Conceptual Free-Form Styling in Virtual Environments

Dissertation zur Erlangung des Grades des
Doktors der Ingenieurwissenschaften (Dr.-Ing.)
der Naturwissenschaftlich-Technischen Fakultät I
der Universität des Saarlandes

eingereicht im März 2004

von

Gerold Wesche

Dekan:	Prof. Dr. Jörg Eschmeier
Vorsitzender des Prüfungsausschusses:	Prof. Dr. Joachim Weickert
Akademischer Mitarbeiter:	Dr. Marcus Magnor
1. Gutachter:	Prof. Dr. Hans-Peter Seidel
2. Gutachter:	Prof. Dr. Heinrich Müller
Tag des Kolloquiums:	20. August 2004

Kurzfassung

Diese Dissertation beschreibt Werkzeuge zum Entwurf kompletter virtueller Modelle von Grund auf. Dies geschieht direkt in einer tischartigen, virtuellen Arbeitsumgebung mit Hilfe von Tracking der Hände und der Kopfposition. Die Modelle sind aus Freiformflächen aufgebaut und werden als Netz von Kurven mit Hilfe eines getrackten, stiftartigen Eingabegerätes direkt im Raum gezeichnet. Es werden interaktive Deformationswerkzeuge für Kurven und Flächen vorgestellt, die auf Methoden des Variational Modeling basieren. Durch das Ausrichten des Modells mit der linken Hand wird das Editieren mit der rechten Hand erleichtert. Dies entspricht einer natürlichen Aufteilung von Aufgaben auf beide Hände. Zusätzlich stellt diese Arbeit neue Techniken für die 3D-Interaktion in virtuellen Umgebungen, insbesondere im Bereich Anwendungskontrolle, vor, die die Aufgabe der Werkzeugauswahl in den Arbeitsablauf der Formgestaltung integrieren.

Das Ziel dieser Arbeit ist es, besser geeignete Schnittstellen für den computerunterstützten, konzeptionellen Formentwurf zur Verfügung zu stellen; ein Gebiet, für das Standard-Desktop-Systeme wenig geeignete Unterstützung bieten.

Abstract

This dissertation introduces the tools for designing complete models from scratch directly in a head-tracked, table-like virtual work environment. The models consist of free-form surfaces, and are constructed by drawing a network of curves directly in space. This is accomplished by using a tracked pen-like input device. Interactive deformation tools for curves and surfaces are proposed and are based on variational methods. By aligning the model with the left hand, editing is made possible with the right hand, corresponding to a natural distribution of tasks using both hands. Furthermore, in the emerging field of 3D interaction in virtual environments, particularly with regard to system control, this work uses novel methods to integrate system control tasks, such as selecting tools, and workflow of shape design.

The aim of this work is to propose more suitable user interfaces to computer-supported conceptual shape design applications. This would be beneficial since it is a field that lacks adequate support from standard desktop systems.

Zusammenfassung

Diese Dissertation stellt neue dreidimensionale Interaktionstechniken in virtuellen Umgebungen zur Unterstützung der konzeptionellen Phase des Formentwurfs vor. Komplette virtuelle Modelle aus Freiformflächen können direkt in einer virtuellen Umgebung von Grund auf erstellt und verformt werden. Hierbei werden beide Hände benutzt.

Die traditionelle Sichtweise virtueller Umgebungen hebt in erster Linie ihre Fähigkeit zur Simulation eines intensiven Präsenzeindrucks in einer virtuellen Szene hervor und das Gefühl, in ihr eingebettet zu sein. Benutzerinteraktion wird oft lediglich als Navigation durch eine virtuelle Welt verstanden. In letzter Zeit werden virtuelle Umgebungen zunehmend als Benutzerschnittstelle zu Computeranwendungen wie dreidimensionale Simulation und Visualisierung eingesetzt. Hierbei stehen jedoch nur eingeschränkte Möglichkeiten zur Interaktion mit den Daten zur Verfügung.

In dieser Arbeit wird herausgestellt, dass virtuelle Umgebungen ein großes Nutzungspotential auch für kreative Anwendungen haben, da sie die Fähigkeit des Menschen zur Benutzung der Hände unterstützen können. Dies ist nicht auf die Nachahmung manueller Abläufe beschränkt. Abgesehen davon, dass die Übertragung von Prozessen aus der realen Welt in eine virtuelle Umgebung nicht immer möglich ist, ist sie oftmals auch nicht angebracht. Demgegenüber können künstliche Konzepte für manuelle Interaktion sich die Möglichkeit zur freien Bewegung der Hände in virtuellen Umgebungen zu Nutze machen.

Ein Anwendungsfeld, das besonders auf manuellen Fertigkeiten beruht, ist die Formgestaltung von Produkten. Die Anfangsphasen im Design sind besonders kreative Prozesse, in denen eine Form in erster Linie durch eine gekonnte Benutzung der Hände bestimmt wird. Der konzeptionelle Designprozess wird auch heute noch von traditionellen Arbeitsmethoden dominiert, wie dem Skizzieren auf Papier und dem Modellbau.

Ab einer gewissen Phase des Entwurfsprozesses wird jedoch ein digitales Modell benötigt. Die Übertragung eines realen Modells oder einer Zeichnung in ein digitales Modell ist ein sehr aufwendiger und zudem fehleranfälliger Vorgang. Daher gibt es einen Bedarf für Computerunterstützung auch in den frühen Entwurfsphasen; jedoch werden traditionelle Methoden bevorzugt, trotz des Nutzens digitaler Modelle. Der Grund hierfür liegt in der unzureichenden Unterstützung kreativer Prozesse durch derzeitige Computer-Aided-Design-Systeme (CAD-Systeme).

In dieser Arbeit wird die Nutzbarmachung von CAD-Funktionalität durch virtuelle Umgebungen als Schnittstellen vorgeschlagen, um diese Einschränkungen zu überwinden. Darüber hinaus sind neue Designprozesse denkbar, die auf dem Entwurf virtueller Modelle basieren, aber die Notwendigkeit realer Modelle durch Nutzung neuartiger dreidimensionaler Printing-Technologien einbeziehen.

Das Design der Oberfläche vieler Industrieprodukte basiert auf der Verwendung glatter Freiformflächen. Der Entwurf einer Form mit Freiformflächen ist eine besonders herausfordernde Aufgabe, da eine sehr hohe Anzahl an freien Form gebenden Parametern existiert, die von Hand oder mit Hilfe von Algorithmen vom Designer festgelegt werden müssen. Für eine effektive Unterstützung des Entwurfs von Freiform-Modellen sollte ein Flächenansatz gewählt werden, der sowohl den technischen als auch den methodischen Anforderungen des Entwerfens in virtuellen Umgebungen Genüge tut, und für den Entwurfsmethoden zur Verfügung stehen, die die Konstruktion kompletter virtueller Modelle erlauben.

Der Modellierer, der im Zusammenhang mit dieser Arbeit implementiert wurde, wurde unter Berücksichtigung dieser Aspekte entwickelt. Er basiert auf dem Zeichnen der Konturen eines Modells als Netzwerk von Kurven (kubische B-Spline-Kurven), das die Form des gesamten Modells darstellt. Diese Kurven werden direkt

in einer tischartigen virtuellen Umgebung gezeichnet, genannt Responsive Workbench. Die Responsive Workbench ist ein projektionsbasiertes Displaysystem mit Tracking des Kopfes und der Hände des Benutzers, das Objekte so dargestellt, dass sie über dem Tisch erscheinen.

Das Kurvennetz wird mit Flächen ausgefüllt, die den vom Netzwerk angedeuteten Formverlauf widerspiegeln. Zwei Flächenansätze werden unterstützt, die Flächen von Kuriyama und Catmull-Clark-Flächen, welche direkt deformierbar sind.

Für Kurven stellt diese Dissertation interaktive Deformationswerkzeuge vor. Beispielsweise wird eine Kurve durch das Entlangführen eines virtuellen Cursors an der Cursorposition allmählich geglättet oder geschärft. Die Genauigkeit dieses Vorgangs kann durch das indirekte Einwirkenlassen der Deformation wirksam kontrolliert werden. Hierfür werden Methoden des Variational Modeling verwendet. Durch die Kombination von Energiefunktionalen mit Funktionen zur Gewichtsverteilung wird ein dynamisch positionierbarer Maximaleffekt der Glättung bzw. Schärfung erzielt. Die Deformationswerkzeuge für Flächen benutzen ebenfalls Methoden des Variational Modeling und erlauben interaktives Ausformen.

Ein zweihändiges Interaktionsschema, das auf Arbeitsteilung für beide Hände beruht, unterstützt den Modellierungsprozess. Die linke Hand richtet das Modell so aus, dass die rechte Hand Teile des Modells zum Editieren leicht erreichen kann.

Im Gegensatz zu Desktop-Systemen, bei denen sich Benutzerschnittstellen weitgehend ähneln, wird allein aus der Funktionalität des Modellierers noch nicht seine eigentliche Arbeitsweise klar. Für virtuelle Umgebungen sind effektive Interaktionstechniken zum Zugriff auf die zur Verfügung stehenden Funktionen keineswegs offensichtlich. Das einfache Übertragen von bekannten Techniken der Desktop-Interaktion führt in den meisten Fällen zu Schnittstellen, die hinderlich statt förderlich sind. Dies gilt besonders für die Komponenten zur System-/Anwendungssteuerung. Leider unterschätzen viele Entwickler die Bedeutung, die die Benutzbarkeit dieser Komponenten für die Effizienz des Gesamtsystems hat.

Interaktion im Bereich Systemsteuerung in virtuellen Umgebungen beinhaltet den Zugriff des Benutzers auf ein Werkzeug oder eine Funktion. Als Beispiel hierfür kann eine komplexe Modellierungsaufgabe dienen, die es erfordert, wiederholt das Werkzeug zu wechseln. Hierfür sind Methoden zu finden, die die notwendigen Aktionen des Benutzers auf elegante Art in den eigentlichen Arbeitsablauf der Formgebung integrieren. Daher werden in dieser Dissertation zwei Ansätze zur Unterstützung der Funktionsauswahl vorgeschlagen. Das Hand-Menü folgt der Hand, erlaubt dabei jedoch die Auswahl einer Funktion mit derselben Hand. Das „Tool-Finger“-Widget stellt mehrere Funktionen gleichzeitig an der Hand zur Verfügung. Eine bestimmte Funktion kann durch Positionieren des zugeordneten Bereiches auf dem Widget mit dem zu editierenden Objekt zügig ausgewählt und unmittelbar angewendet werden.

Diese Dissertation trägt durch die Entwicklung neuer Methoden in den Bereichen Design von Freiformflächen und Systemsteuerung in virtuellen Umgebungen zur Weiterentwicklung von Designprozessen bei.

Summary

This dissertation proposes novel spatial interaction techniques in virtual environments for the support of the conceptual stages of shape design. Complete virtual models consisting of free-form surfaces can be created from scratch and deformed directly within a virtual environment, using both hands.

The traditional view on virtual environments primarily emphasizes their ability to simulate a high sense of presence, and the impression of being immersed within a virtual scene. User interaction is often understood merely as navigation through a virtual world. In recent years, there has been an increase in the use of virtual environments as interfaces to computers. Typical applications include three-dimensional simulation and visualization; however they allow only a restricted interaction with the data.

In this work it is argued that virtual environments have great potential to benefit creative applications. Furthermore, this work provides novel interfaces that rely on the human's ability to use the hands. This is not restricted to imitating manual work procedures within the physical world. Despite the fact that practices within the natural world are not always applicable in virtual environments, they are often not appropriate as well. In contrast to this, artificial concepts for manual interaction could make use of the possibility of free hand movement in a virtual environment.

An application field that strongly relies on manual skills is the field of industrial shape design. The initial stages of shape design involve particularly creative processes such as the skillful use of the hands. This is the most important factor that contributes to the resulting shape. The conceptual phase of design is still dominated by traditional activities, such as sketching on paper and building physical models.

However, at some stage in the development process a digital model will be required. The conversion of a physical model or a drawing into a digital model is a very costly process, which is prone to error. Therefore, there is a need for computer support at the early stages of design, but in spite of the benefits of digital models, traditional methods are preferred over the use of computers. The reason is that the current interfaces to computer-aided design (CAD) systems lack support for creative activities.

In this work, an innovative use of CAD functionality through virtual environment based interfaces is proposed, in order to overcome these drawbacks. Furthermore, new design processes are conceivable if based on the sketching of virtual models, as well as take account for the need of physical models by using the emerging technology of three-dimensional printing.

