the knowledge collected in our case studies (Chapter 3), the guidelines from Chapter 4, and our studies of related work (Chapter 5). We do not believe that we can find a paradigm that covers all the requirements and all the wanted functionality, but this is the main reason for designing the platform to be expandable, so several paradigms could be implemented to cover more of the requirements.

Since only one paradigm will be chosen, it would be an advantage if it was one that fulfilled a majority of the requirements and functionality. It is important that it satisfies as many as possible of the following:

- Visual thinking
- Natural interaction
- Creative freedom
- Flow of thought

This way it aids the user in the "Artistic Shape Design" process, as described in Chapter 1.

We have chosen Surface Drawing, introduced by Steven Schkolne et al. in [SS99], [SPSa] and [SPSb], to illustrate the use of the platform designed in Section 7.1 and 7.2. We believe this paradigm satisfies the topics above. This is because in Surface Drawing there is a one-to-one mapping between the movement of the hand and the creation of new geometry, which in our opinion gives good "Visual Thinking". Furthermore, the input to the system is achieved with a data glove, which we believe to be a "Natural Interaction", especially to artists that are used to working with their hands. Also, we believe that the Surface Drawing paradigm supports "Creative Freedom" and "Flow of Thought", because it in so many ways resemble how real artists work when modeling. The choice of the Surface Drawing paradigm places us in the 3D sketching category, because Surface Drawing is a construction paradigm, without any kind of physical modeling.

The Surface Drawing paradigm can be regarded as very similar to the drawing principles in Chapter 2. The drawing principles could therefore, to some extent, be used for the Surface Drawing paradigm too. A clear distinction between Surface Drawing and traditional drawing, is that traditionally drawing is done on a surface, and Surface Drawing is the creation of the surface itself in 3D space. The drawing process for traditional drawing also goes for Surface Drawing. Surface Drawing is also an iterative process, and the artist needs the possibility of refining the initial surfaces. Furthermore, Surface Drawing is raised beyond a manual skill due to the use of visual thoughts involved in the refinement and use of the minds eye.

At first, Surface Drawing seems to be a method for contour drawing (See Section 2.2 on page 10), but it should indeed be possible to extend it to analytical drawing. This will require the possibility of drawing a volumetric framework. The framework will likely only be used for building guidelines, since the problem from 2D drawing with volumetric objects being flat will be reduced extremely, now where the user draws directly in 3D. In fact all traditional problems with proportions is almost non existing in Surface Drawing, because objects can be placed correctly in all three dimensions directly.

Building on geometry (Section 2.3 on page 11) is possible with Surface Drawing too, using precisely the same principles as in 2D, but in 3D there may be an advantage of using a deformation paradigm instead for the additive form, because it seems natural that an addition of an object could be extruded. The subtractive form leads the thoughts to carving, because this resembles an sculpturing artist starting with a block of stone, and then removing the useless parts and thereby ending up with a piece of art. The complex form, which is a combination of the two forms could be achieved by an approach containing both the deformation and carving approaches.

Even though building on geometry leads the thoughts to carving and deformation, it is, as mentioned, still adaptive to Surface Drawing, with precisely the same principles as in traditional drawing. So, the close relations between traditional drawing and Surface Drawing, and the knowledge that traditional drawing is a very powerful modeling paradigm, means that Surface Drawing is well suited for artistic shape design.

7.4.1 Design of Surface Drawing on the ASD Platform

Designing a modeling paradigm for use with the ASD platform primarily involves two steps, namely the design and implementation of at least one module and a number of plug-ins. As we mentioned earlier, a module is the modeling paradigm itself and the appropriate interface to the remaining parts of the platform, whereas plug-ins are pieces of encapsulated and reusable code, that is utilized by the modules in the system. To understand how Surface Drawing fits together with the notion of modules and plug-ins, we introduce the different concepts that, as a whole, constitute the Surface Drawing paradigm.

Since the underlying mathematical representation of the geometry in Surface Drawing is polygonal, the notion of polygons and vertices is a natural part of the system. In the case of Surface Drawing, the simplest kind of polygons, namely triangles, are used as building blocks for the geometry in the system. Each triangle, which we represent in the class named `Tri`, has three vertices.

Traditionally, vertices only represent a position in three-dimensional space. Instead, we use the notion of a sample, represented by the `Sample` class, that expands this definition to also include orientation. As a result, each vertex in a triangle also contains information about orientation, that comes in handy with some of the geometric calculations in other classes.

Connections between the vertices of a triangle are represented by the `Edge` class, which is a low level class that connects two vertices, of type `Sample`, together. The role of edges in the system, is to implement methods for checking whether a sample object is contained in the edge object, and for retrieving the sample at the other end of the edge. Together, the `Tri`, `Sample` and `Edge` classes are the foundation for the polygonal model used in the Surface Drawing system. Based on these classes, the `Mesh` class implements functions for creating these polygonal meshes. This way, we can use the notion of a mesh, to represent a collection of triangles with associated vertices and edges.

Since the primary means of interaction in Surface Drawing is a glove, we introduce the notion of a `GloveInterface`. `GloveInterface` functions as an interface, that utilizes information from the data glove the user is wearing, in order to respond to the gestures and interaction from the user. Based on this information, `GloveInterface` constructs a series of strokes, which we represent in the `Stroke` class.

The `Stroke` class implements a stroke as a collection of samples, that connects samples into groups of four or five, depending on whether the user is drawing using normal drawing, with five samples from the fingertip to the base of the palm, or detail drawing, that has four samples at the distal end of the index finger. An example of a stroke, with a width of five samples, is shown in Figure 7.7 on the next page.

Finally, the top level object in Surface Drawing is the concept of the current drawing itself, that is represented in the `Drawing` class. The `Drawing` class responds to the samples, that are created by the `GloveInterface`, by gradually merging them with the existing mesh. It also provides functionality for visualization and transformation of the geometry in the current scene of the virtual environment, that the user is immersed in.

In the case of Surface Drawing, the logical approach towards structuring the paradigm, is to place the two classes `Drawing` and `GloveInterface` in the module itself. This matches the guidelines for the purpose of modules, since the `Drawing` class encapsulates the Surface Drawing paradigm, and `GloveInterface` facilitates how Surface Drawing utilizes the information gathered from the connected hardware devices. The remaining classes, namely `Tri`, `Edge`, `Sample`, `Mesh` and `Stroke` fit the profile of plug-ins, that provide the Surface Drawing module with the functions needed for constructing the polygonal meshes and so forth. A complete view of the Surface Drawing paradigm as a module and associated plug-ins is illustrated in Table 7.1.

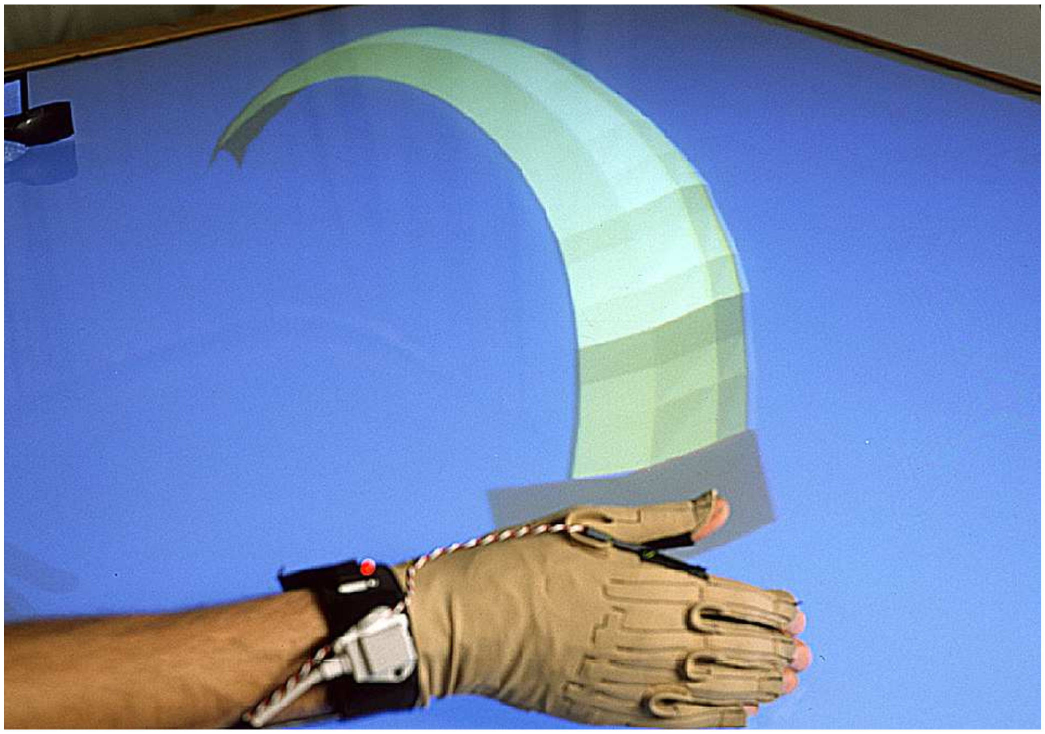


Figure 7.7: The Stroke class represents strokes in the Surface Drawing paradigm, based on information gathered from the connected data glove

Surface Drawing Running on the ASD Platform		
Class Name	Description	Type
Drawing GloveInterface	The Drawing class represents a drawing in three-dimensioal space. The GloveInterface class implements an interface to the information from the data glove hardware.	Module
Mesh	The Mesh class represents a non-manifold polygonal collection of triangles.	Plug-in
Stroke	The Stroke class represents a single stroke performed by the user wearing the data glove.	Plug-in
Tri	The Tri class represents triangles sitting in three-dimensional space.	Plug-in
Edge	The Edge class represents an edge in three-dimensioal space.	Plug-in
Sample	The Sample class represents a vertex in three-dimensional space.	Plug-in