The design of the surface of many industrial products makes use of smooth, free-form surfaces. The shape conceptualization with free-form surfaces is a particularly challenging task, since there are an overwhelming number of free shape parameters that have to be specified by the designer either manually or with the aid of algorithms. In order to have effective support of conceptualizing free-form models, a surface approach should be selected that meets the technical as well as the methodical requirements for sketching in virtual environments. These requirements include the availability of sketching methods that allow the creation of complete virtual models.

The modeler that has been implemented in the context of this work has been developed with regard to these issues. It is based on drawing the contour curves of a model and forming a network of curves, using cubic B-spline curves, which outlines the shape of the complete model. The curves are drawn directly within a virtual environment system, called Responsive Workbench, using tracked input devices. The Responsive Workbench is a head-tracked table-like projection-based environment that displays virtual models in such a way that they appear above the table.

The curve network is filled with surfaces that reflect the shape outlined by the contour curves. Two kinds of surfaces are supported, namely the surfaces of Kuriyama, and Catmull-Clark surfaces, which have the advantage of being directly deformable.

For curves, this dissertation introduces interactive deformation tools. For example, moving along a virtual pointer on a curve gradually smoothes or sharpens it at the location of the cursor. Such an indirect deformation metaphor enables to effectively control the precision of this process, using variational methods. Energy functionals are combined with a weight function, which yield a local effect of the energy minimization that is dynamically changing. The deformation tools for the surfaces also use variational methods and enable interactive surface sculpting.

A two-handed interaction scheme, based on observations of how tasks are distributed onto the hands in the real world, supports the modeling process. The left hand aligns the virtual model appropriately, so that the right hand can easily reach parts of the model for editing purposes.

Apart from desktop systems, which have very similar user interfaces, a specification of the functionality of the modeler alone does not imply how it actually works. For virtual environments, effective interaction techniques to access the provided functionality are far from being obvious. Simply transferring methods known from desktop interaction in most cases leads to obstructive rather than supportive, user interfaces. This is especially true regarding the components for system control. Unfortunately, many developers underestimate the importance of usability issues in this field for the efficiency of the whole system.

System control interaction tasks include the user accessing a tool, or a function. Consider a complex modeling task that requires changing the current tool repeatedly. Methods have to be found that fluently integrate the actions needed to control tool selection into the main task, which is modeling a shape. Therefore, two approaches for the support of tool selection are proposed in this dissertation. The hand menu is following the hand, but still allows for the selection of a tool with the same hand. The "Tool Finger" widget attaches several tools to the hand. A particular tool can be quickly selected by overlapping the associated region of the widget with the object that is supposed to be edited.

This dissertation contributes to the further development of design processes by developing novel methods in areas concerning free-form shape design and system control in virtual environments.

Contents

List of Figures

Chapter 1

Virtual Environments

1.1 Introduction

The initial stages of shape design would greatly benefit from adequate computer support. At the beginning of these processes, designers create ideas of a shape from scratch. The skilful use of the hands and the quick creation of multiple variants of a shape are characteristic for this design phase. Applying these work methods using current desktop based interfaces is impracticable, due to their unsuitable input and output components, and inadequate interaction styles.

With the advent of virtual environment technology, novel user interfaces for computer aided three-dimensional geometric modeling are conceivable: they hide the computer in the background, and bring the designer's most important tools, his hands, close to the virtual work piece. Modeling tools interpret the movements of the tracked hands and determine how to deform an object or generate drawing strokes, making it possible to create a design from scratch. Both hands are in use at the same time; with one hand to position and orient the model and the other hand to manipulate it. Naturally, the virtual model is represented on a scale that corresponds to the working area of the hands. A perspectively correct stereoscopic projection of the model ensures that the user can use his hands directly. In short, the designer is modeling within a *virtual environment*, that replaces the keyboard and the computer screen.

This particular approach is discussed in further detail in this work. It describes a computer aided conceptual styling system that has been developed and implemented for direct use in a virtual environment. The modeler relies on spline-based free-form surfaces that are designed from scratch; using both hands. The hands form complete virtual models that can be further elaborated using deformation tools provided with the modeler.

1.1.1 Main contributions

In the field of geometric modeling, this dissertation contributes extensions to variational modeling methods. They allow the application of variational modeling to deform curves and surfaces in a much more flexible way than before. Furthermore, novel 3D user interface concepts that make use of spatial relationships between virtual objects and the hands of the user are presented. These techniques integrate system control tasks into the main workflow, and allow a more fluent control of applications.

The modeler that is formed by these components is based on curve networks that are drawn in space. The purpose of the curve network is to outline the shape of the model, so that its surface need not be directly specified. The idea to use

curve networks for surface sketching has already been proposed; see section 2.5. In addition, this work presents how a projection-based virtual environment can support the modeling and editing of curve networks. Furthermore, it demonstrates how the virtual environment supports the definition of the topology necessary to create the surface.

The deformation algorithms proposed for curves and surfaces are also applicable to other modelers that are based on direct manual interaction with curves and surfaces. Similarly, the spatial interaction techniques for system control introduced in this work are generally applicable to all virtual environments that provide a large set of functions for object manipulation.

1.1.2 Purpose

Virtual reality has passed the phase where the users just had the role of passive spectators and bystanders. These days, the user can control the application directly in the virtual environment by specially developed input devices and spatial interaction techniques. However, user interaction with the model in most cases is restricted to transformations and selections between different representations.

Taking interaction in virtual environments one step further leads to *creation* of geometry, including computer-aided design. The beginning of this development is marked by systems that allow an immediate input of shape. Conceptual sketching is particularly supported; whereas further elaboration and editing of the model is rarely considered.

The purpose of this work is to foster the use of virtual environments for creative tasks such as shape design. This can be achieved with modeling tools for curves and surfaces developed specifically for use in a virtual environment, and interaction techniques adapted to this work method. The goal is to find ideas for user interfaces to computer-aided styling tools that enable creative work. As an extension to previous contributions in this field, the modeling functionality presented in this dissertation goes beyond just spontaneous input of initial sketches.

Since geometric modeling in virtual environments is an emerging field that is in the beginning stage of development, it is too early to conclude to what degree novel methods will be accepted in the end by designers. Moreover, it is difficult to predict at which stage virtual environments will be included into design processes of the future. In any case, newly developed interaction and deformation metaphors will be of great value for future human-computer interaction in the field of design.

Unfortunately, software-based approaches and solutions as presented in this work, are not enough. Various technical shortcomings in areas such as tracking, user comfort and stereo viewing have to be overcome in order to achieve user acceptance.

1.1.3 Overview

In summary, this work introduces interaction techniques for geometric modeling in virtual environments, particularly with regard to conceptual free-form styling. For this purpose, a modeler has been developed and implemented.

The first three chapters provide introductory material. In chapter 1, the concept of virtual environments is presented. Many definitions of virtual environments found in publicly available literature emphasize the aspects immersion and presence. To the contrary, virtual environments are discussed here with regard to user interaction. Additionally, the most important hardware components and software components of virtual environment systems are presented.

In chapter 2, the conceptual phase of the shape design process is examined. Traditional as well as computer-based methods are presented. It is argued that var-

ious shortcomings of current desktop computer-aided design (CAD) systems prevent their use in the conceptual phase. The potentials as well as the restrictions of virtual environments for the support of conceptual design are discussed. Furthermore, recent work in this field is summarized as well as discussing the various sketching and modeling approaches; including surface and volume oriented solutions.

Technical as well as methodical requirements for sketching in virtual environments are presented together with important surface approaches in chapter 3. This includes B-spline curves, tensor-product B-spline surfaces and Catmull-Clark surfaces.

The next three chapters describe creation and manipulation tools for modeling free-form objects in a virtual environment. Chapter 4 describes a modeler that allows a designer to form the surface of an object using curve networks. Methods that demonstrate how to draw, connect, and edit connected curves are presented. Moreover, it is shown how the topology of a curve network can be determined using interactive tools in a virtual environment.

Chapter 5 introduces deformation tools on spline-based curves and surfaces for use in a virtual environment. They rely on variational methods. For spline curves, these methods have been further developed into interactive shaping tools, which is presented in detail. The tools are designed for direct manual interaction with virtual curves and surfaces, and work reasonably well even without the use of force feedback. Moreover, they are also applicable in other spline-based modeling systems.

In chapter 6, methods are presented on how to access functionality, and how to integrate the task of selecting a tool into the workflow of shape design in a more fluent manner. This is achieved with 3D interaction widgets specifically designed for use in virtual environments.

Several examples of models that have been sketched with the presented tools are shown in chapter 7.

A conclusion about conceptual free-form styling in virtual environments is drawn in chapter 8.

1.1.4 Publications

Parts of this dissertation are based on three publications by the author at scientific conferences. In [WD00], Wesche and Droske describe in detail the variational curve and surface deformation tools that are the topic of chapter 5. Wesche and Seidel give an overview about the capabilities of the modeling system implemented for the purposes of this dissertation in [WS01]. The components of the user interface that control how to access the functionality of the modeler deserve special attention. For that purpose, a novel interaction widget has been developed and presented by Wesche in [Wesc03]. This contribution is included in chapter 6.

1.1.5 Notation

In the mathematical parts of this work, the following notation is used. Points and vectors are assumed to be columns and are denoted as boldface lower case characters, like $\mathbf{v}$, whereas matrices are denoted as uppercase boldface characters, e.g., $\mathbf{M}$, or as (m_{ij}), where m_{ij} is the (scalar) element at row i and column j. Scalars are denoted as italicized characters, e.g., w or E. Scalars that are coordinates of a point or of a vector $\mathbf{v}$ are denoted e.g. as v_i. The dot product between two vectors $\mathbf{x}$ and $\mathbf{y}$ is written $\mathbf{x}^T\mathbf{y}$. Derivatives are denoted by points, by primes, or by subscripts, e.g., $\dot{\mathbf{x}}$, $\mathbf{d}'$, or $\mathbf{s}_u$.

1.2 Virtual environment systems

These days industrial design processes rely on efficient computer-supported methods to meet increasing requirements concerning development time, flexibility, economy, and product quality. Besides the emerging performance of numerical power and high-end computer graphics, the usefulness of computer-supported methods highly depends both on the input and output components of the human-computer interface. The development of *virtual environment systems* still is focussed on output components. Projection technology, stereoscopic viewing, graphics hardware and rendering algorithms is used to increase the visual quality of artificially generated environments. Compared to the efforts made in this field, the development of the input components of 3D environments still lags far behind. This is also noticeable if you study the definitions of virtual environments.

Ellis [Elli95] defines a *virtual environment (VE)* as a synthetic, interactive, illusory environment, which is perceived this way when a user wears or inhabits the appropriate apparatus, providing a coordinated presentation of sensory information and imitating a physical environment.

The hardware and software components, which generate a virtual environment, form a *virtual environment system (VE system)*. More precisely, VE systems can be defined as computer-based information technologies for interactive, real-time oriented simulation, and multi sensory representation of objects, processes, and their results [BBB97].

Making usable VE systems as *tools* to support various industrial, medical and other processes has been and still is an emerging field of research. However, most definitions of VEs and VE systems are centered on mediating artificial human sensations. The active role of the user is rarely emphasized explicitly.

In contrast to that, interpreting a VE system as a human-computer interface means that criteria such as a high degree of presence of being immersed in a 3D scene become less important. M. Krueger [Krue91] initiated these concepts with his work on non-immersive *responsive environments*, in which the computer acts as a server in the background; reacting to the user's input across multi sensory interaction channels.

This work is largely based upon such virtual *work* environments. An application for conceptual free-form styling will be introduced, running on a Responsive Workbench (1.2.2), in which users generate surfaces from scratch and deform them directly in space using the hands. Research within this field, referred to as *immersive modeling*[1] or VE-based modeling (see 2.3), concentrates on providing a restricted class of users, namely designers or artists having corresponding skills, with an adequate user interface. The goal therefore has migrated from imitating physical environments to supporting users, or certain classes of users, with interfaces adapted to their skills and to their needs. This includes, but is not restricted to, developing input devices and interaction techniques that exhibit a *natural* behavior.

However, this not only can turn out to be very difficult, but also is not always appropriate. Therefore, the scope of research should include *artificial* concepts for spatial interaction. It is important to note that interaction methods which do not imitate natural work methods may require from the user a higher effort to adapt to the conditions imposed by the VE, and to the offered user interface. This seems to be acceptable since the proper use of physical tools needs learning as well.

Not all existing VE systems are suitable for a certain application field. The available setups have certain characteristics that have direct consequences for the application fields they can be used for. In the next paragraphs, the hardware and software components of VE systems, and the related applications, will be presented.

[1] The term *immersive* is not always appropriate, since non-immersive VEs are also used.

1.2.1 Head-mounted displays

The first VE systems have been built with head-mounted displays (HMDs), which provide each the left and the right eye with a separate image for stereo viewing. Sensors recognizing the head movement provide the necessary information for updating the view definition. Worn as helmets, HMDs isolate the user from the physical environment, so that the user looses all reference points. This in combination with system lag can cause motion sickness. Typical technical problems that HMD-based VEs suffer from are poor display resolution, and restricted field of view. The full immersion of the user into the virtual scene predestinates HMDs for "walkthrough" or "flythrough" applications. Being equipped with a joystick device, the user can navigate through virtual scenes.

1.2.2 Projection-based virtual environments

Projection-based virtual environments (PBVEs) consist of an arrangement of projectors connected to the output of graphics computers, mirrors, and screens. In the case that the projectors are located behind the screens relative to the viewer, the setup is referred to as *back-projected*, otherwise as *front-projected*. With projection technology, a stereo image can be generated using two different methods. The first method is alternately projecting a separate image for the left and right eye and synchronizing stereo shutter glasses with the corresponding frequency. The second method uses separate projectors for the view of the left and the right eye, combined with polarization filters mounted to the projector's lenses, as well as polarizing stereo glasses.

Whereas a HMD, as the name indicates, is mounted to the user's head, the screens of a PBVE are stationary, so that the user can move relatively to the setup. PBVEs do not completely isolate the viewer from the outer world, since it is always possible for the viewer to recognize objects from the physical environment, such as other participants even when looking through stereo glasses. Moreover, it is much easier for a person, in case of discomfort or for other reasons, to get rid of the stereo glasses than to get rid of a HMD, which can require assistance from other persons.

A characteristic common to all PBVEs is the large screen size, which typically reaches from about one up to a few meters. This results in a large field of view, and it allows for the representation of virtual objects on a scale suitable for direct interaction with the hands. The larger field of view, combined with a high-resolution graphics output, is one reason why PBVEs are currently superior to HMDs for various types of applications.

The CAVE

The surround-screen projection-based CAVE (Cave Automatic Virtual Environment) [FSCN93] completely immerses users in a virtual scene. The projection screens have an extent of about 3 by 3 meters and form a cube, see Figure 1.1. Up to 6 screens are installed, however 5- oder 4-sided CAVEs are more commonly used. In this case, the top screen and the back screen are often omitted. Together with the eye points of a head-tracked user, each screen forms a separate pair of view definitions. Each view plane onto which the 3D scene is rendered, corresponds to a projection screen. Equipped with stereo glasses, the reference viewer is able to perceive a stereo image of the scene that has the correct perspective. This is possible regardless of the direction the viewer is looking into, even if projections of objects partially overlap several screens. In order to minimize the visibility of the edges and the corners at adjacent screens, the projectors must be calibrated properly.

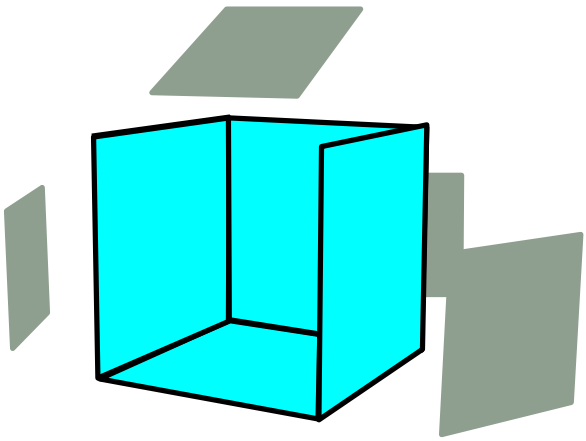

Figure 1.1: A four-sided CAVE setup. The blue areas are the projection screens, and the grey areas denote the location of the mirrors. The four projectors are placed accordingly.

However, several people typically participate in a CAVE session, but only the tracked viewer's perspective is rendered. All other users observe perspective distortions that increase when they move away from the tracked user. This problem is common to all PBVEs, in which head tracking is used.

Due to its construction, a CAVE setup primarily aims at generating a convincing illusion of presence, giving users the feeling of being completely immersed in a virtual scene. Application fields that benefit from immersion include architecture, such as designing and furnishing the interior of buildings, interior car design, walking and flying through artificial worlds, simulating processes in factory halls, or presenting complex medical processes inside the human body.

The Responsive Workbench

The immersion of the user into a virtual scene is not always necessary and is not desirable for many applications. W. Krüger et al. [KBF+95] concentrated on a restricted class of users, such as physicians, engineers, architects, and designers. The problems these users encounter typically require desks or tables. This is based on a user task analysis, resulting in the Responsive Workbench virtual work environment.

The original Responsive Workbench has one 1.8 m by 1 m sized projection screen that is horizontally arranged, similar to a working desk. An inclined mirror that is mounted under the transparent tabletop reflects the light, coming from a single beamer, see Figure 1.2. The parameters of the view definition result in an off-axis stereoscopic projection, with a view direction always pointing vertically downwards. As in other PBVEs, head tracking is used at the Responsive Workbench. The system is tuned such that the virtual objects appear to be above the table. The maximal height of virtual objects is limited to about 40 cm according to the size of the desk. Objects having a higher extent are cut by the upper plane of the viewing frustrum, reaching from the user's head to the distant edge of the display. This restriction led to the idea of seamlessly adding a second vertical projection screen to the edge of the display, thereby adding a separate viewing frustrum. The result is the two-sided Responsive Workbench, shown in Figure 1.3.

Applications running on the Responsive Workbench benefit from the coincidence of the physical projection plane and the view plane. This is used as a virtual table where the objects of the scene are placed similar to using a real table.

The Responsive Workbench has been introduced as a non-immersive VE system.

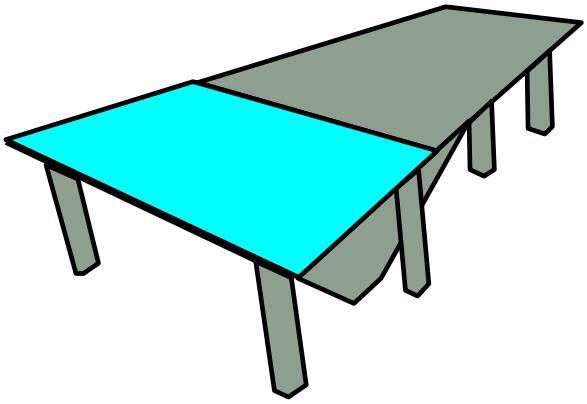

Figure 1.2: The Responsive Workbench setup, similar to a picture in [KBF+95]. The blue area denotes the projection screen. The projector is located under the back part.

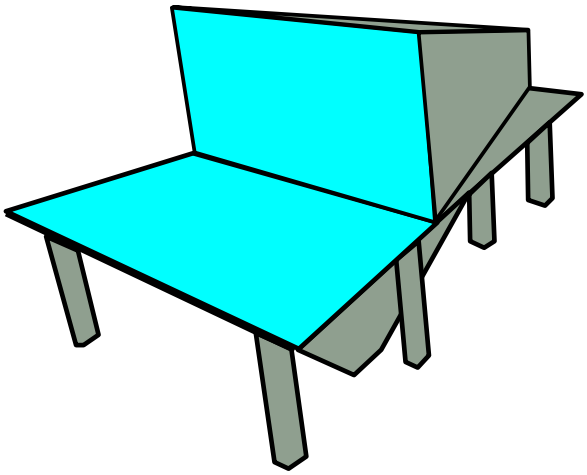

Figure 1.3: Setup of the two-sided Responsive Workbench. Two projectors are integrated in the housing.

The size of the projected objects, and the relative position of the user in front of the table, make it well suited for applications that require interactions performed directly with the hands.

The workbench can be used as a resting position for both hands or for one hand if the other hand is in use. It can also serve as a place to lay aside input devices. This can be very helpful when several such devices are needed.

These characteristics can have a positive influence on the comfort of the user. For example, it has been reported [DBW+00], that designers miss resting positions when interacting with immersive modeling applications in the CAVE.

Applications for the Responsive Workbench include interactive 3D exploration of data, such as results from crash simulations or computational fluid dynamics. Furthermore, these are represented on a scale corresponding to the table size, which do not require spatial navigation through the data. Since recently, the Responsive Workbench is increasingly being used for more complex applications such as physical based assembly simulation, or geometric modeling.

Other PBVEs

Beamers and projection screens can be arranged in various ways, forming different PBVEs. Two examples have been presented, the CAVE and the Responsive Workbench. Another commonly used setup consists simply of one vertical projection screen and one or two beamers, depending on the method of stereo viewing. In the case the applications do not require that users are located *inside* a model or navigate *through* a scene, one screen is fully sufficient for the purposes of visualization or presentation.

Considering interactive applications, a Responsive Workbench seems better suited. Contrary to the Responsive Workbench, a projection wall, also referred to as *powerwall* is usually not as tightly coupled to the user. Since the working space of the hands at the Responsive Workbench coincides with the region where the virtual objects are located, there seems to be a tighter spatial relationship between the user and the objects.

By combining several projectors with a large projection screen that is curved for achieving a higher degree of immersion, a VE system for larger groups is formed. In such environments head tracking is normally disabled. Instead, the view definition uses a centered viewpoint, so that all participants observe from the approximately correct viewpoint.

1.2.3 Hardware components

Where appropriate, the other basic hardware components common to most VE systems include stereo glasses, sensors to track the motion of the head, or the hands, input devices such as pen-like devices, joysticks, or special-purpose devices.

Stereo glasses

The two most common technologies for stereo glasses are LCD-based shutters and polarization filters. In interactive environments, like the Responsive Workbench, where the user often changes his position and turns his head, shutter glasses are predominantly used. Contrary to some polarization-based methods, the stereo effect is active independent of head movement.

Emitters, whose infrared light signal is received by a sensor on the glasses, synchronize the shutter glasses. It should be ensured that the emitters are appropriately placed so that the signal always reaches the glasses when turning the head.

Tracking

Tracking means continuously measuring the position and orientation of relevant parts of the user's body and of input devices. Since tracking is the basis for all VE-based user interaction, it is a crucial component of the whole system.

In an interactive VE, tracking all 6 degrees of freedom (6-DOF tracking) of the user's head and of the hands is essential to make full use of spatial movement. 6-DOF tracking effectively means that a transformation matrix between the local coordinate system of a *sensor* or *receiver*, mounted to the user's head or to the hands, is determined with respect to a fix reference coordinate system.

The tracking technology has to provide this data reliably, with sufficient accuracy, and without any significant delay. Currently, electromagnetic tracking is most commonly used in VE systems, but camera-based methods are gaining importance, since they offer cable less tracking. In case of electromagnetic tracking, the reference coordinate system is determined by the location of the *transmitter*, which emits an alternating electromagnetic field. Consequently, this technology can seriously be affected by metal. If the VE system setup is designed carefully (i.e. no

metal around), and the transmitter is sufficiently close to the working area of both the hands and the position of the head, the accuracy and the stability of the data can be surprisingly high. Therefore, it may be used for interactive tasks that require fine motor skills, such as geometric modeling in a VE. This can be demonstrated with the modeler that has been developed in the context of this work. However, away from the main working area, the user has to be aware that deviations and jittering effects increase.

Head tracking is used to determine the position of the eyes with respect to a world coordinate system in which the virtual scene is given. The head sensor is usually mounted to the stereo glasses. The world system has a fixed transformation matrix with respect to the transmitter coordinate system. This transformation has to be determined only once in a calibration step and remains constant as long as the setup is not changed. If the position of the user's eyes are known in the coordinate system of the head sensor, they will be able to be transformed into the world system. Thus, they will be used to set up two independent view definitions. The direction of the view definition is constant and is determined by the normal vector of the projection screen.

Hand tracking yields a transformation matrix for the sensor connected to the input device. This matrix provides the necessary data to implement user interaction.

Input devices

Input devices for VEs are designed specifically for spatial interaction, i.e. interaction with a VE. One of the simplest interaction devices is the so-called stylus, a tracked device that resembles a pen. In addition to the tracking sensor, it is equipped with just one button. Its simplicity makes it a universal input device that is widely used for selection, manipulation, and creation tasks. However, the ergonomics of the stylus design and of similar devices currently available needs to be improved. This is a topic of current research on input devices.

In addition to such universal devices, special-purpose devices, referred to as props, have been developed which are adapted to specific interaction tasks. For example, the Cubic Mouse [FPW+00] supports controlling cutting planes with three sticks in interactive visualization applications. The virtual palette [CW99] combines the use of a tracked transparent plate, held with the left hand, with the use of the stylus, in order to select items displayed at the location of the palette with the stylus.