Table 7.1: A schematic view of the Surface Drawing paradigm on the ASD Platform

8 Implementation

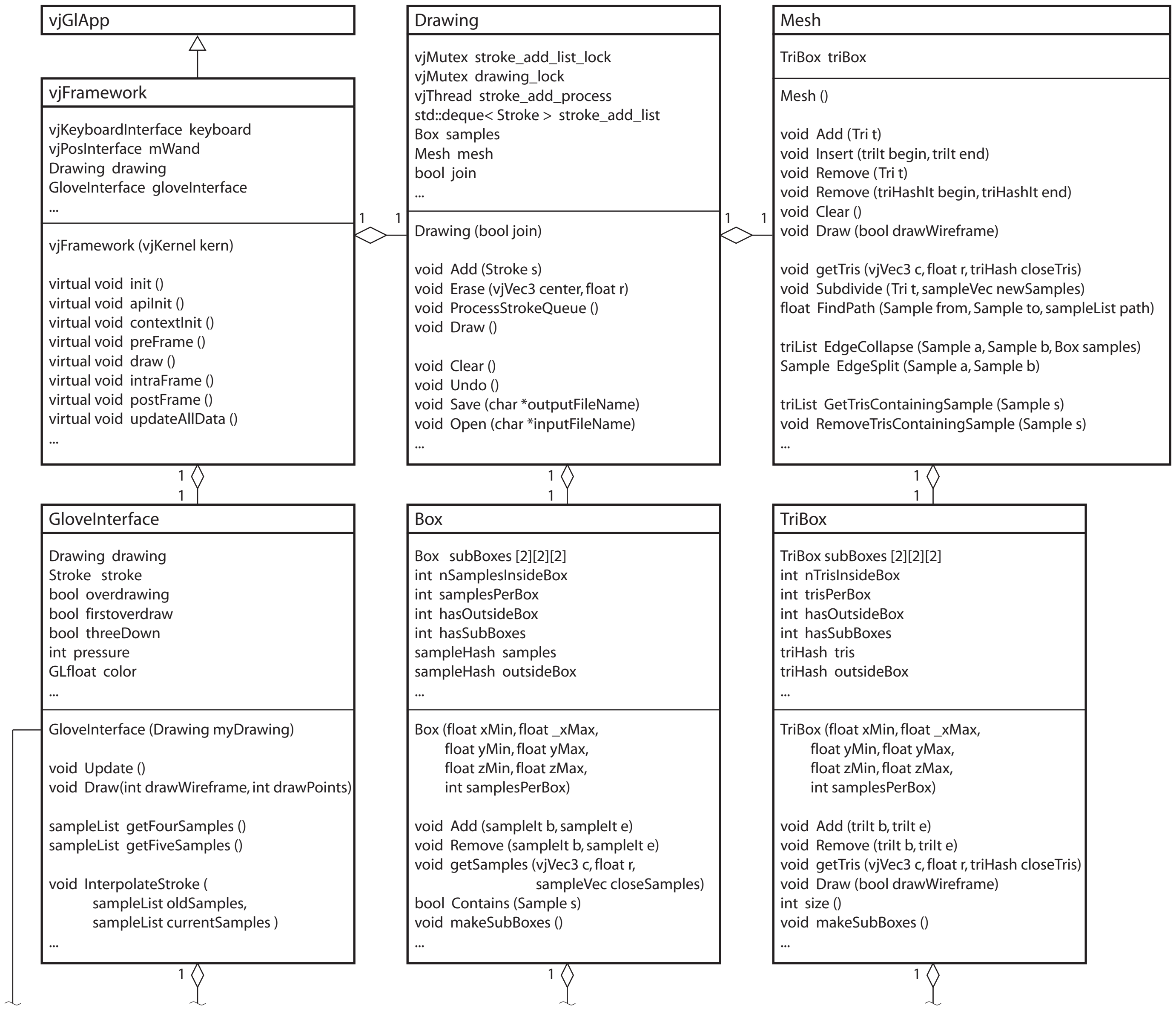
This chapter documents the implementation performed during this semester. Furthermore, we will discuss the results we achieved and the problems we encountered.

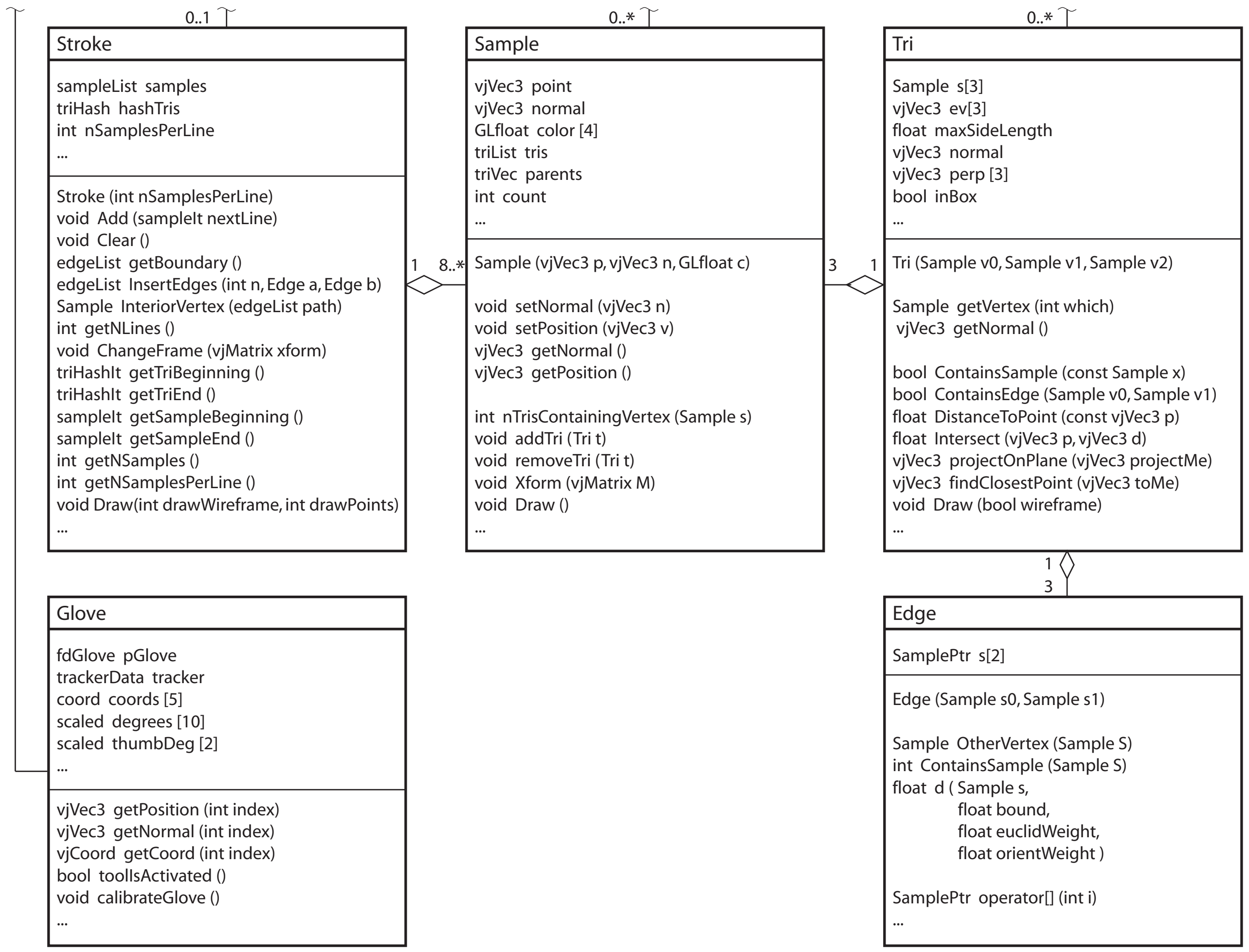
Due to the time period at hand we chose not to follow the designs from Chapter 7 in every detail. This is because we estimated it to take somewhat longer than the time we had available for the implementation. Therefore, we chose to cut away parts of the designed system, namely all of the platform API, openNURBS, the possibility to save/load, and the Module and the Plug-in layers. Instead we will concentrate on the paradigm itself and implement the basics of it, based on VR Juggler.

We know this will have some kind of impact on the proof that the platform architecture was a good idea, but we believe it is more important to prove that paradigms exist, which to some extent fulfill the requirements of an artistic 3D sketching application. We are confident that this application is our Proof-of-Concept of that such a paradigm exists, and if implemented using the platform it would be possible to expand the application with other paradigms to get a system which satisfies most users needing an artistic 3D sketching application.