All these devices have to be gripped and laid down with the hand just as ordinary objects. On the other hand, consider the data glove. In addition to a tracker mounted to it, it has sensors for the joints of the fingers. After the glove has been calibrated, which has to be done individually, hand gestures such as making a fist or pointing can be captured. Grabbing virtual objects is intuitively realizable. As a *general* input device, the glove however has not fully acquired acceptance from users of VEs. Unreliable gesture recognition, the necessity of recalibration for new users, and the feeling of discomfort caused by the tightly clinging glove seem to be the main reasons for this.

The most sophisticated devices currently available for VE interaction are force-feedback devices. They are actually both input and output devices. As their name indicates, these devices simulate tactile sensations. Examples of such sensations are material characteristics when touching a surface, or the simulation of deformation forces. Some devices have to be attached to the arm and hand. Others are mounted to the VE system and consist of a system of axes and joints, and a handgrip, which can be grabbed and released easily.[2] Despite the additional

[2]Usually, a safety pedal or other safety mechanisms are available.

feedback mechanism, manual interaction benefits from constraints on hand motion introduced by force feedback devices. These advantages come at a price, however. Apart from user comfort considerations, integrating such devices into PBVE systems, like the Responsive Workbench, results in two severe drawbacks. To begin with, visible parts of the devices interfere with the virtual scene, thus destroying the depth information and disturbing stereo perception. Secondly, imposing constraints on hand motion means that hand movements are limited to the working area of the device, which is usually too small to cover the entire relevant area. Furthermore, without an additional device, the hands could move much faster where they are needed. Moreover, a large piece of hardware would probably be in the way in such situations.

For the purpose of VE-based geometric modeling, the stylus, the glove, and force-feedback devices seem to be suitable in general, because they mainly support hand-based interaction. However, specific preference of device used depends on the modeling approach. Drawing curves seems to be a task for the stylus, whereas sweeping a surface may be easily accomplished using a glove. As stated before, the usefulness of force-feedback is questionable, at least for creation tasks such as drawing and sweeping. Furthermore, a large device installed close to the working area should not hinder hand motion.

Input devices for the modeler The modeling approach presented in this work makes use of general input devices. The right hand uses a stylus, whereas the left hand uses a tracked input device that has a simple shape and is equipped with three buttons. The reason for choosing general input devices is that the modeling approach chosen here consists of more than shape creation and shape deformation tasks. In fact, the modeling process is characterized by successive changes between several tasks, including system control. In chapter 6, an interaction scheme for the modeler is presented.

Graphics computers

Although graphics computers are the driving component for a VE system, to benefit the user they should be hidden in the background completely. It should be possible for the user to interact with the VE using the functionality and the features of the VE, rather than being required to operate the computer through the desktop interface.

Concerning the requirements of graphics systems, numerical performance is as important as graphics performance for applications that are based on complex simulations.

Consider for instance, a modeling application being run in a PBVE, where objects are designed from scratch. From the designer's point of view, there are two main requirements. In situations when the designer inspects the model by looking at it from all sides, it is important that the perspective projection is updated without any significant latency. Otherwise, the designer would recognize gradually decreasing perspective distortions immediately after he has moved his head. For the visual evaluation of a shape, this would be a serious hindering factor. That means that the graphics engine must be able to generate a frame rate of about 15 to 20 frames per second. Furthermore, in situations where the model is manually edited, the achievable interactivity depends on the available numerical performance as well, especially for physically based simulations.

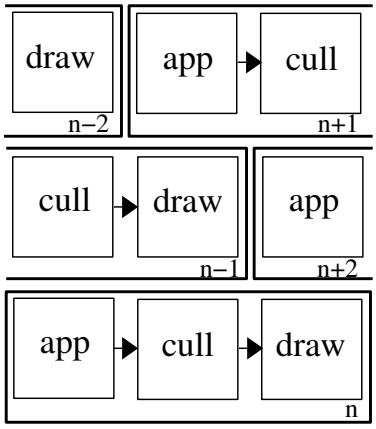

Figure 1.4: The stages of the rendering pipeline. Each frame passes through the stages *app*, *cull*, and *draw*. In parallel to drawing frame n, frame $n + 1$ is culled, and frame $n + 2$ is in the application stage.

1.2.4 Software components

A common method used to implement high performance graphics is the rendering pipeline. This is used in the widespread graphics framework SGI Performer [PERF], which is based on scene graphs. The task of the pipeline is to generate an image of the scene graph according to a view definition. Both the view definition and the scene graph can dynamically change. Therefore, the pipeline consists of three different stages, namely *application*, *cull*, and *draw*. These stages are organized as three different parallel processes, which are assigned to three processors in the ideal case. In parallel to the draw process rendering frame n, the cull process works on frame $n + 1$, and frame $n + 2$ is assigned to the application process, as shown in Figure 1.4. The application process maps user input into changes of the scene graph, and into new object attribute values. The cull process removes parts from the scene that are outside the viewing frustrum, and the draw process forwards the remaining polygons to the graphics subsystem, where they are rendered. In many VE applications, where a huge amount of polygons form the scene, the duration of the draw process determines the maximum possible frame rate. In complex interactive applications, such as in geometric modeling, the application process is the most critical stage. As a consequence, the choice of approaches suitable for geometric modeling in VEs is restricted to methods that allow an interactive frame rate. The modeler proposed in this work takes into account these requirements.

The scene graph is a direct acyclic graph whose inner nodes contain transformation matrices, or grouping functions, and whose leaf elements represent the polygons. These polygons are organized as triangle meshes or independent polygons, together with attributes specifying material and lighting. The virtual reality framework “Avango” [Tram99] encapsulates the object attributes into so called field containers and provides attribute bindings using the interpreted programming language Scheme. Scene definitions as well as various display configurations are written in Scheme and can be loaded into the application, making the application independent from the display setup. The Scheme binding enables a very powerful runtime environment, providing full application control for the user. Furthermore, distributed applications are supported using the field container concept as you can see in [Tram99].

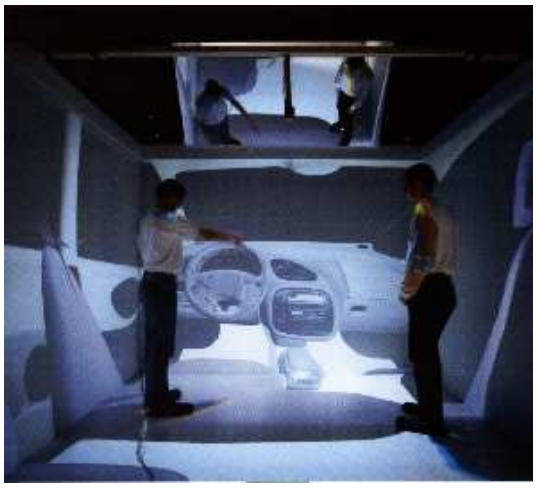

Figure 1.5: Review of the interior design of a car, from [GWWF00]. Data courtesy of Volkswagen AG

1.2.5 Applications

In recent years, virtual environment applications have in part reached the level of productive, industrial use. The advance of display technology and graphics as well as numerical performance has driven VEs toward reaching the requirements of engineers, designers, geophysicists, managers, or medical doctors. However, the potentials of VEs have still not fully been exploited, which is probably due to the lack of maturity among current 3D user interfaces.

Application fields

Virtual environments are used in engineering, medicine, entertainment, and for interactive exhibits. Engineering applications include: visualization of 3D simulation data such as computational fluid dynamics (CFD) and structural mechanics. As an example, consider the 3D visualization of CFD data simulating the thermal comfort in a car cabin. The support of VEs for industrial design processes is currently restricted to visualization of physical prototypes. Further application fields include molecular modeling, simulation, and training [HDSG97, Broo99].

Industrial areas

The main motivation of the car industry for investing into VE systems is to reduce the need for physical prototypes. For example, VEs are increasingly used for the review of design studies. Depending on application requirements, different VE systems, such as CAVEs, or projection walls are applicable. In some systems the user naturally is surrounded by the virtual scene. In this case, a CAVE would be the best suited environment, see Figure 1.5. Here, users immersed in a car cabin can inspect its interior turning the head and looking at different directions. As long as there is no need for such kind of immersion, a single projection screen would be a fully sufficient, and cheaper, alternative. Consider e.g. the task of reviewing the external shape of a car body design. The spatial relationship of the user and the model in this case is completely different. Quite reasonably, the user always is located outside the car and can align the model as needed. Therefore, there is no need for additional projection screens placed sidewards or behind the user.

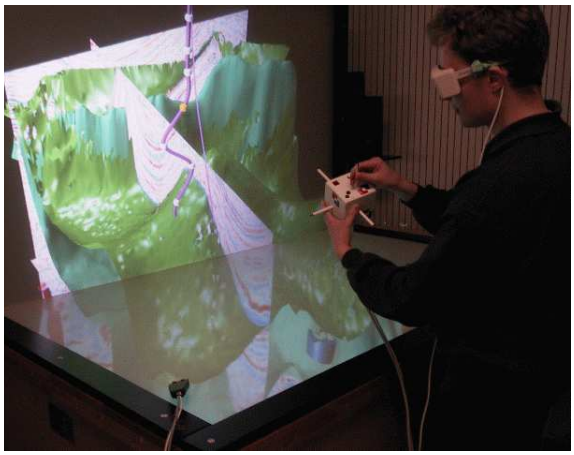

Figure 1.6: Interactive exploration of seismic data on a two-sided Responsive Workbench, from [FPW+00]

Usually, a tradeoff between the achievable frame rate and the visible surface quality of the model has to be made. The frame rate is important for a continuous update of the perspective, when the user is moving his head. The model loaded into the application is generated out of original CAD data by a tessellation step taking into account both requirements. Attempts were made to integrate complete CAD systems into VEs, by supporting stereo viewing on large displays [Bert99]. However, these CAD systems are not optimized for high frame rates. Therefore, such VE applications have a very restricted degree of interactivity.

Oil and gas companies use VEs for the visualization of seismic data. Pilot applications in this area have been developed, and it is expected that they will be routinely used in the near future. The analysis of user requirements in this field has led to the development of special purpose input devices for interactive data exploration and manipulation. The Cubic Mouse [FPW+00] has been designed for the control of cutting planes for visualizing 3D volumetric data and is used in a workbench environment. It is additionally useful for moving control points of drill paths, as shown in Figure 1.6.

Medical applications in VEs include real time data visualization, telepresence, rehabilitation therapy, and psychological treatment. Moreover, projection systems integrated in the operational theater augment the operation situs with virtual anatomical structures, in order to support the surgeon during the intervention [GTB+04].

Within these domains, VEs significantly influence important decisions and help to increase the efficiency of industrial processes and reduce costs. Therefore, the investment into a VE system is often more than compensated for, as users in the oil and gas industry argue it.

Furthermore, Virtual Environments provide important military capabilities in areas such as battle management, command and control, and simulation.

Manipulation and Interaction

The virtual scene in the described application fields mainly consists of precomputed data that is loaded into the system. The possible ways to manipulate geometry directly in the VE are mostly restricted to 3D transformations and changing visualization parameters. Extending a CAD system with stereo viewing capabilities is not considered a VE application because interactive frame rates that are needed to support head tracking cannot be achieved with full model complexity. Furthermore, user interaction mostly occurs through a separate desktop in such applications.

Chapter 2

Conceptual Shape Design

In this chapter, a brief overview about the current conceptual phase of industrial shape design processes is given. Traditional as well as computer based methods are discussed.

It is evident that at the intermediate design stages a digital model will be needed. Transforming a 2D paper sketch or a 3D physical model into a digital model is in fact one of the most challenging tasks of current design processes. To enhance the efficiency and to reduce costs it would therefore be desirable to have a digital model available from the beginning. To achieve this, designers need computer interfaces and tools that allow them to use their creative skills at the initial stages of the design process, even enabling them to start from scratch. Virtual environments have the potential to be developed further in that direction.

In his dissertation, Gribnau [Grib99] discusses aspects of computer-supported conceptual 3D modeling similar to those presented here. He considered the use, the benefits, and the requirements of computer support for conceptual modeling, and found that standard CAD systems are insufficient to support the conceptual design phase, similar to the statements of Deisinger et al. [DBW+00]. However, the work of Gribnau aims in a different direction, which is improving the usability of CAD systems. This is achieved by utilizing two-handed interaction for desktop systems that use standard computer screens.

Compared to the work of Gribnau, this chapter presents the potentials of virtual environments to serve as a user interface to CAD functionality. Taking into account improvements in the field of 3D printing technology, the potentials of alternative future design processes are discussed.

Despite the fact that industrial shape design processes cannot be completely specified, a number of consecutive design phases can be identified. These phases are characterized either by creative or by engineering activities.

The ideation and the conceptual phase The design process begins with the *ideation (idea creation)*. Various design ideas are communicated through sketches or illustrations. Quickly generating and evaluating multiple concepts is the main characteristic of these initial design stages. Traditional methods such as drawing on paper or building physical models are the predominantly used methods in the conceptual design phase.

The elaboration phase consists of detailing a selected concept, including high-quality surface rendering, and quantitative measurements.

The engineering phase When the engineering phase starts, the creative work on the shape of a product has been completed. During this stage of the

design process, the designer has to ensure that engineering and manufacturing requirements are fulfilled.

2.1 Traditional methods

Conceptual design is very much dominated by traditional methods. These methods are strongly based on the skills of a designer, who often spends years to learn a specific technique [Grib99]. The most important means a designer is using to translate an idea into a shape are the hands. This is an important observation, since it illustrates why traditional methods usually are preferred over computer systems for conceptual design.

2.1.1 Sketching

In the early stages of shape design, the most important tools for a designer are pieces of paper and pencils. Freedom, and directness characterize the nature of sketching: an idea of a shape, evolving in the mind of the designer turns into concept drawings on paper. The activity of drawing and sketching itself is an important means for the designer to develop a more elaborate idea of a new shape; in other words, a sketch is not the result, but rather an integral part of a creative process.

2.1.2 Traditional Modeling

Modeling means creating a physical object that is suitable to communicate the shape of a product in a more detailed, and more concrete form than a paper sketch. The object does not need to necessarily resemble a three-dimensional counterpart of the complete product, and the model does not need to be a solid object. It can also represent e.g. a cross section of the 3D shape, or it can be made as a wire frame [Grib99]. The feedback on the shape of a product is enhanced by the fact that the designer uses both hands to build the model.

Building physical models contributes significantly to the conceptual design process, and is used in parallel to sketching on paper, or after more concrete drafts of a possible shape are available. A wealth of hand-based methods of building physical models exists.

Architects build models of building complexes using materials such as paperboard, paper, or styrofoam. Correspondingly, their tools include such simple things as pencils, rulers, and scissors.

Another example is the use of traditional modeling for cars, as described in detail by Tovey [Tove92]. In summary, scale models of cars are built by carving away pieces from clay-like material, using a specialized set of tools. This procedure involves an enormous effort, regarding labor, cost and time. Moreover, changing the design would eventually mean starting the process from the beginning.

Another traditional method used in the car industry is known as tape drawing. Designers create full-scale drawings of the principal curves of the car body on large, vertical surfaces. The principal curves of a car are those few essential curves that define the characteristic shape and styling of a car body design [GBK+02]. The term *tape drawing* stems from the fact that traditionally instead of using pencils, tape was fixated on the drawing surface using both hands. Appropriately trained designers are able to create smooth continuous curves using this technique. Another advantage is the full scale of such kind of models.

What is common to all design methods is that the quality of the result depends on the skills of the designers that have built it. These abilities include spatial imagination, artistic and creative skills, fine manual motor skills, as well as a sense

for esthetics. In short, a designer must be able to imagine a shape and to create it using the hands together with simple auxiliary means. Furthermore, experience is an important factor for being able to apply specialized techniques accurately and rapidly.

2.1.3 Conversion into a digital model

The later phases of the design process require digital models that are used for simulations such as improving the surface quality, finite element analysis, stress analysis, and for the purposes of computer aided manufacturing. This means that somewhere in the process chain, a digital model has to be either directly created, or more commonly, generated from physical models or two-dimensional drawings.

There are different approaches that convert a physical model or a 2D drawing into a digital model. They share several things in common. For example, user-support is intensively needed in order to control the process, or to manually correct the result obtained from algorithms, using CAD functionality.

Overbeeke et al. [OKHS97] for example, reconstruct a 3D digital model from a 2D sketch, using the line of sight as an auxiliary means to obtain a network of 3D curves, that describe the "edgelines" of the model. From the curve network, a surface of the model can be generated. The work of Overbeeke et al. is not very detailed. Principally, the 3D curves have to be reconstructed from 2D curves that the user draws, by projecting a scan of the paper sketch onto a drawing board.

In order to obtain a digital model from a solid physical model, 3D digitization methods are available. There are contact-based, as well as non-contact based methods. One example of a non-contact based method is laser scanning. Such processes take time, and are prone to errors. Especially since the 3D scanning devices have to be adjusted and operated carefully. Post processing the scanned data is required to obtain a suitable description of the surface. Also, the pure geometric information has to be complemented with topological and semantic information afterwards.

2.2 Computer-based methods

Although computer aided conceptual design tools currently are not preferred over traditional methods, their use has important benefits. Recently, certain traditional activities of the design process, which have only slightly changed over the years were successfully computerized [BFBK00]. In addition to such approaches, the use of virtual environments may allow computers to offer a wider support rather than for specified purposes.

2.2.1 The need of computer support

Whether or not there is need of computer support for conceptual shape design contains two aspects. Clearly, a digital model is needed for the later phases of the design and engineering process, as argued before. For this reason there is need for computer support. A separate question is whether the creative shaping process would benefit from computer support in the conceptual phase.

The advantages of using computer-supported conceptual design systems have been worked out by Gribnau [Grib99]. He emphasizes the following aspects:

Variations It is easy to maintain variations on a design, and preserve their history.

Access A virtual model can easily be shared among several users, including distributed users in collaborative application scenarios.

Freedom Using virtual models, designing a shape is not constrained by the limitations of materials and tools from the real world. This means that the possibilities to create a shape are wider [Grib99]. Notice, however, that the available shaping tools may depend on the actual representation of the digital model.

Flexibility With a CAD system, the designer can choose the scale of a model representation according to the needs of the editing task in a flexible way. It is easy to work out details after zooming into the corresponding parts of the design, or switch back to the overall view of the design. On the contrary, the scale of a physical model is determined from the start.

Dynamics It is easier to communicate the dynamical aspects of a design, using animated sequences, or even interactive virtual product presentations.

Despite these benefits, computers usually are not used in the conceptual phase of design, although they are intensively used in the later phases. The main reason is that current user interfaces do not directly support *creative, manual shaping*; in other words these interfaces do not make use of the *freedom of form* that could be made available. A closer look on how CAD systems are currently used illustrates their drawbacks more clearly.

2.2.2 Interaction with CAD systems

Using a standard CAD system for conceptual design means that the creative shape forming process depends on using a mouse, a keyboard, and eventually a graphics tablet. Instead of manually forming a piece of material, one hand is holding the mouse; the other hand is either resting on the desk or entering information on the keyboard. The eyes are looking at a computer screen, where a small virtual model is represented at a size that is smaller than one hand. Thereby, the head is almost mounted at a fixed position. Fine motor skills that otherwise would help the designer to elaborate shape details, are now needed to hit icons with the mouse pointer. Drawing using a conventional mouse is nearly impossible. Drawing on a graphics tablet using a pen-like input device seems feasible, yet unnatural, since it requires looking at the screen while drawing on a separate surface. This interaction scenario typical for desktop applications is known as the "WIMP" interaction style (windows, icons, mouse, pointer).

Furthermore, most of the functionality of current CAD systems is inadequate, or not needed for the purposes of conceptual design. Most offered functions force the designer to think in mathematical terms, as also stated by Deisinger et al. [DBW+00]. Shape attributes that the designer can modify through the user interface often directly reflect the mathematical representation of objects. For example, in a spline-based modeler, control points, knots, tangents and weights can all be manipulated. Such powerful interfaces are more appropriate for the detailed elaboration of shapes.

Special purpose conceptual design systems, such as the Alias|Wavefront Studio Tools, provide designers a more adequate set of tools, but are still desktop oriented. Using a graphics tablet and a pen, restricted support for conceptual design is available.

On the other hand, considering the benefits of computer support (section 2.2.1), and the complex process of transferring physical models into virtual models (section 2.1.3), using computers for conceptual sketching would be highly desirable. The key problem is finding suitable user interfaces capable of forming a link between the designer's skills and powerful algorithms.

2.2.3 Novel computer-based methods

One possible approach to attack this task is to offer computer support for specific tasks of an industrial design process such as providing computer-aided tools that are based on traditional methods. Unlike the use of standard CAD systems, these tools would be tailored to the designers' skills. The designers have acquired these skills over years, and therefore it would be very easy for them to learn using such tools. They could be integrated into design processes easily and be efficiently used since a digital model would be obtained using traditional methods.

Obviously, there are no straightforward methods of transferring traditional ways of modeling to a computer. However, there is work where such a transition was successful.

Balakrishnan, Buxton et al. [BFKB99, BFBK00] developed a tool for the support of tape drawing, a method of modeling the principal curves of car bodies by gluing tapes onto a surface, see also section 2.1.2. Using a mono-view powerwall and a tracked input device for each hand, Balakrishnan et al. make available the traditional interaction technique of tape-drawing for the creation of virtual curves. The user draws the curves directly on the screen in full scale, using the corresponding two-handed interaction technique. Balakrishnan et al. report that the designers indeed were able to transfer their skills to the new system.

Grossman et al. [GBK+01] extend this work, using a 3D virtual working volume that allows for a direct creation of a 3D model. Planar cross-sections of this working volume are shown on the display system, still a mono-view powerwall, on which the 2D curves are drawn. A restriction of this tool is that only planar curves are supported, which was, however, removed by the same authors recently [GBK+02]. In their new tool, spatial curves are drawn using a two-step method. The shape of a spatial curve in the third dimension is defined by drawing a depth-curve on the display surface as a 2D curve. The depth curve forms an extruded surface, onto which the actual curve, also drawn as a 2D curve, is projected. As a result, a spatial curve is obtained.

2.2.4 Modeling in virtual environments

Another possible approach of making use of computer support in the early stages of design is to utilize the designer's skills on a more general level. Instead of computerizing specific tasks, as it has been done in digital tape drawing, conceptual design could benefit from integrating computer-based forming tools with virtual work environments.

Novel user interfaces based on this approach would make use of the designer's ability to use the hands. However, in such a way that is not necessarily restricted to simulating traditional work methods. One important benefit of computer support for conceptual design that has been identified by Gribnau (see section 2.2.1) is the freedom that computers have made available for shape design not constrained by the limitations of materials and tools of the real world. For example, curves or surfaces can be formed by performing drawing strokes directly in 3D within a virtual environment. The virtual model represents a true 3D design.

On one hand, it can be argued that such methods deviate from the way designers usually work. On the other hand, the possibilities of computer support can be enhanced. All advantages of digital models could be utilized while offering a user interface that supports the designer in several ways. Such support may include head-tracked stereo-view and direct manual interaction tools that allow the use of both hands. Fundamental aspects of traditional sketching and modeling methods, such as two-handed work and large-scale interaction, can be preserved using virtual environments.

However, these interfaces and customization by designers to traditional methods meet only halfway. Novel shape creation or deformation tools that are applied directly on a virtual model clearly require some acquired skills by the designer. In case developers of virtual environments would enable designers to make creative use of such tools, the requirement to learn would not be a criterion against their use. This would be true most likely because it is commonly accepted that traditional methods may require extensive training.

Thus, conceptual design tools based on virtual environments have the potential to benefit the design process. These potentials and restrictions are discussed in section 2.3.

2.2.5 Conversion into a physical model

It was argued that a digital model is needed for further stages of the development process, including the elaboration phase of shape design. However, designers evaluate a shape in more ways than just visually observing. The assessment of a shape is also strongly based on haptics, especially in the later phases of shape conceptualization. Haptic feedback could be made available in virtual environments, but would not be able to completely replace a physical model. Consider e.g. a hand-held product, where it is important to give a professional opinion on the ergonomics of a device.

Therefore, there is need for physical models in the design process as well. Compared to traditional model making, mechanical processes by which materials are shaped or formed are available to generate a physical model from a digital model automatically. It might be necessary to generate several versions of a model in the conceptual phase of design, so the need for inexpensive solutions exists.

Currently, three-dimensional printing technology is emerging, while the printing hardware is becoming more affordable. 3D printers work much like a laser printer. The digital model, given in the standard STL format, is sent to the printer, which forms layers out of talcum powder or flour substance in a few hours [Harr]. The printers provide good quality, highly detailed models that are sufficient for client presentation, and for design evaluation. The relatively short printing cycles and the cheap materials allow the refinement of the design, using several test prints. Alternatively, other processes are available that use high-grade materials and produce objects of improved quality. These processes take a much longer time and are too costly to be used on a trial and error basis. For an overview about this technology, see e.g. [BCMS03].

3D printing technology is still at its beginning stage therefore it is too early to predict how successful it will be. However, compared to the earlier process, called stereo lithography, its advantages are promising. Stereo lithography creates a solid plastic object by hardening layers of liquid polymer plastic. The drawback of this method is that both the machine and the material are very expensive. Furthermore, the process is very slow. As a result, stereo lithography is not widely used.