Our implementation of the Surface Drawing paradigm is to some extent based on parts of an IRIX/Inventor implementation, that was kindly provided by Steven Schkolne[1]. Our primary modifications of this code can be summarized into three areas, including platform specific issues between the IRIX and the Win32 platform, differences between Inventor and VR Juggler, and lastly compiler incompatibilities regarding STL implementations. As a re-

[1]Steven Schkolne (ss@cs.caltech.edu) from the California Institute of Technology


vjGlApp
vjFramework
vjKeyboardInterface keyboard
vjPosInterface mWand
Drawing drawing
GloveInterface gloveInterface
...
vjFramework (vjKernel kern)
virtual void init ()
virtual void apiInit ()
virtual void contextInit ()
virtual void preFrame ()
virtual void draw ()
virtual void intraFrame ()
virtual void postFrame ()
virtual void updateAllData ()
...
Drawing
vjMutex stroke_add_list_lock
vjMutex drawing_lock
vjThread stroke_add_process
std::deque< Stroke > stroke_add_list
Box samples
Mesh mesh
bool join
...
Drawing (bool join)
void Add (Stroke s)
void Erase (vjVec3 center, float r)
void ProcessStrokeQueue ()
void Draw ()
void Clear ()
void Undo ()
void Save (char *outputFileName)
void Open (char *inputFileName)
...
Mesh
TriBox triBox
Mesh ()
void Add (Tri t)
void Insert (triIt begin, triIt end)
void Remove (Tri t)
void Remove (triHashIt begin, triHashIt end)
void Clear ()
void Draw (bool drawWireframe)
void getTris (vjVec3 c, float r, triHash closeTris)
void Subdivide (Tri t, sampleVec newSamples)
float FindPath (Sample from, Sample to, sampleList path)
triList EdgeCollapse (Sample a, Sample b, Box samples)
Sample EdgeSplit (Sample a, Sample b)
triList GetTrisContainingSample (Sample s)
void RemoveTrisContainingSample (Sample s)
...
1
1
1
1
1
1
1
1
1
1
GloveInterface
Drawing drawing
Stroke stroke
bool overdrawing
bool firstoverdraw
bool threeDown
int pressure
GLfloat color
...
GloveInterface (Drawing myDrawing)
void Update ()
void Draw(int drawWireframe, int drawPoints)
sampleList getFourSamples ()
sampleList getFiveSamples ()
void InterpolateStroke (
sampleList oldSamples,
sampleList currentSamples)
...
Box
Box subBoxes [2][2][2]
int nSamplesInsideBox
int samplesPerBox
int hasOutsideBox
int hasSubBoxes
sampleHash samples
sampleHash outsideBox
...
Box (float xMin, float _xMax,
float yMin, float yMax,
float zMin, float zMax,
int samplesPerBox)
void Add (sampleIt b, sampleIt e)
void Remove (sampleIt b, sampleIt e)
void getSamples (vjVec3 c, float r,
sampleVec closeSamples)
bool Contains (Sample s)
void makeSubBoxes ()
...
TriBox
TriBox subBoxes [2][2][2]
int nTrisInsideBox
int trisPerBox
int hasOutsideBox
int hasSubBoxes
triHash tris
triHash outsideBox
...
TriBox (float xMin, float _xMax,
float yMin, float yMax,
float zMin, float zMax,
int samplesPerBox)
void Add (triIt b, triIt e)
void Remove (triIt b, triIt e)
void getTris (vjVec3 c, float r, triHash closeTris)
void Draw (bool drawWireframe)
int size ()
void makeSubBoxes ()
...
1
1
1



```
Stroke
sampleList  samples
triHash  hashTris
int  nSamplesPerLine
...
Stroke (int nSamplesPerLine)
void  Add (sampleIt nextLine)
void  Clear ()
edgeList  getBoundary ()
edgeList  InsertEdges (int n, Edge a, Edge b)
Sample  InteriorVertex (edgeList path)
int  getNLines ()
void  ChangeFrame (vjMatrix xform)
triHashIt  getTriBeginning ()
triHashIt  getTriEnd ()
sampleIt  getSampleBeginning ()
sampleIt  getSampleEnd ()
int  getNSamples ()
int  getNSamplesPerLine ()
void Draw(int drawWireframe, int drawPoints)
...
```

```
Sample
vjVec3  point
vjVec3  normal
GLfloat  color [4]
triList  tris
triVec  parents
int  count
...
Sample (vjVec3 p, vjVec3 n, GLfloat c)

void  setNormal (vjVec3 n)
void  setPosition (vjVec3 v)
vjVec3  getNormal ()
vjVec3  getPosition ()

int  nTrisContainingVertex (Sample s)
void  addTri (Tri t)
void  removeTri (Tri t)
void  Xform (vjMatrix M)
void  Draw ()
...
```

```
Tri
Sample  s[3]
vjVec3  ev[3]
float  maxSideLength
vjVec3  normal
vjVec3  perp [3]
bool  inBox
...
Tri (Sample v0, Sample v1, Sample v2)

Sample  getVertex (int which)
 vjVec3  getNormal ()

bool  ContainsSample (const Sample x)
bool  ContainsEdge (Sample v0, Sample v1)
float  DistanceToPoint (const vjVec3 p)
float  Intersect (vjVec3 p, vjVec3 d)
vjVec3  projectOnPlane (vjVec3 projectMe)
vjVec3  findClosestPoint (vjVec3 toMe)
void  Draw (bool wireframe)
...
```

```
Glove
fdGlove  pGlove
trackerData  tracker
coord  coords [5]
scaled  degrees [10]
scaled  thumbDeg [2]
...
vjVec3  getPosition (int index)
vjVec3  getNormal (int index)
vjCoord  getCoord (int index)
bool  toolIsActivated ()
void  calibrateGlove ()
...
```

```
Edge
SamplePtr  s[2]
Edge (Sample s0, Sample s1)

Sample  OtherVertex (Sample S)
int  ContainsSample (Sample S)
float  d ( Sample s,
           float bound,
           float euclidWeight,
           float orientWeight )

SamplePtr  operator[] (int i)
...
```

Figure 8.1: Class diagram for the Surface Drawing system

sult of these issues, our implementation utilizes VR Juggler event processing and mathematical libraries in place of posix threads and SGI Open Inventor (http://www.sgi.com/software/inventor/). As the hardware devices we utilize in our system also differ from those originally used with the code fragments supplied by Steven Schkolne, the classes for interfacing with hardware such as tracker equipment and data gloves has been implemented from scratch. A more detailed description of the implementation issues we encountered in this process is presented in Section 8.2 on page 93.

8.1 Classes

This section outlines the classes that constitute our implementation of the Surface Drawing paradigm, their role in the system, as well as the primary and most important parts of the interfaces of the classes. A specification of the classes in our implementation is illustrated in Figure 8.1 on the two previous pages.

vjFramework - `class vjFramework : public vjGlApp`
The primary class in the system is the `vjFramework` class, that facilitates the communication between VR Juggler and the remaining classes in the system. The class inherits from `vjGlApp`, in order to provide the VR Juggler kernel with the required event handlers, that execute in specific time slots during program execution. Access to information about input devices is possible through the `keyboard` and `mWand` class members. Upon construction, `vjFramework` initializes an instance of an object of type `Drawing`, followed by a `GloveInterface` object, that is associated with the current drawing.

Drawing - `class Drawing`
The `Drawing` class represents a drawing in three dimensional space. It is the top level object that responds to the samples, that are created by the `GloveInterface` methods. It provides functionality for visualization and transformation of the geometry in the current scene of the virtual environment, that the user is immersed in. A collection of new samples, encapsulated in a `Stroke` object can be added with the `Add()` method. The primary method for responding to new geometry created by the user is the `ProcessStrokeQueue()` method, that merges the recently created geometry with the geometry that is already a part of the drawing. The visualization is performed by the `Draw()` method, that renders the geometry in the current scene. The new geometry that is waiting to be merged with the drawing can be rendered using the `DrawStrokes()` method.

Mesh - `class Mesh`
The `Mesh` class represents a non-manifold polygonal collection of triangles. It implements functions for creating polygonal meshes based on the `Tri` and `Sample` classes. The `TriBox` is utilized for effective storage and retrieval of the triangles in the mesh. The class contains a set of method for working with the triangles in the current drawing, including:

- `Insert()`, that inserts the triangles contained in a triangle iterator into the mesh.
- `getTris()`, that is utilized by the over drawing and eraser tools for determining what triangles are within a certain distance of the tool.
- `Remove()`, for removing triangles from the mesh, as a result of the eraser tool being applied to the mesh.

Additional utility functions, that maintain the infrastructure of the mesh are also implemented.

TriBox - `class TriBox`
The `TriBox` class implements an adaptive spatial data structure for effective storage and retrieval of `Tri` objects. It contains methods for working with large sets and hash sets of `Tri` objects. A `TriBox` object is a hierarchical object that contains a collection of subboxes, that divide the underlying space into seperated blocks. The `Box` provides functionality that is somewhat similar to that of the `TriBox` class, but for `Sample` objects instead.

Tri - `class Tri`
The `Tri` class represents triangles in three-dimensional space. It implements methods for constructing and working with triangles. The vertices of the triangles are references to `Sample` objects. The class contains a collection of methods for manipulating the properties of triangles, as well as a number of utility functions used for geometric calculations.

Edge - `class Edge`
The `Edge` class represents an edge in three-dimensional space. It is a low level class that connects two vertices of type `Sample` together. The class implements methods for checking whether a `Sample` object is contained in the `Edge` object, and for retrieving the `Sample` at the other end of the edge. The `Edge` class also implements a collection of static utility methods for various purposes, such as checking for an edge object in a list of edge objects.

Sample - `class Sample`
The `Sample` class represents a position and orientation in three-dimensional

space, given by two three-vectors. Samples store a list of triangles that they belong to, as this information is useful when deleting triangles. The class implements a collection of methods for getting and setting the position and orientation of the sample, as well as utility functions utilized by higher level geometric methods.

Box - `class Box`
The `Box` class implements an adaptive spatial data structure for effective storage and retrieval of `Sample` objects. It contains methods for working with large sets and hash sets of `Sample` objects, that are created by the `GloveInterface` methods. A `Box` object is a hierarchical object that contains a collection of subboxes, that divide the underlying space into seperated blocks. The `TriBox` class provides functionality that is somewhat similar to that of the `Box` class, but for `Tri` objects instead.