2.3 Shape design in virtual environments

Shape modeling applications running directly in a virtual environment are not very common, but they are currently gaining increasing attention. Product designers recognize the capabilities of virtual environments to support shape creation. The question of whether creative shape design in VEs is superior to desktop modeling depends on the supported task.

2.3.1 Potentials and restrictions

In order to identify the potentials as well as the restrictions of PBVE systems to support interactive shape design and geometric modeling, a closer look at the setup design, and at the technical components is necessary. These are both described in section 1.2.

Essentially, the goal of supporting shape design with VEs is to provide a novel interface to a computer. Regarding user input, VEs react on spatial gestures or postures performed with the hands, rather than on keyboard inputs. Regarding graphical output, stereographic, high-resolution images aim at giving the user a realistic, large-scale 3D impression.

Considering input aspects, on the one hand, such environments seem to have great potential to support the *creation* and *modification* of geometry. These tasks even further benefit from a head-tracked, 3D representation of the model, which can potentially help the user to understand the shape of a 3D object. The possibility to move the head and turn the object at any time particularly enhances the perception of shape.

On the other hand, the technical solutions for stereo viewing, hand and head tracking, and input devices are still underdeveloped. User discomfort is caused by various ergonomic problems such as cables that hinder free movement, or the stereo glasses, which are not individually adapted for fitting.

While those technical shortcomings may be reduced in the future by cable less tracking systems and wearer friendly stereo glasses, there are two principal problems. Firstly, an update lag due to the computation time needed by the draw process is always present, which results in a frame sequence that is slightly outdated. This is noticeable to the user, who when moving or turning his head perceives a slightly incorrect moving 3D model. For designers, a wrong perspective is a heavily disturbing factor when exploring a design, and it has to be avoided.

Secondly, direct interaction with virtual models, as implied by VEs, naturally means "interacting in the air", which in the first instance is an unfamiliar way of using the hands, at least for people that have not become accustomed to VEs. Two principal solutions to this problem exist. The first is the use of haptic feedback devices, discussed in section 1.2.3. The second approach, pursued in this work, aims at designing interaction methods and tools adapted to using the hands freely in the environment. This means that these methods cannot directly be derived from known modeling methods, yet still should support the creative use of the hands.

2.3.2 Designer's needs

The development of modelers that run in a VE should be based on the needs of designers. To achieve this, Deisinger et al. [DBW+00] invited a group of designers representing the fields industrial design, product design, car design, jewelry design and other designs. 47% of the designers had former experiences with VEs. The designers, including an early version of the modeler described in this dissertation, have tested three different immersive modeling approaches, running in a CAVE. By analyzing the subjective statements made by the designers about important criteria for the design process using immersive modeling and about the particular modelers, the authors derived the following guidelines for the development of an immersive modeler. [1]

1. A tool that combines the conceptual phase with support for a certain degree of elaboration would be useful.

2. A plausible transfer of design ideas into the digital model must be possible.

[1] Deisinger et al. use the term *immersive* modeling instead of VE-based modeling.

3. Mechanisms to support working under constraints, similar to real life experience
4. Direct and real time interaction
5. Full scale modeling, large working volume
6. Intuitive, easy-to-learn

To be superior to the desktop oriented tools currently in use, the potentials of VEs have to be exploited, whereas restrictions need to be overcome by new technical solutions. This does not mean that a VE application could completely replace traditional methods such as drawing and painting on paper. However, from the designer's point of view, there is a clear need for simple, intuitive 3D computer aided sketching systems, and for more supportive user interfaces.

2.4 Future design processes

In this chapter, two alternative conceptual design processes have been discussed. The traditional method builds sketches and physical models, whereas the computer-based method creates digital models. It was argued that there is a need for both digital as well as physical models in the design process. Digital models are required for integration with the later stages of development. A purely virtual modeling approach, however, is not realistic since physical models are of great value for the assessment of the shape, both visually and haptically.

The traditional methods are preferred over the use of computers because forming a shape is a highly creative process that is not adequately supported by current CAD systems. However, traditional modeling processes are slow, inflexible, and take a vast amount of time. The necessary digitizing step is very involved and difficult to operate. Creating variations of a design, which is an essential process of shape conceptualization, requires too much effort to use such procedures.

Computerizing specific tasks so that designers can apply their methods in an unaltered way may provide successful solutions, but is not generally applicable. Compared to this, virtual work environments have potentials that may allow them to be developed further into user interfaces to conceptual design systems.

Assumed that such interfaces are available, and considering the evolving 3D printing technology, future design processes are conceivable that combine creative forming skills with rapid prototyping of physical models. Described in a simplified way, such a design process would contain the following stages:

1. Eventually sketch shape ideas on paper.
2. Start the design of a virtual model from scratch, using a virtual design environment that supports spatial interaction and provides a head-tracked stereoscopic view.
3. Generate physical models using 3D printing technology.
4. Evaluate the form and apply modifications to the virtual model, then print new models.
5. When the shape corresponds to the ideas of the designer, print out high-quality models and present them to clients and/or to decision makers.
6. When the design has passed the evaluation, export it to a CAD/CAM system.

2.5 A Survey of 3D Modeling Applications

2.5.1 Common principles

In most of the surface modeling systems for PBVEs proposed so far the user acts in a very direct way. With these tools, the designer immediately converts a shape idea into a surface using a corresponding gesture.

Such a direct approach for surface sketching is "Surface Drawing" by Schkolne and Schröder [SS99, SPS01]. Polygonal surfaces are created by moving a hand that wears a glove, through space on the workbench. Similarly, Stork [SdA00] presents a modeling system, called "Arcade", which is implemented on a workbench and supports sweeping Coons surfaces from drawing strokes. The Coons patch is a simple free-form surface approach based on bilinear blending between the four boundary curves [Fari97].

On one hand, quick initial sketching is elegantly supported by such approaches. They are very easy to learn, and do not bother the designer with obstructive user interface overhead. On the other hand, such spontaneous modeling often results in virtual objects that look unfinished and have a cluttered appearance. They often lack a satisfying surface quality, or important constraints such as symmetry, or continuity, are missing. The imposed restrictions of such systems can be summarized as:

- Only basic functionality is provided.
- The support for elaboration is restricted.
- Since the input strokes are unconstrained, it is almost impossible to stitch surfaces continuously together.

The modeler that is presented in this work (described in chapter 4, 5, and 6) differs from previous VE based modelers in that the initial surfaces are created indirectly by modeling the geometry and the topology of a network of contour curves. Since the surfaces do not result from spontaneous sketching, it is easier to ensure constraints, and therefore more complex models can be created from scratch. Moreover, providing elaborative functions only makes sense if the initial model already has useful geometric properties, e.g. symmetry, and an acceptable surface quality. These aspects are additionally important for further processing of the virtual model in the subsequent stages of the design process.

2.5.2 Overview of current applications

Voxel-based applications

Krause and Lüddemann [KL96] present a voxel-based virtual clay modeler, allowing the designer to generate virtual material or take it away from the model at arbitrary locations. The drawback of voxel-based solutions is that manipulations performed in a different direction than that of a coordinate axes can be slow or can produce unwanted shape details. In addition, if the visualization of a smooth surface is required as a result, the corresponding calculations can be rather slow.

Spline-based applications

In 1976, Clark [Clar76] built a system which used a head mounted display (HMD) and some buttons to design bicubic patches rendered as line-drawings. It seems to be the first modeling application running in a VE.

The first work that proposes the idea of two-handed interaction and editing curve networks, as in our system, is "3-Draw" from Sachs et al. [SRS91]. This is

a desktop oriented system. One hand controls a tablet while the other hand draws curves on the tablet. Both hands are electromagnetically tracked. The system lacks surfaces, so that only curve networks are shown.

Van Dijk [vD93] developed a similar system. It is called "Fast Shape Designer", and is based on irregular networks of NURBS curves. Only 3-sided or 4-sided surfaces are allowed.

Usoh et al. [USV96] define spline-based surfaces in a HMD environment by sweeping them with the hand and manipulate them by simulating deformations.

Three publications exist about the modeler described in this dissertation. In [WD00], deformation tools for curves and surfaces are presented (see also chapter 5). An overview about the application as a whole can be found in [WS01]. An interaction technique for the support of direct manipulation and system control tasks is proposed in [Wesc03].

Subdivision-based applications

Kobbelt [Kobb00] presents a new method for the generation of fair triangle meshes which are optimal with respect to a discretized curvature energy functional. He introduces variational subdivision schemes, which support the generation and the deformation of high quality meshes. The resulting models show fully detailed surfaces of good quality.

Particle-based applications

The "Skin" approach [MCCH99] proposes a particle-based surface representation for sculpting free-form surfaces. Users interactively guide the particles to form triangulations suitable for subdivision, so that a smooth limit surface can be obtained.

Tonnesen [Tonn98] presents a new modeling technique, based on dynamically coupled particle systems, for creating and manipulating complex three dimensional polygonal meshes in a fluid like manner. The system has been further developed for use at the Responsive Workbench.

Applications based on polygonal surfaces

The "THRED" system by Shaw, Green et al. [GLS95, SG97] is a simple two-handed desktop free-form editor.

Dani and Gadh [DG97] present a desktop VR system for design, which is called "COVIRDS". They use a combination of hand gestures, voice input, and keyboard input to create and manipulate a 3D artifact. They have extended their work for projection-based VR systems.

The "Teddy" sketching interface [IMT99] allows quick and easy designing of free-form models. From several 2D strokes, plausible polygonal surfaces are automatically constructed.

Other applications

"JDCAD" by Liang and Green [LG94] is a 3D solid modeling and animation system.

An approach which relies on pictographic gestures describing superquadrics has been realized by Nishino et al. [NUK98]. It is a 3D modeling system running in a projection wall environment.

Interaction and drawing techniques

The first two-handed interaction techniques for the Responsive Workbench have been introduced by Cutler et al. [CFH97].

Zeleznik et al. [ZHH96] describe a gesture based design system for 3D objects. It does not yet support free-form curves or surfaces.

Buxton et al. [BFBK00] contribute a styling approach that simulates the tape drawing process used by automotive designers, called digital tape drawing. The principal curves of car bodies are drawn on the surface of a powerwall display, using tracked input devices and mono view. This application had the restriction that only planar curves were supported. Grossman et al. [GBK+01] recently extended it with a method that allows drawing of spatial curves, using two planar drawings. The work of Buxton, Grossman et al. is explained in more detail in section 2.2.3.

Chapter 3

Surface Sketching in Virtual Environments

In this chapter, the requirements for sketching free-from models in head-tracked virtual environments are identified. It is argued that there is a need of a continuous update of the perspective and of a continuous visual feedback showing the effect of applying a shape deformation in real time. This has consequences for whether a certain surface approach is effectively usable for sketching in a virtual environment.

It should be emphasized that the sketching method, as well as the used surface approach, need to be capable of forming complete free-form models. For example, although a wealth of sketching techniques are available for tensor product spline surfaces, the methods available for assembling complex models using that kind of surfaces are very limited. This is due to the methodical requirements that a sketching method for models should fulfill.

Taking into account these aspects, the most common surface approaches are presented.

3.1 Sketching free-form models

Usable solutions for sketching free-form models do not only depend on adequate shape creation and shape manipulation metaphors. They also depend on a suitable approach for representing the complete surface of the model. In addition, this representation should allow an interactive visualization of dynamic sculpting processes.

3.1.1 Technical requirements

Requirements on surface models that are supposed to be visualized and interactively modified in a virtual environment are in certain respects different from that on models presented on a computer screen.

In a head-tracked virtual environment, the perspective projection of the model must be updated continuously, in order to provide the user with an object that remains stable at its place, just like a real object standing on a table. To achieve this, an interactive frame rate is a must. This has to be ensured by a high performance rendering pipeline (see Figure 1.4 on page 11 for an explanation of the rendering pipeline). The load of the draw process can be reduced by a sufficiently low number of polygons that represent the surface model.

In an interactive modeling application, surfaces are dynamically being sculpted and deformed. These surface modifications are computed in the application process

of the rendering pipeline. The limitating factor for the frame rate is displaced to this stage; therefore, tesselation algorithms that efficiently generate a polygonal representation out of the surface should be available. They should be adaptable to the current load of the rendering pipeline, in order to ensure fluent updates of the image. For example, a reduced level of detail is required when the user is interactively changing the shape of a surface. In the moment when the interaction task ends, which reduces the load of the application process, a more accurately sampled model should be generated.

Parametric surface approaches that possess an efficient evaluation procedure, such as spline-based surfaces and, thanks to Stam [Stam98], also certain subdivision surfaces, namely Catmull-Clark surfaces and Loop surfaces, meet these requirements. They allow the use of efficient multilevel triangulation methods that refine an initial set of triangles based on an error estimation. In addition, the refinement can be controlled according to the load of the application and draw processes. The accuracy of the triangulation can thus be dynamically adapted.