GloveInterface - `class GloveInterface`
The `GloveInterface` class implements an interface to the `Glove` class. It functions as an interface, that accesses the methods in the `Glove` class, in order to respond to the gestures and interaction from the user that wears a data glove. `GloveInterface` constructs a series of strokes (implemented in the `Stroke` class), that it adds to the scene via the `Drawing::Add()` method. The class also has method for tool selection, eg. over drawing and erasing.

Stroke - `class Stroke`
The `Stroke` class represents a single stroke performed by the user wearing the data glove. It implements a stroke as a collection of samples. Samples are connected into groups of four or five samples, depending on whether the user is drawing using normal drawing (five samples from the fingertip to the base of the palm) or detail drawing (four samples at the distal end of the index finger). Samples are added to a `Stroke` object using the `Add()` methods, given an iterator that contains references to a collection of `Sample` objects.

Glove - `class Glove`
The `Glove` class provides access to the data glove and the position tracking equipment. It contains methods to handle the following:

- Initialization of the glove. Connect/open the glove, set calibration, and so on.
- Close the glove connection.
- Initialization of the tracker.
- Close the tracker.

- Read the calibration file (.cal), specified in the program arguments with the '-c' switch.
- Calculating the snap, based on the snap-value specified in the program arguments with the '-s' switch.
- Fetching data from the glove and the tracker.
- Small conversions like deg2rad(), which converts degrees into radians.
- Runtime calibration.
- Drawing the hand/finger.
- Gesture recognition.
- Different data update methods.

The next section presents the implementation issues encountered during the implementation of the application.

8.2 Implementation Issues

As previously stated, our primary modifications of the code fragments supplied by Steven Schkolne, can be summarized into three areas, namely:

- Platform specific issues between the IRIX and the Win32 platform
- Differences between Inventor and VR Juggler
- Compiler incompatibilities regarding STL implementations

Porting parts of the Surface Drawing system from IRIX to Win32 based operating systems posed a number of difficulties. First of all, several of the libraries used in the original code, was IRIX only implementations, such as the SGI Inventor library. VR Juggler provided us with functionality that is somewhat similar to that of Inventor, but changing this aspect of the system involved a number of changes, as we will discuss later in this section. Another platform specific issue is how the operating system handles multi-threading, and the details of the specific libraries for this purpose. In our case, we found it best to allow VR Juggler to handle this aspect of the system, by utilizing the available `vjThread` and `vjMutex` classes. Again, this had implications in the source code, that required us to restructure parts of the source code.

As expected, we found a number of differences in how Inventor and VR Juggler implements geometric routines for matrices and vectors. The VR Juggler library uses a somewhat more object oriented approach for working with vectors and matrices, than that of Inventor. For instance, to transform a vector `center` by a matrix `xform` and store the result in vector `res`, we use:

```
res.xformFull(xform, center);
```

whereas Inventor uses the notion of source and destination vectors, for the transformation to be carried out:

```
xform.multVecMatrix(center, res);
```

Fortunately, in most cases, the two libraries pretty much agreed on what kind of functionality in the vector and matrix classes, that is required for computer graphics and geometric modeling.

Another major implementation issue was due to a number of compiler incompatibilities, regarding the STL implementation that the specific compiler utilizes. In our case, parts of the original source code presented problems when compiled with the windows based Visual Studio environment. One of the problems was the lack of simple and associative hashing containers, such as hash maps and hash sets. Fortunately, the .NET version of Visual Studio presented a solution to this problem, since the STL implementation in this version now included hashing containers and other STL extensions. However, VR Juggler was not compatible with this version, resulting in conflicts during the compilation process of our system. A final solution to this problem was to recompile the entire VR Juggler library from scratch in the .NET environment.

8.3 The Application

Although the application we implemented does not rely on the designed ASD platform, we were able to demonstrate that the application is in fact capable of supporting interactive 3D modeling and artistic shape design. Multi threading, combined with effective hashing adaptive spatial data structures,

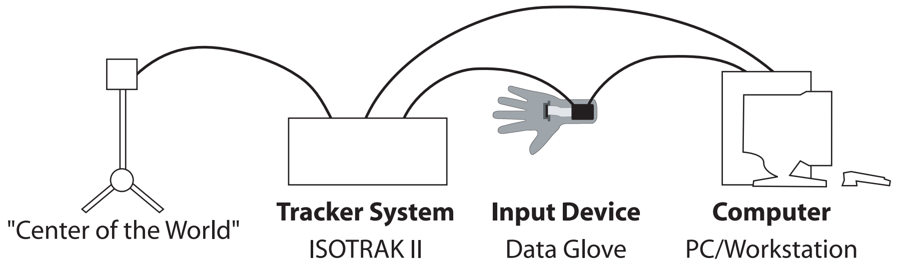


Figure 8.2: The hardware setup used in the application

allows us to run the application on a desktop computer, at frame rates that provide the user with real time continuous visual feedback, as they incrementally build up geometry.

The hardware setup, illustrated in Figure 8.2, includes a desktop computer, with an OpenGL accelerated graphics adapter. A data glove is connected to this computer, in order to send information about the flex of each finger to the application, that responds accordingly. Furthermore, an ISOTRAK II tracker system, that is also connected to the computer, provides information about the position of the data glove, that is relative to the location of the "Center of the World", which is also connected to the tracker system. The visual feedback is provided by the monitor of the desktop computer, which unfortunately means that no depth perception is possible. For this purpose, a HMD mounted with at second tracker could be used, in order to provide users with full dimension visualization. Alternatively, the CAVE could be used.

The following sequence of figures demonstrates some of the tasks, that can be performed with the system. The four cylinders is the graphical representation of the hand in the virtual environment. Figure 8.3 on the next page shows the most basic action in the system, namely the creation of a surface, that is the result of a simple motion of the hand. The purple color of the geometry indicates that the user is currently drawing, and in this example extending the length of the stroke.

The next illustration, in Figure 8.4 on the following page, shows four surfaces, or strokes, that cross each other at approximately perpendicular angles, in order to demonstrate the freedom that users have for creating surfaces.

Figure 8.5 on page 97 demonstrates, that the user has perfect control of the surfaces, and is thereby able to create intricate shapes by the movement of the hand, much like in the case of freehand sketching.

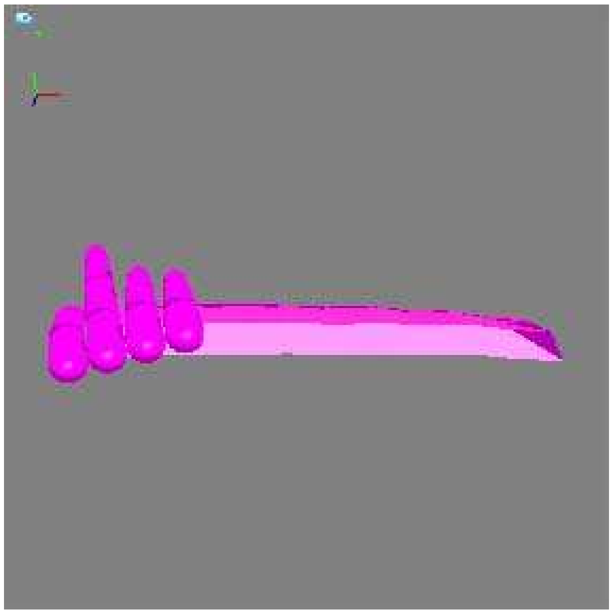


Figure 8.3: The most basic action in the system, namely the creation of a surface

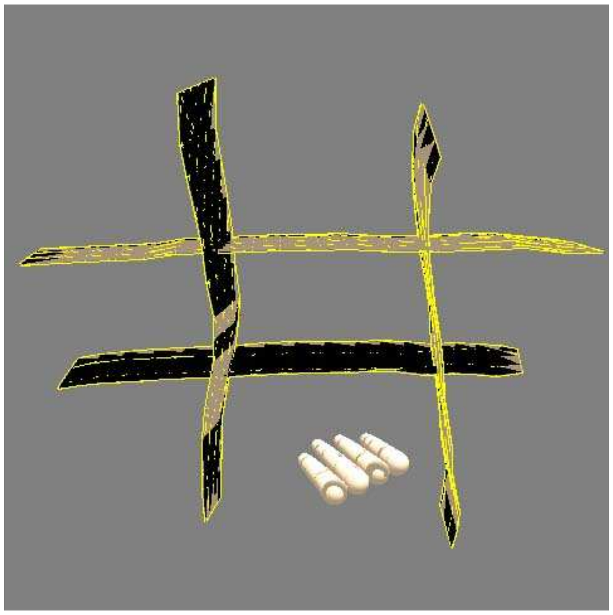


Figure 8.4: Four surfaces, or strokes, that cross each other at approximately perpendicular angles

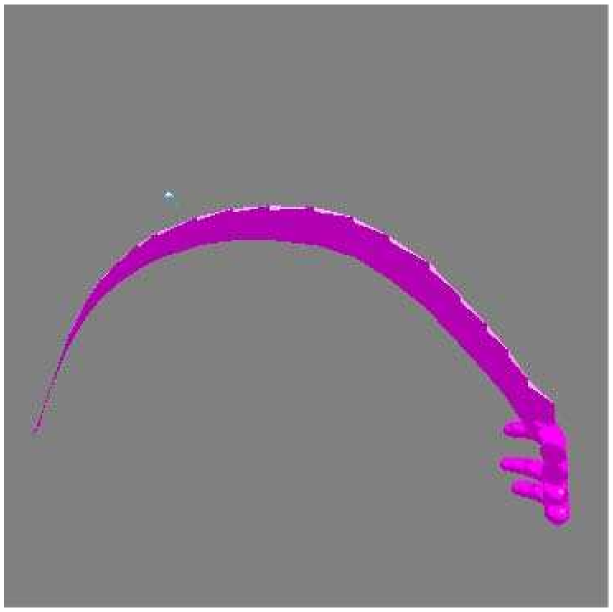


Figure 8.5: Creation of intricate shapes by movement of the hand in 3D space

In Figure 8.6 on the following page, the user created an new surface while flexing the fingers, in order to create the profile shown in Figure 8.7 on the next page.

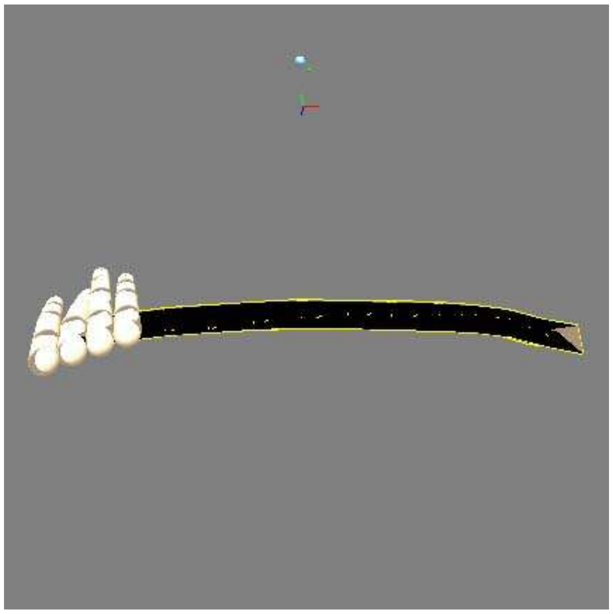


Figure 8.6: User creating a surface while the fingers are flexed

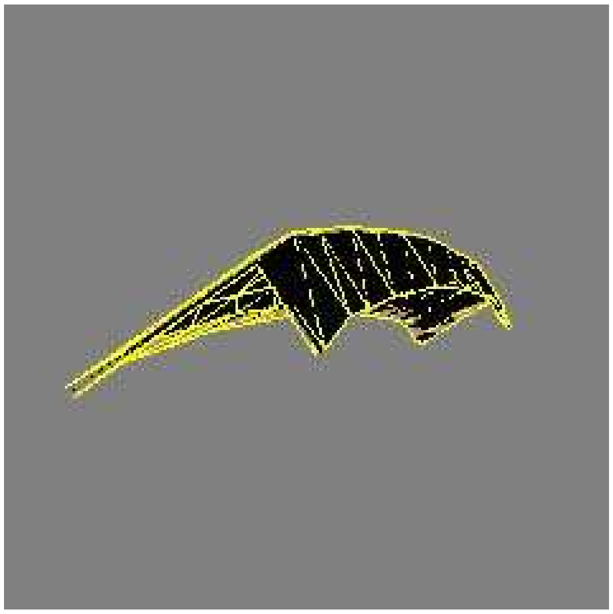


Figure 8.7: A profile of the surface created with the fingers flexed in Figure 8.6

9 Design & Implementation Summary

This chapter summarizes the Design & Implementation part, and elaborates upon the topics discussed in this part.

In our studies of related work we discovered that it was hard to find one paradigm which fulfilled every possible aspect of artistic shape design. For this reason we chose to design a platform which should make it possible to include more paradigms which, as a whole, could span the entire context of artistic shape design.

Due to lack of time, we did not implement all of the designed system. We still believe that the implemented part proves that the Surface Drawing paradigm can be used in relation to artistic shape design. We chose Surface Drawing as a module directly implemented for VR Juggler, because it provides us with an application which can be regarded as our Proof-of-Concept, which concerns sketching in 3D. Still, as future work it could be very interesting to implement all of the platform, to prove that it really works as it was intended. Furthermore, it would be necessary to design and implement other paradigms on the platform.

Our choice of using VR Juggler should to some extent support portability to other hardware platforms. It would be interesting in future work to move the implementation to a 3D environment, like the CAVE.

During the implementation of our application, we realized that the hardware had difficulties with precision regarding tracking and calibration. Especially the calibration of the glove caused some frustration, since the optical fibres in the glove for measuring the flex and the spreading of the fingers affects each other. This causes the glove to be rather unpredictable, since the influence between the optical fibres somehow seems random. The box where the optical fibres are connected is not firmly mounted on the glove, which means it can tilt and we therefore had to use a solution which was not optimal, namely to hold it against the hand using a rubber band. Furthermore, the glove fits better on some hands than others. This is especially a problem if the user has small hands, since the glove can very easily slip from the initial and intended position. The ability to calibrate the glove by using a calibration file generated for the individual user, helps the glove in recognizing the flex and the spread of the fingers.

Beside the flex and spread of the glove, we also had to relate to the capabilities of our tracking device. The area in which tracking is possible, is about 1.5 meters from the box marked as "center of the world". This gave us a sphere with a diameter of 3 meters to interact in. This may sound as plenty of space, but the tracking of the glove did not work close to the "center of the world" and has a very little precision at the peripheral part of the sphere. These circumstances resulted in a limited area in which interaction was possible. Furthermore, azimuth rotation had a weakness when the user violates specific thresholds, because the azimuth value changed to the opposite sign. Still, we believe it might be possible to implement a solution to solve this problem, since the problem is consistent.

Because of the inaccurate hardware we do not believe a usability test would be possible to perform based on the implemented system. We believe the test users would, to some extent, blame the application for the problems rather than the hardware. If better hardware could be used it would be natural, as future work, to perform a usability test of the entire system.

We executed our application on a desktop computer with a AMD[1] 1100 Mhz CPU and a Matrox[2] G400 graphic card. This hardware was capable of executing our application in real time, however we experienced a lag of approximately one second, when the application merged geometry from two or more strokes. As our application executes in threads, we expect that a system with multiple CPUs will be able to execute even the geometry merge in real time.

[1]AMD, One AMD Place, Sunnyvale, California 94088-3453
[2]Matrox Graphics Inc., 1055, boul. St-Régis, Dorval (Québec), Canada H9P 2T4

Part

3

Future Work & Conclusion

This part contains the work we identify which could be interesting in continuation or as a supplement to this project. We then conclude upon the results gained through this project.

10 Future Work

This chapter describes the future work which we identified through the report. This includes not only work which is in continuation of this project, but also work which could function as supplementary to this project, to support our choices and decision.

Through the analysis we discovered how important it was to study how potential users worked. Unfortunately, our choice of artistic trades all worked around an axis, which indirectly influenced the tasks we asked the 3D graphic designer to perform. It would be a good idea, as a supplement to this project, to study other artistic trades, such as sculptors and painters. We believe this would give an even better knowledge base. Furthermore, studies of more related literature would be an important since this area moves forward with an astonishing pace.

Also, it could be interesting to design more paradigms to the ASD platform to check if the interface between the platform and the modules and the plug-ins was general enough to support other paradigms. This would naturally lead to the implementation of the entire platform and all the designed paradigms. Again, to see if it actually works as it is supposed to or the platform has to be redesigned to fulfill the requirements. The implementation would also show if the platform has an overhead which would result in problems regarding the execution time.

Another interesting thing to perform as future work, is to move the implemented system from Windows to a platform which gives the opportunities to execute the application in the CAVE or another 3D environment. This should be possible since the system is based on VR Juggler and platform dependent issues in the source code is kept at a minimum.

Furthermore, it could be interesting to try the application with newer and/or better hardware, such as the data glove and the tracking system. Also, to perform a thorough calibration of the hardware would help on the system performance and would therefore be interesting as future work. This would then also make a usability test possible, because the test users better would be able to evaluate the system as a whole, instead of criticizing the system for the flaws in the hardware.

It could also be interesting to expand the system so the user had both hands in the environment. This could be achieved by adding another data glove to the system. The user would then be able to perform the primary tasks, such as drawing and erase, with the primary hand, and use the secondary hand to other tasks such as navigating in the system and rotating objects.

Also, as our case studies reveled, special functions, such as revolve, would have some use in specific cases. A function like revolve could be implemented in different ways. One would be to take the function as it is implemented in 3D modeling applications like Maya. Here the user "draws" a profile of the desired object, marks an axis and then revolves the profile around the axis. Another approach is it implement the ceramic artist's potter's wheel and then let the user use it as in the real world, just creating surfaces by holding the hand still and then rotating the wheel.

11 Conclusion

Our studies of drawing principles provided us with a good insight into how sketching works in two dimensions. We have shown that these principles can also be beneficially used for sketching in three dimensions, thereby utilizing the powerful modeling paradigm that the paper and pencil is, in interactive 3D modeling applications for artistic shape design. Together with a series of cases studies, drawing principles provided us with knowledge, that was crucial in the understanding of how computerized tools can be improved, in order to provide artists with previously unseen degrees of creative freedom and artistic expression. The results include an improved understanding of the work processes of artists and their tools, as well as a discussion of common and shared concepts between ceramic artists, glassblowers and 3D graphic artists. One of the important discoveries in this project, is that our studies have shown that major contributions to the research of interactive modeling application comes not only from technological advances, but also from an improved understanding of the processes for which we are developing computerized tools.

Based on these observations we presented a number of guidelines for interactive modeling applications for artistic shape design, with focus on functionality and interaction, followed by a recommendation of the appropriate libraries for the underlying platform of these applications. Together with the observations, these guidelines were useful for an investigation of related work, that contributed with a thorough discussion and evaluation of existing technologies for interactive modeling with focus on artistic shape design.