3.1.2 Methodical requirements

In addition to the technical requirements, a surface approach needs to support the task of sketching a complete free-form model in a 3D environment from scratch. Even in the conceptual design phase, a model likely consists of several surface parts that need to be connected. Burdening the sketching process with interaction tasks that need explicit input of connection information would hinder a creative, fluent shape conceptualization.

From the nature of sketching, which is described in section 2.1.1, the following criterions could be derived that the surface approach and the sketching method should fulfill. It should be possible to

- input a shape quickly,
- change parts of the shape without having to resketch the complete shape,
- stitch together separate parts of a shape, and
- form complete models, not just individual surfaces.

It is noteworthy that, according to the work of Deisinger et al. [DBW+00], a certain support for the elaboration of the model would be desirable for a modeler running in a VE. This means that pure sketching functionality should be complemented by interactive shape deformation tools. Another requirement is that the mathematical details of the surface representation should be hidden from the designer. In case of spline-based surfaces, the designer should not need to deal with control points, knots, or weights when sketching a shape. These shape parameters are too low-level to be useful for quick input. Instead, high-level creation and deformation tools should be available for the chosen surface approach.

Regarding the surface quality, in the conceptual design phase an infinitesimal smoothness and higher order continuity transitions between neighboring parts of the model seem not to be of major importance. However, what is of particular relevance is a visually pleasing shape. This can require that fairness measures, and efficient algorithms to implement them, are available for the chosen surface approach. As a consequence of this, the designer should be supported by the system with methods that transform his sketched input, which initially might lack the desired surface quality, into a more pleasant shape.

3.2 Surface approaches

Available surface approaches only partly fulfill the various and conflicting requirements of surface sketching in virtual environments. In this section, the most important surface approaches are briefly presented and their suitability for surface sketching in virtual environments is discussed. The basics of spline surfaces, subdivsion surfaces, and curve networks are presented in more detail since they will be needed in the subsequent chapters for the development of an interactive modeler.

3.2.1 Polygonal meshes

Usually, polygonal meshes are a low-level type surface representation, required by graphics engines for rendering the surface using scan-conversion of triangles. For the purpose of model description, typically triangular or quadrilateral meshes are used.

However, it is interesting to note that on the one hand meshes can be obtained as output from a sampling process, and on the other hand they may appear as an initial input to a subdivision approach, resulting in a smooth limit surface. The rendering of this limit surface may again result in another mesh. The use of meshes as an input to a refinement procedure is particularly relevant in the context of geometric modeling, since it makes possible the computation of smooth forms out of roughly sketched input.

Sketching with polygonal meshes

Since polygon meshes are a low-level type of surface representation, they can be designed from scratch using various methods. Higher-level surface approaches that can be triangulated may be used for that task. However, polygonal meshes may be sketched directly as well, using drawing strokes from tracked input devices, as done in the "Surface Drawing" system [SPS01]. In that case, algorithms for stitching together surface parts resulting from several drawing strokes are necessary in order to avoid a cluttered appearance of the result. This can be time-consuming, since each polygon within the overlapping region must be processed against its neighbors. This can affect the interactivity of the whole system.

The use of triangular meshes for sketching will have several important advantages over the use of piecewise polynomial spline surfaces, if subdivision or discrete fairing capabilities are integrated, as proposed by Kobbelt [Kobb00]. Provided that a usable method for designing from scratch is available, e.g. "Surface Drawing", or the (triangulated) surfaces from Kuriyama (see 3.2.5), the shape of a model can be roughly created. A fair, pleasant shape can be obtained using subdivision and discrete fairing. The designer can work on an arbitrary triangle mesh and is no longer hindered by the restrictions imposed by tensor-product spline surfaces, which are described in the next paragraph.

Using polygonal representations directly has one drawback, since normally an explicit parameterization is not available. Therefore, the performance of the rendering step, i.e. the load of the draw process, depends directly on the model geometry. There is no means to quickly and dynamically adapt the amount of triangles to the achievable framerate.

3.2.2 Implicit surfaces

Implicit representations use a real, continuous, defining function $F(\mathbf{p})$ that assigns to each point $\mathbf{p} = (x, y, z)^T \in E^3$ a scalar value. With this function, the following

classification of points is associated:

$$\begin{aligned} F(\mathbf{p}) > 0 &\quad \text{if } \mathbf{p} \text{ is inside the object;} \\ F(\mathbf{p}) = 0 &\quad \text{if } \mathbf{p} \text{ is on the boundary of the object;} \\ F(\mathbf{p}) < 0 &\quad \text{if } \mathbf{p} \text{ is outside the object.} \end{aligned}$$

An implicit surface is the boundary of the object defined by this functional representation.

As an example of how implicit surfaces are used, consider the most popular algebraic surfaces, the *quadrics*. A quadric surface is the set of points $\{\mathbf{p}|\mathbf{p}^T\mathbf{Q}\mathbf{p} = 0\}$, where $\mathbf{p} = (x, y, z, 1)^T$ and $\mathbf{Q} \in \mathbb{R}^{4x4}$ is the matrix of polynomial coefficients. Varying these coefficients, primitives like spheres, ellipsoids, cylinders, and cones can be defined. Composite objects may be built using the binary operators intersection, union, and difference. The objects may be represented by a binary tree, in which the inner nodes contain the operators, and the leaf nodes correspond to the primitives. The operators can be defined in the following way [PASS95]: the intersection of two objects F_1 and F_2 is given by

$$\begin{aligned} F_1 \& F_2 = \min(F_1, F_2) &= \frac{1}{2}(F_1 + F_2 - \sqrt{F_1^2 + F_2^2 - 2F_1F_2}) \\ &= \frac{1}{2}(F_1 + F_2 - |F_1 - F_2|), \end{aligned}$$

the union is defined as

$$\begin{aligned} F_1 | F_2 = \max(F_1, F_2) &= \frac{1}{2}(F_1 + F_2 + \sqrt{F_1^2 + F_2^2 - 2F_1F_2}) \\ &= \frac{1}{2}(F_1 + F_2 + |F_1 - F_2|), \end{aligned}$$

and the difference can be written

$$F_1 \setminus F_2 = F_1 \& (-F_2).$$

Sketching with implicit surfaces

Implicit surfaces, particularly volume-based representations, where the surface is described using a distance function, have the advantage that model parts can be stitched together in a flexible way. The task of designing from scratch is supported by predefined primitive objects that can be loaded into the environment. The shape parameters such as the radius of a sphere, for example, can easily be specified.

The main drawback that currently prevents implicit representations from being useful for interactive sketching environments is that the separate surface extraction step cannot be performed at interactive frame rates. On request, the implicit representation may be converted into triangles before rendering, using isosurface extraction methods such as the marching cubes algorithm.

As a consequence, sketching with implicit surfaces cannot be immediately visualized, therefore a preliminary visual feedback of a drawing or deformation gesture needs to be generated. In certain cases, when only a small region is influenced by a deformation, the result may be obtained interactively. In the context of this work, implicit surfaces are therefore not relevant, and the reader is referred to [MWB+96] for further information about implicit surfaces for geometric modeling.

3.2.3 Spline-based surfaces

Spline-based surfaces are the most widespread surface class within the CAD community. The surface of many industrial products is constructed using spline surfaces: the shape of car bodies, household appliances, and various consumer articles.

One of the reasons for their success, and their biggest advantage over polygonal and implicit surfaces regarding the definition of free-form shapes, is that they are parametric surfaces represented in a closed form that possesses automatic smoothness. Moreover, the basis functions of splines allow the control point coefficients to roughly approximate the shape of the smooth surface.

Among spline-based surfaces, the most common concept is the tensor product surface that uses products of univariate curve basis functions and inherits many properties from the corresponding curve schemes. Therefore, to prepare the presentation of surfaces, Bézier curves and B-spline curves are discussed first.

Bézier curves

Bézier curves, independently developed by P. de Castlejau and by P. Bézier, are an ideal standard for the representation of polynomial curves. They lead to an easy geometric understanding of B-spline curves. A polynomial curve of degree n can be written in the Bernstein basis as

$$\mathbf{b}(t) = \sum_{i=0}^{n} \mathbf{b}_i B_i^n(t), \tag{3.1}$$

$t \in [0,1]$, with

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}.$$

That curve representation is referred to as an n^{th}-degree Bézier curve. The Bernstein basis functions $B_i^n(t)$ fulfil the following recurrence relation,

$$B_i^n(t) = (1-t)B_i^{n-1}(t) + tB_{i-1}^{n-1}(t),$$

with

$$B_0^0(t) = 1 \quad \text{and} \quad B_i^n(t) = 0 \quad \text{for} \quad i \notin \{0, \ldots, n\}.$$

As an example, consider cubic Bézier curves. They have the basis

$$\begin{aligned} B_0^3(t) &= (1-t)^3, \\ B_1^3(t) &= 3t(1-t)^2, \\ B_2^3(t) &= 3t^2(1-t), \\ B_3^3(t) &= t^3. \end{aligned} \tag{3.2}$$

B-spline curves

If the curve to be modeled has a complex shape, then its Bézier representation will have a prohibitively high degree. Moreover, although its shape can be outlined by means of its control points, the control is not sufficiently local. Such complex shapes can be modeled using composite Bézier curves that have a lower degree, also referred to as *B-spline curves*.

A B-spline curve of maximal degree n is defined by (see [Fari97])

$$\mathbf{d}(u) = \sum_{i=0}^{L+n-1} \mathbf{d}_i N_i^n(u). \tag{3.3}$$

The $\mathbf{d}_i$ are the control points, and the $N_i^n(u)$ are the B-spline basis functions defined using the non-decreasing knot vector

$$U = \{u_0, \ldots, u_{L+2n-2}\},$$

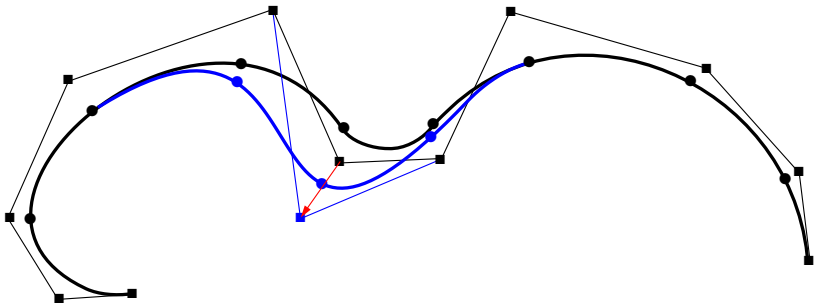

Figure 3.1: Local control for cubic B-spline curves (after [Fari97]). Moving a control point influences only four cubic segments, which are represented in blue. B-spline points are denoted by squares, junction points are denoted by spheres.

where L is the potential number of polynomial curve segments. Not all of the u_i have to be distinct. If r successive knots coincide, i.e. $u_i = \cdots = u_{i+r-1}$, u_i is called a knot with *multiplicity* r. A knot with multiplicity one is a *simple* knot. The curve $\mathbf{d}(u)$ is only defined over the *domain knot* interval $[u_{n-1}, u_{L+n-1}]$. In case all domain knots are simple, L denotes the number of segments or the number of domain intervals. The spacing of the knots can be *uniform*, i.e. $u_i = i, i = n-1, \ldots, L+n-1$, or *non-uniform*.

An nth-degree basis function $N_i^n(u)$ can be defined by linearly combining two basis functions of degree $n-1$ recursively:

$$N_i^n(u) = \frac{u - u_{i-1}}{u_{i+n-1} - u_{i-1}} N_i^{n-1}(u) + \frac{u_{i+n} - u}{u_{i+n} - u_i} N_{i+1}^{n-1}(u) \tag{3.4}$$

$$N_i^0(u) = \begin{cases} 1 & \text{if} \quad u_{i-1} \le u < u_i, \\ 0 & \text{else} \end{cases}$$

This recurrence formula can yield the quotient "0/0"; it is treated as zero. Note that the basis functions are completely defined by the knot vector and the degree.

The k^{th} derivative of a B-spline curve, denoted by $\mathbf{c}^{(k)}(u)$, can be expressed in terms of the k^{th} derivative of its basis functions,

$$\mathbf{d}^{(k)}(u) = \sum_{i=0}^{L+n-1} \mathbf{d}_i N_i^{n,(k)}(u).$$

Particularly,

$$\mathbf{d}'(u) = n \sum_{i=1}^{L+n-1} \frac{\mathbf{d}_i - \mathbf{d}_{i-1}}{u_{n+i-1} - u_{i-1}} N_i^{n-1}(u).$$

Hence, $\mathbf{d}'(u)$ is a vector-valued $(n-1)^{th}$-degree B-spline curve.