Given the knowledge obtained in these investigations, we realized that no single modeling paradigm had the versatility that artist expect from their tools, which emphasize visual thinking, natural interaction, creative freedom and flow of thoughts. For this reason we designed a platform that supports a multitude of modeling paradigms, based on the notion of module and plug-in interfaces. This ASD Platform allows developers to focus on the modeling paradigm itself, without worrying about implementation issues that are not directly related to the modeling paradigm. Furthermore, simple connectivity to a variety of hardware, including interaction devices, visualization and so forth is provided through the of use of an API layer based on VR Juggler. The platform also supports parallel representations of polygonal and parametric based geometric data, which enables developers to choose the representation that is best suited for the specific use context, be it the implementation of a construction paradigm or deformation paradigm.

Next, we presented a design of a modeling paradigm called Surface Drawing, for use with the ASD Platform, in order to show an application of the platform, and that the notion of modules and plug-ins is useful for structuring a modeling paradigm. With drawing principles and the case studies in mind, we also argued that Surface Drawing is an excellent tool for artistic shape design, in that is has a close relation to drawing principles, intuitive use, and provides control of intricate shapes, without the complexity of the traditional computerized modeling applications. Most importantly, the Surface Drawing paradigm allows the artists to think directly in the terms of the shape they are trying to create, while their movements of the hand is directly linked with the perception of the shape.

Lastly, we implemented a subset of the functionality of the Surface Drawing paradigm, in order to demonstrate its usefulness for artistic shape design purposes. From these experiences we learned that hardware devices for interaction, such as position tracking systems and data gloves, still have a long way to go, before they can provide data that matches the precision of human fine motor skills. The performance of our implementation of the Surface Drawing paradigm, makes it possible to run the application using a desktop computer, when connected with a tracking system and a data glove, and get real time visual feedback while incrementally building geometry by simple movements of the hand.

As the cost of computing power decreases, and better hardware technologies for interaction devices emerge, we believe that some artist will begin to embrace computerized modeling tools, and utilize the flexibility and advantages of digital representation and modeling, as it begins to provide them with the intuitiveness, creative freedom and artistic expression they enjoy from powerful media such as sketching and physical modeling. In this context, we see great potential and promising possibilities in the Surface Drawing paradigm, that has managed to capture the essence of sketching by bringing it to three dimensions, allowing artists to think perceptually about the creation of shapes in their full dimension.

As we stated in the introduction, we believe, that an interactive modeling application that exhibits these properties can be useful in many contexts, including the role of a support tool for artists, but also as the primary tool for the creation of virtual art, that creates new possibilities, such as viewing and collaboration of art across physical boundaries. Eventually, people will find other uses for this kind of application, possibly for entertainment purposes much like they enjoy sketching, painting, and using physical media like clay. At some point, the entertainment industries may also begin to use this kind of application, eg. for game- and character design and special effects in movies, and even for rapid evaluation and creation of ideas and shapes in industrial design.

Bibliography

[B´86] P. Bézier. The mathematical basis of the unisurf cad system. Technical report, 1986. Butterworth and Co.

[CJ98] Francis D. K. Ching and Steven P. Juroszek. *Design Drawing.* Van Nostrand Reinhold, 115 Fifth Avenue, New York, NY 10003, 1998. ISBN: 0-442-01909-2.

[CMZ+99] Jonathan M. Cohen, Lee Markosian, Robert C. Zeleznik, John F. Hughes, and Ronen Barzel. An interface for sketching 3d curves. Technical report, Brown University Site of the NSF Science and Technology Center for Computer Graphics and Scientific Visualization, Providence, RI 02912 and PIXAR, 1999.

[DBW+00] Joachim Deisinger, Roland Blach, Gerald Wesche, Ralf Breining, and Andreas Simon. Towards immersive modeling - challenges and recommendations: A workshop analyzing the needs of designers. Technical report, Fraunhofer Institute, 2000.

[Dic02] Dictionary.com. Dictionary.com, March 2002. http://www.dictionary.com.

[DM93] Bo Dahlbom and Lars Mathiassen. *Computers in context - The philosophy and practice of system designs.* Blackwell Publisher, Cambridge, Massachusetts, 1993. ISBN: 1-55786-405-5.

[GD01] James E. Gain and Neil A. Dodgson. Preventing self-intersection under free-form deformation. Technical report, Collaborative Visual Computing Laboratory, Computer Science Department, University of Cape Town and Rainbow Graphics Group, Computer Laboratory, University of Cambridge, October-December 2001.

[GEL00] Arthur D. Gregory, Stephen A. Ehmann, and Ming C. Lin. in-touch: Interactive multiresolution modeling and 3d painting with a haptic interface. Technical report, Department of Computer Science University of North Carolina, Chapel Hill, NC 27599-3175, 2000.

[How] Geoff Howland. When immersiveness goes bad. http://www.lupinegames.com/articles/badimmerse.html.

[HZTS01] Shi-Min Hu, Hui Zhang, Chiew-Lan Tai, and Jia-Guang Sun. Direct manipulation of ffd: Efficient explicit solutions and decomposible multiple point contraints. Technical report, Department of Computer Science and Technology, Tsinghua University, Beijing 100084 and Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong, 2001.

[IMT99] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: A sketching interface for 3d freeform design. Technical report, University of Tokyo and Tokyo Institute of Technology, 1999.

[MA00] Robert McNeel and Associates. opennurbs™ initiative, December 2000. http://www.opennurbs.org.

[MCCH99] Lee Markosian, Jonathan M. Cohen, Thomas Crulli, and John Hughes. Skin: A constructive approach to modeling free-form shapes. Technical report, Brown University site of the NSF and Technology Center for Computer Graphics and Scientific Visualization, Box 1910, Providence, RI 02912, 1999.

[SPSa] Steven Schkolne, Michael Pruett, and Peter Schröder. Surface drawing: Creating organic 3d shapes with the hand and tangible tools. Technical report, Caltech Department of Computer Science.

[SPSb] Steven Schkolne, Michael Pruett, and Peter Schröder. Surface drawing: Direct creation of freeform shape (white paper kindly supplied by steven schkolne). Technical report, Caltech Department of Computer Science.

[SRS91] Emmanuel Sachs, Andrew Roberts, and David Stoops. 3-draw: A tool for designing 3d shapes. Technical report, Massachusetts Institute of Technology, November 1991.

[SS99] Steven Schkolne and Peter Schröder. Surface drawing. Technical report, Caltech Department of Computer Science, 1999.

[Tea01] VR Juggler Team. Programmer's guide, April 2001. http://www.vrjuggler.org.

[USV96] M. Usoh, M. Slater, and T. I. Vassilev. Collaborative geometrical modeling in immersive virtual environments. Technical report, 1996. In 3rd Eurographics Workshop on Virtual Environments.

[WP98] Robert Wilt and Inge M. Poulsen. Multilingual dictionary of pottery words - english/danish, December 1998. http://www.dinoclay.com/info/dict/mdpwenda.html.

Part 4 Appendices

This part contains a more detailed description of VR Juggler, than we have inside the report, and a presentation of the concepts of Virtual Reality in general.

A VR Juggler

This section describes the basics of how VR Juggler works and how to implement applications using VR Juggler. More information on the subject can be found at [Tea01].

A.1 Overview

VR Juggler is a platform independent development tool for creating VR Applications. It is independent because the developers of VR Juggler has made all the necessary libraries for a lot of platforms. Furthermore the configuration tool for VR Juggler is implemented in Java, a platform independent programming language.

VR Juggler is currently supporting:

- IRIX
- LiNUX
- Windows NT
- Solaris
- FreeBSD
- HP/UX

Furthermore VR Juggler supports a wide variety of VR hardware, such as tracking systems, data gloves and input devices. Also the visualization can be performed on almost anything from the expensive CAVE to the cheaper shutter glasses combined with a monitor.

All the different hardware have their own configuration file, which describes how VR Juggler shall communicate with the device. These configuration files can be edited with the configuration tool that comes with VR Juggler. This tool is based on Java to make it platform independent.

Developing applications in VR Juggler is a little different compared to ordinary application development. In VR Juggler all applications are objects that VR Juggler uses to create the virtual environment in which the user interacts. The application object implements interfaces needed by VR Juggler's virtual platform to create the environment (See Figure A.1).

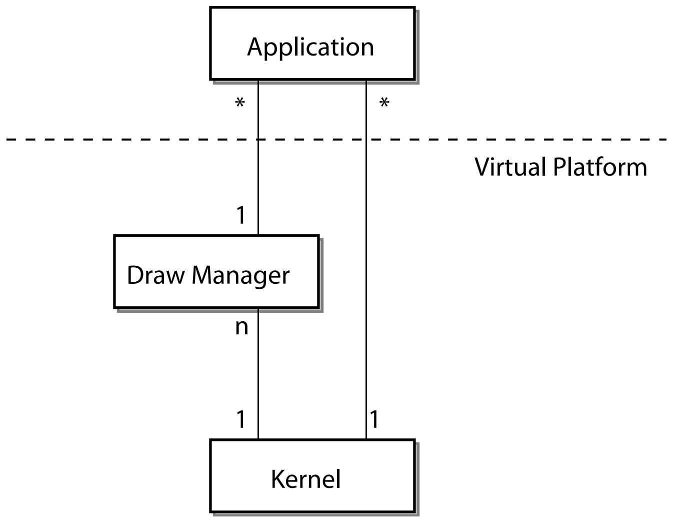


Figure A.1: Application object and virtual platform [Tea01]

This means that, unlike ordinary applications, these application objects do not have a `main()` method as entry point to the program. Usually `main()` is the method called by the Operating System (OS) when the application is requested to start running. The OS then handles the scheduling of all running applications so they each get some processing time. In VR Juggler this scheduling is handled by the kernel of VR Juggler by invoking the methods of the application object.

The interfaces for application objects are derived from the hierarchy shown in Figure A.2. The kernel expects the base class `vjApp` interface to be implemented, which among other things specifies the initialization and shutdown of the application object. Furthermore there is a base class for each draw manager interface (`vjPfApp`, `vjGlApp`, etc.).

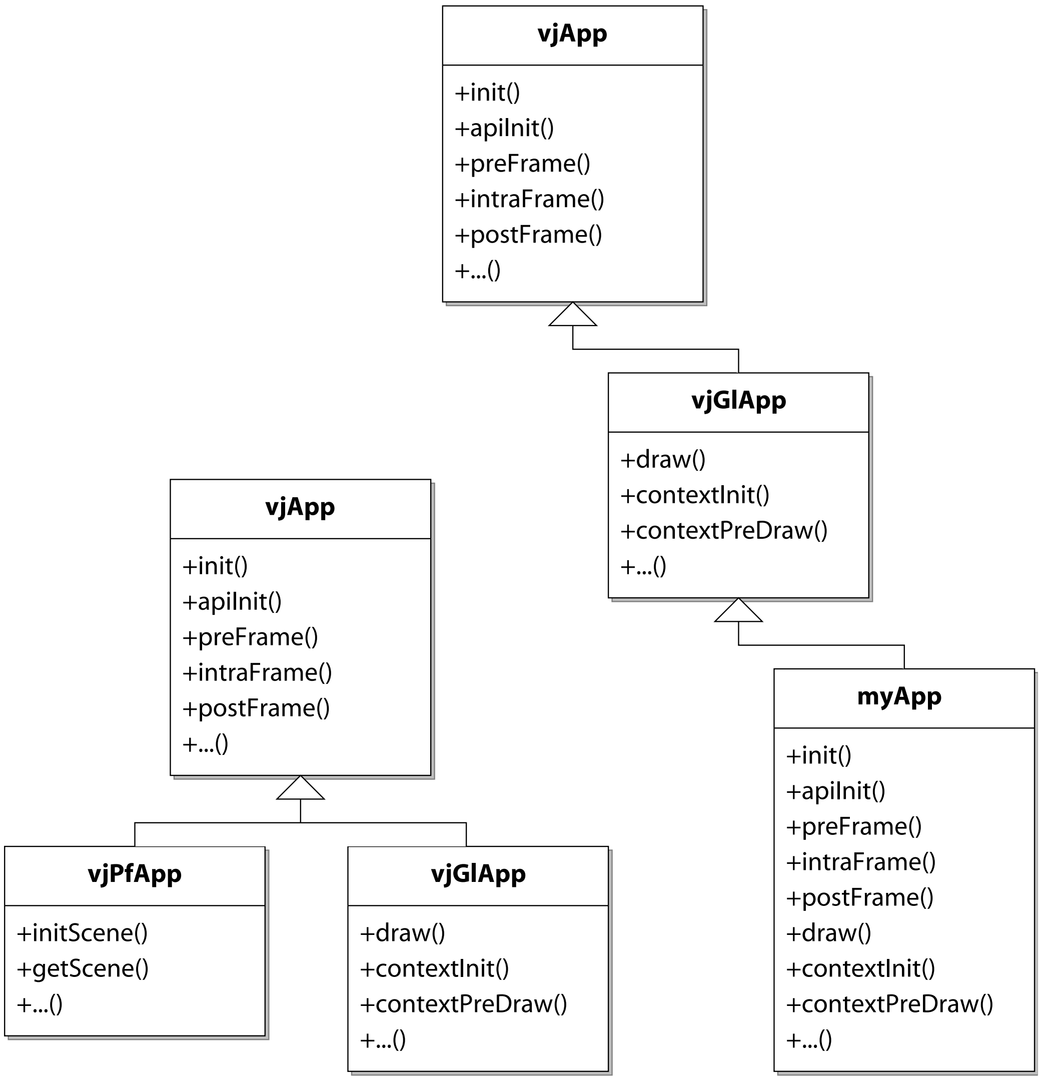


Figure A.2: vjApp hierarchy [Tea01]

The system expects that all application objects implement these methods, that the kernel uses to interact with the application object. Developing applications in VR Juggler is not much else than “filling in the blanks”. If a developer does not implement all of the methods a default implementation will be used, usually this default implementation is empty (Does nothing).

It was not exactly true when said that VR Juggler applications did not have a `main()`. The applications do not have a `main()`, but of course the system needs a `main()` to start. The only thing this `main()` does is to start the kernel and then pass an application object to the kernel.

A simple `main()`, looks something like this:

```
#include <simpleApp.h>

int main(int argc, char* argv[])
{
   vjKernel* kernel = vjKernel::instance();    // Get the kernel
   simpleApp* app = new simpleApp();           // Create app object

   kernel->loadConfigFile(...);                // Configure kernel

   kernel->start();                    // Start the kernel thread

   kernel->setApplication(app);        // Give application to kernel

   while(!exit)
   {
     // sleep
   }
}
```

A quick walk through of the `main()`:
First we find (and possibly create) the VR Juggler kernel and store this handle for future reference. Then we instantiate a copy of the user application object, in this example `simpleApp`. It is worth noticing that we include the header file that defines the simpleApp class.

Next we load the configuration files, which is achieved by calling the kernel's `loadConfigFile()` function. Now we can start the kernel by calling the kernel's `start`. Finally we can give the kernel an application object to run.

A.2 VR Juggler Helper Classes

In VR Juggler there are some helper classes which often are used when writing a VR Juggler application. Some of these are:

- vjVec3 and vjVec4

- vjMatrix

The classes vjVec3 and vjVec4 are designed to work the same way as three- and four dimensional mathematical vectors. Meaning that vjVec3 can be compared to a vector consisting of three elements (`<x, y, z>`). Whereas vjVec4 is the same as a vector consisting of four elements (`<x, y, z, w>`)

There is implemented a simple-to-use interface for performing standard vector operations. For a single vector these operations are available:

- Inversion (changing the sign of all elements)
- Normalization
- Multiplication by a scalar
- Division by a scalar
- Conversion to a Performer vector

For two vectors these operations are possible:

- Assignment
- Equality/inequality comparison
- Dot product
- Cross Product
- Addition
- Subtraction

It is possible to work on the vector as a whole or on the elements individual. As an example of this consider setting a vjVec3 to some value. When setting the whole vector at once it is done like this:

```
vjVec3 vec1;

vec1.set(x, y, z);
```

The same result can be achieved by setting the elements individually:

```
vjVec3 vec1;

vec1[0] = x;
vec1[1] = y;
vec1[2] = z;
```

Or if the values are known at instantiation time, it can be done instantly:

```
vjVec3 vec1(x, y, z);
```

The class vjMatrix work in much the same way as vjVec3 and vjVec4, but for more information on the precise syntax consult [Tea01]

A.3 The Control Loop

To control the application VR Juggler has a very strict control loop which controls when the different tasks that should be performed. This is shown in Figure A.3 on the facing page.

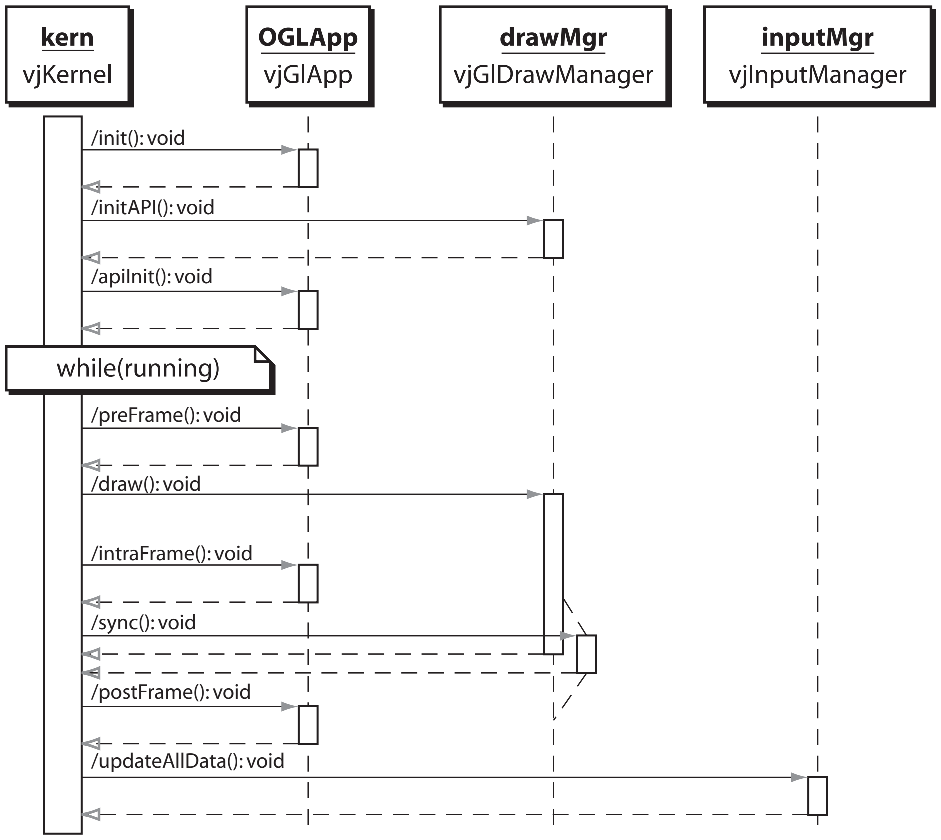


Figure A.3: The kernel control loop

This control loop has the following steps:

1. `init()`
2. `initAPI()`
3. `apiinit()`
4. `preFrame()`
5. `draw()`
6. `intraFrame()`
7. `sync()`
8. `postFrame()`

9. `updateAllData()`

The first three enumerations (1 - 3) is only run at the beginning of execution. These initializes application data and start the VR Juggler API. In these control steps tasks that has to be run before the visualization starts needs to be placed. In the first one (`init()`), tasks that has to be carried out before the graphic starts has to be placed. The next two (`initAPI()` and `apiinit()`) starts the graphical API and the system API, respectively.

The last six enumerations (4 - 9) is the loop VR Juggler enters after initialization. One pass through the loop corresponds to one frame visualized. Here it is important to place the tasks, wished to be executed, in the correct steps in the control loop. If misplaced the application may lag more than necessary, because some of the steps are implemented to run in parallel and others are not.

Below here are these six steps described in more detail:

`preFrame()`:
Here tasks that needs to be executed before a frame is visualized should be placed. Heavy calculation should never be in this step, since it will make the execution lag. This could prepare data needed in the visualization or in calculations which will performed later.

`draw()`:
This step handles the visualization. It runs in parallel with `intraFrame()`. No calculations should be placed here, only functions used to visualize, eg. OpenGL commands.

`intraFrame()`:
In this step the heavy computations should be placed. This is because this step is executed in parallel with the visualization (`draw()`), hence it will not lag the system. This could eg. be calculations that prepares the data for the next frame to be visualized. Since this method is run in parallel with `draw()` it is important that the data which the work is performed on isn't the same as the data used for visualization. If it is not possible to have two data structures or if it is very space consuming there must be placed a Mutex lock on the critical parts.

`sync()`:
`sync()` is used to synchronize the system after both `draw()` and `intraFrame()` has completed their execution.

`postFrame()`:
This step will be performed after the visualization is complete. Also here

there should not be placed heavy calculations, since this also could lag the system more than necessary.

`updateAllData()`:
Here updates of data should be. Often updates from input devices and other devices would be placed here.

B Virtual Reality

This appendix provides an overview of VR, and describes different concepts that are related to VR applications. This includes HCI and available platforms for developing VR applications.

B.1 Overview

VR is a digital illusion of the real world or a fantasy world. Hardware like the CAVE or a HMD is only a few technical solutions of many to make the illusion of a world, that may or may not exist. The VR we know today is primarily built on visual illusion, but the effect can be optimized by sound and/or force feedback. Force feedback is a technique to make the user believe he can feel what he is interacting with, in other words a haptic[1] response. This is simulated by a electronic powered physical resistance.

B.2 Human-Computer Interaction

HCI is the concept of users interaction with the system, so by making this interaction more intuitive, the use will be more intuitive as a whole. The most intuitive interface for modeling is likely to be data gloves, because they are supposed to act like hands. A big difference between using a data glove and actually using your hands on a physical object, is the fact that the data glove does not give any response (eg. force feedback) if touching an object. In other words you can not "feel" the objects you are touching. Another

[1]Based on the sense of touch. Usually achieved by force-feedback.

possibility is to use a wand for interaction, which is a handle with buttons. Such a wand can be controlled in 3D, but using gloves will be more intuitive. An example of a data glove is the 5DT 5+2 sensors data glove which is a lycra glove, that measures the flex of each finger, where 0 is a straight finger and 4095 is the maximum flex. Furthermore the gloves have a sensor which measures the pitch and roll of the glove (See Figure B.1).

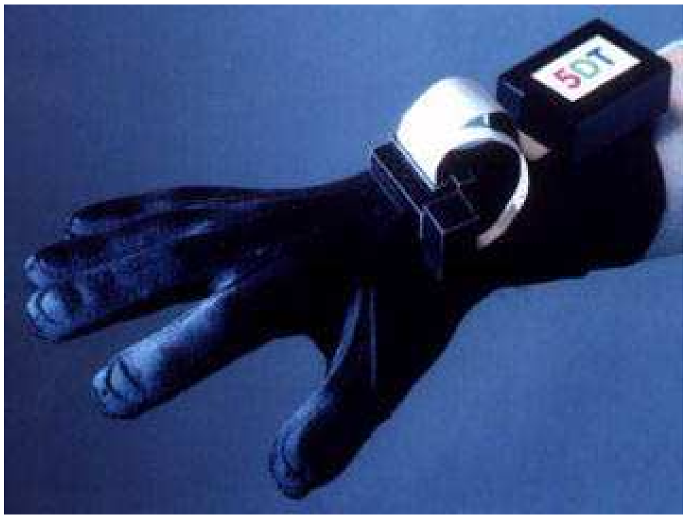


Figure B.1: A 5DT data glove

Another HCI consideration that is important is the visualization of the 3D world. This can be achieved by using the CAVE, a HMD or by a simpler solution as shutter glasses combined with a monitor. Shutter glasses is not real 3D, but a way to trick the eyes to see a screen as 3D. It blocks the view for one eye by turns at the frequency the screen flicker between an image for the right eye and one for the left eye. The HMD is showing the 3D world to the user by two small monitors placed close to the eyes, with the picture moving depending on the way the user turns the head (See Figure B.2).

Finally there is the CAVE, which is a closed cube the user is placed inside. Typically the user wears shutter glasses inside the CAVE to get the illusion of 3D. Figure B.3 shows a situation from inside the CAVE, where the illusion of the engine hanging in thin air is a result of the projections through all 6 walls in the CAVE combined, with the shutter glasses the users is wearing. The situation on the figure is put up for the camera, since the users will not get the same illusion. This is because the illusion is synchronized for the position of a tracked pair of shutter glasses.

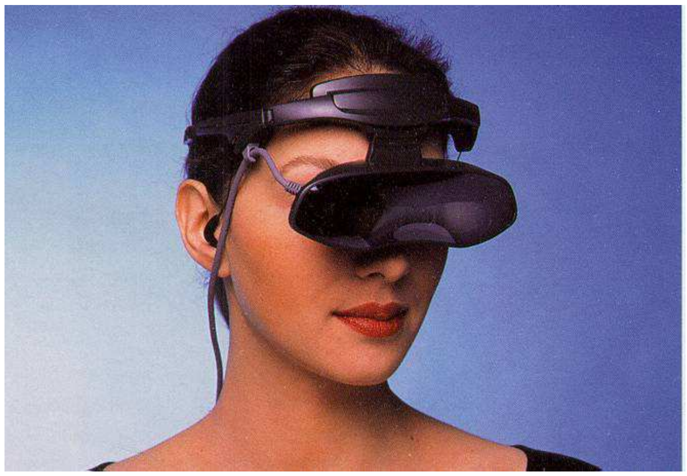


Figure B.2: An example of a Head Mounted Display

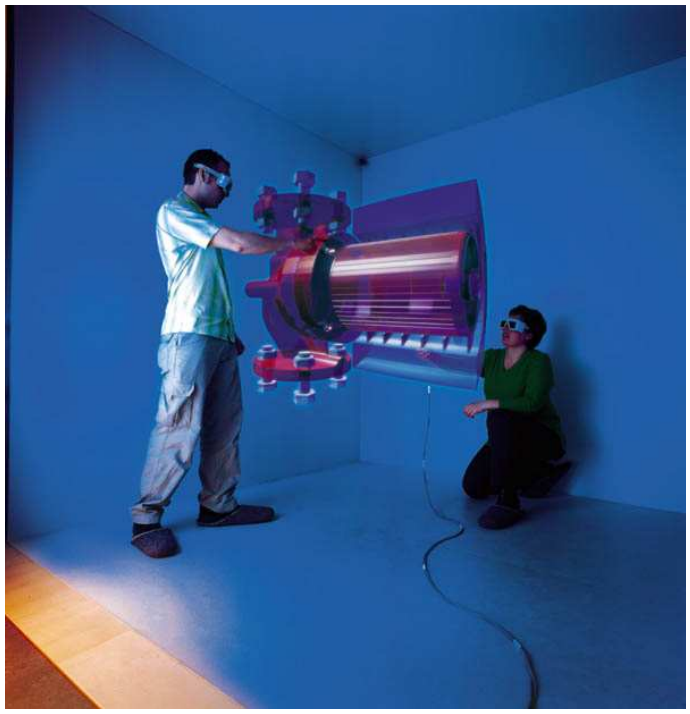


Figure B.3: The inside of a CAVE with an interacting user

A difference between a HMD and the CAVE, is how the user's hands appear. In the CAVE the hands will always be in front of the object, because the object is projected at the wall. With the HMD solution, without see through screens, the hands are represented by a digital image and can therefore be placed both in front and behind the object, provided the hand is being tracked (See Figure B.4). See through screens enable the user to see the real world behind the images that are shown on the HMD.

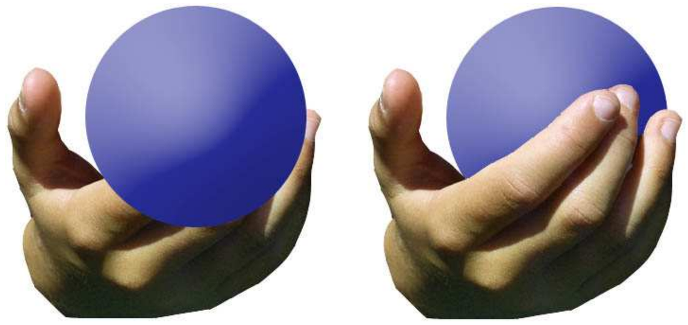


Figure B.4: In the CAVE, the users hand will always be in front of the object, because the objects is projected on the wall. This scenario will look like the picture to the right. The HMD, without see through screen, has the possibility of showing the hand behind the object, since the hand can be an object itself, provided the hand is being tracked.