There are a number of very useful properties related to B-spline curves, which make them very attractive for free-form modeling and sketching. The most important ones are:

Automatic continuity At knots with multiplicity r, a B-spline curve is at least C^{n-r} continuous. At all other points, the curve is infinitely often differentiable. In particular, a cubic B-spline curve is twice continuously differentiable at simple knots. This automatic continuity of connected polynomial segments disburdens a curve designer from ensuring smooth transitions explicitly, which certainly is a very valuable property.

Local modification Moving a control point $\mathbf{d}_i$ influences $\mathbf{d}(u)$ only in the interval $[u_{i-1}, u_{i+n})$. For cubic curves, only the four nearby segments are changed, see Figure 3.1.

Control polygon The control polygon represents a piecewise linear approximation to the curve. A smooth B-spline curve can therefore be outlined by just setting a sequence of control points.

Affine invariance An affine transformation $\Phi(\mathbf{p}) = \mathbf{A}\mathbf{p}+\mathbf{v}$ is applied to the curve by applying it to the control points, i.e.

$$\Phi(\mathbf{d}(u)) = \sum_{i=0}^{L+n-1} \Phi(\mathbf{d}_i) N_i^n(u).$$

The affine map Φ is a combination of translations, rotations, scalings, and shears, with $\mathbf{A} \in \mathbb{R}^{3\times3}$ and $\mathbf{p}, \mathbf{v}$ points and vectors in the Euclidean space E^3.

Affine parameter transformations Applying the parameter transformation $au_i + b, a, b \in \mathbb{R}, a \neq 0$ to the knot vector U does not change the shape of the curve and the continuity at the knots, since only distances of knots are used to define the basis functions.

The following important geometric algorithms related to B-spline curves are informally repeated here. For detailed descriptions of the algorithms, the reader is referred to the literature, particularly to the book of Farin [Fari97] for a comprehensive introduction to curves and surfaces for CAD, and to the "NURBS Book"of Piegl and Tiller [PT97], who compiled an encyclopedic collection of methods and algorithms on non-uniform rational B-splines (NURBS), including notation in pseudo code.

Knot insertion It is possible to *refine* a B-spline curve by inserting a new knot into U, or by increasing the multiplicity of an existing knot. As a result of the algorithm, a locally refined control polygon that closer approximates the curve is obtained. The shape of the curve is not changed. Therefore, a designer can reduce the level of continuity at arbitrary parameter values, obtaining more local control close to that region.

Algorithm of de Boor The algorithm of de Boor uses knot insertion to evaluate an n^{th}-degree B-spline curve at a parameter value u by inserting u into the knot sequence until it has multiplicity n. This results in a control point of the refined control polygon that lies on the curve at u.

Piecewise Bézier polygon It is possible to generate a B-spline polygon that is the piecewise Bézier polygon of the curve by reinserting each given knot until it has multiplicity n.

Direct evaluation A B-spline curve can be evaluated at a parameter u in three steps by

1. Find the knot span $[u_{i-1}, u_i)$ in which u lies.
2. Compute the nonzero basis functions. At most $n + 1$ of the N_i^n are nonzero, namely the functions $N_{i-n}^n, \ldots, N_i^n$. This follows from eq. (3.4).
3. Multiply these values with the corresponding control points.

Curve ends and knot multiplicities In most cases, it is desirable to have both end knots u_0 and u_{L+2n-2} of full multiplicity n, i.e. $u_0 = \cdots = u_{n-1}$ and $u_{L+n-1} = \cdots = u_{L+2n-2}$. This causes the first and the last control points $\mathbf{d}_0$ and $\mathbf{d}_{L+n-1}$ to lie on the endpoints of the curve. Moreover, the tangent directions at the curve ends are given by the first and last parts of the control polygon, i.e. by $\mathbf{d}_1 - \mathbf{d}_0$ and $\mathbf{d}_{L+n-1} - \mathbf{d}_{L+n-2}$, respectively. In this way, the behavior of the curve at the ends can be much better controlled. If the end knots have lower multiplicity, the first and last control points do not lie on the curve. In particular, if all knots are simple and the curve is uniformly parameterized, i.e. $u_i - u_{i-1} = c$ for $i = 1, \ldots, L+2n-2$, every polynomial segment of the curve, including the first and last segments, can be represented using the same set of $n+1$ basis functions. Then they have the following property:

$$N_i^n(u) = N_{i-k}^n(u - kc), \quad k \in \mathbb{N}, \quad k \leq i$$

Each segment has $n+1$ control points, which partly overlap with the control points of neighboring segments, and which do not coincide with the end points of the segment.

Remark Unfortunately, the representations of splines is not consistent comparing different standard literature, particularly with regard to [Fari97] and [PT97]. Taking into account data formats such as IGES, in [PT97] and in many other publications, two additional knots "u_{-1}"and "u_{L+2n-1}"are used at the beginning and at the end, increasing the end knot multiplicities by one. As pointed out by Farin, these knots have no influence on any computation. Therefore they are omitted in this work.

Cubic B-spline curves In practice, piecewise cubic curves are most often used. They combine appropriate stiffness with sufficing flexibility. At the joints, cubic B-spline curves exhibit at most C^2-continuity.

Probably one of the most popular curve schemes is the non-uniform C^2 cubic interpolatory spline. It is a powerful tool for the task of passing a curve through a given set of points, and it is implemented in the modeler described in chapter 4 for drawing curves.

Uniform cubic splines have the advantage that each cubic segment can be represented using the same set of basis functions. Let an inner segment of such a curve be defined over a local parameter $t \in [0, 1]$ by

$$\mathbf{s}(t) = \sum_{i=0}^{3} \mathbf{d}_i N_i^3(t).$$

The uniform knot vector from which the basis functions are constructed is then given by $U_s = \{-2, -1, 0, 1, 2, 3\}$, i.e. $u_0 = -2$, $u_1 = -1$, $u_2 = 0$, $u_3 = 1$, u_4=2, and $u_5 = 3$. Applying eq. (3.4) yields the basis functions

$$\begin{aligned} 6N_0^3(t) &= 1 - 3t + 3t^2 - t^3, \\ 6N_1^3(t) &= 4 - 6t^2 + 3t^3, \\ 6N_2^3(t) &= 1 + 3t + 3t^2 - 3t^3, \\ 6N_3^3(t) &= t^3. \end{aligned} \tag{3.5}$$

Note that, in case all knots are simple, every segment of the curve can be written in this form, since the subset of the knot vector from which their basis functions are constructed, can be transformed to U_s. However, if the first and last knots of the curve have multiplicity 3, which is usually the case, different basis functions

for the first and last two segments will be obtained. Let particularly consider a cubical segment defined on $U_s = \{0, 0, 0, 1, 1, 1\}$. In this case, the B-spline basis functions are the cubic Bernstein basis functions (3.2), and hence the segment is a cubic Bézier curve.

The first derivative of a uniform cubic B-spline curve can be simplified to

$$\mathbf{d}'(u) = \sum_{i=1}^{L+2} (\mathbf{d}_i - \mathbf{d}_{i-1}) N_i^2(u)$$

Tensor product surfaces

The concept of tensor product surfaces can be illustrated by the following intuitive description [Fari97]: *A surface is the locus of a curve that is moving in space and thereby changing its shape.* Consider that the moving and deforming curve is a B-spline curve $\mathbf{d}(u)$. Then each individual control point $\mathbf{d}_i$ of the curve traverses another curve, i.e. $\mathbf{d}_i = \mathbf{d}_i(v)$. Of course, all the curves $\mathbf{d}_i(v)$, let them be B-splines as well, can have different shapes, since the initial curve is deforming while being swept out. Suppose the $\mathbf{d}_i(v)$ have control points $\mathbf{d}_{ij}$, then combining the two curve equations yields a tensor product B-spline surface as

$$\mathbf{d}(u, v) = \sum_{i=0}^{L+n-1} \sum_{j=0}^{K+m-1} \mathbf{d}_{ij} N_i^n(u) M_j^m(v)$$

that is a composite surface with $(L+n) \cdot (K+m)$ control points, consisting of $L \cdot K$ polynomial patches. It is assumed that one knot sequence in the u-direction,

$$U = \{u_0, \ldots, u_{L+2n-2}\},$$

and one knot sequence in the v-direction,

$$V = \{v_0, \ldots, v_{K+2m-2}\},$$

are given. The surface is therefore defined over the rectangular domain

$$[u_{n-1}, u_{L+n-1}] \times [v_{m-1}, v_{K+m-1}].$$

U and V determine the basis functions of the surface that are products of the univariate B-spline basis functions $N_i^n(u)$ and $M_j^m(v)$ with degrees n and m, respectively, which are given in the form of eq. (3.4).

Most of the properties and algorithms listed for curves apply to tensor product surfaces as a straightforward extension. They have automatic continuity, are invariant under affine transformations, and change their shape locally if a control point is moved.

Surface boundary and knots multiplicities Similar to curves, end knots having the full multiplicity in U and in V mean that the control points $\mathbf{d}_{ij}$ for which i resp. j equals 0, 1, $L+n-2$, $L+n-1$, $K+m-2$, or $K+m-1$, are also control points of the piecewise Bézier net of the surface. Hence, they determine the boundary curves and the cross boundary derivatives. In case all knots are simple and both U and V are uniform, each polynomial patch of the surface, including the boundary patches, can be represented using the same set of $(n+1) \cdot (m+1)$ basis functions.

Piecewise bicubic B-spline surfaces Again, piecewise bicubic surfaces are of most practical relevance. Lets further assume that both knot vectors are uniform and that all knots are simple. Each bicubic patch can then be written in the form

$$\mathbf{b}(s,t) = \sum_{i=0}^{3}\sum_{j=0}^{3} \mathbf{p}_{ij} N_i^3(s) N_j^3(t), \tag{3.6}$$

defined over a local parameter $(s,t) \in [0,1] \times [0,1]$. The $N_i^3(s)$, $N_j^3(t)$ are the univariate uniform cubic B-spline basis functions and are given by eq. (3.5). However, in case the end knots have full multiplicity 3, which is mostly the case, a different set of basis functions occurs for all patches lying in a boundary stripe that is two patches wide.

Another way to break down a surface into bicubics would be to insert each knot that appears in U or V until it has multiplicity 3. This would result in a control net of the surface that is the piecewise Bézier net. Each patch could then be written as

$$\mathbf{b}(s,t) = \sum_{i=0}^{3}\sum_{j=0}^{3} \mathbf{b}_{ij} B_i^3(s) B_j^3(t),$$

where $B_i^3(s)$ and $B_j^3(t)$ are the Bernstein polynomials according to eq. (3.2).

Notice that the control points $\mathbf{b}_{ij}$ of the Bézier patches are generated from the original control net $\{\mathbf{p}_{ij}\}$ by the knot insertion algorithm. This means that there is no longer any automatic continuity after inserting the knots. Moving control points $\mathbf{b}_{ij}$ only affects a single Bézier patch, since patches have been isolated from their neighboring patches by the process of breaking down the surface. In contrary, representing a bicubic patch in the form of eq. (3.6) just means selecting the corresponding subset of control points and basis functions of the surface. The support of the basis functions covers neighboring patches as well, thus the automatic continuity is retained naturally.

Sketching with tensor product surfaces

A wealth of sketching techniques is available for spline-based tensor product surfaces. For example, when a surface of revolution is created, the user selects an axis of revolution, and indicates the profile to form the surface. Sweeps may be used to create objects by moving a profile along a path. Eventually, the profile is being altered during the sweep, corresponding to the definition of a tensor product surface.

However, building more complex models out of several such surfaces from scratch imposes restrictions regarding the freedom and the spontaneity of user input. A tensor-product spline surface is defined on four-sided domains, which severely restricts its use. Furthermore, continuity among neighboring surfaces depends on the compatibility of the boundary curves. A sketching system should not involve the designer in ensuring these conditions; this would be a serious hindering factor for creative shape input.

Opposed to the task of shape creation, to which restrictions for sketching complete models apply, spline surfaces are very suitable for interactive shaping, based on the following properties:

- Fast triangulation and evaluation algorithms, based on explicit parameterization, allow the sampling of the surface at different levels of detail, which is particularly important when the shape is being deformed.
- An easy transfer into standard CAD packages is possible, which is important for subsequent stages of the design process.

– Efficient algorithms based on variational modeling are available that support fair, pleasant shaping, and allow interactive deformations of the surface.

3.2.4 Subdivision surfaces

The basic idea of subdivision is to refine an initial polygonal mesh by applying subdivision rules to the vertices of the mesh, generating a new mesh that has a greater number of faces and vertices. By repetitively refining the mesh, the sequence of meshes converges to a smooth limit surface, referred to as a *subdivision surface*. Vertices in the finer level are generated by taking a weighted average of nearby vertices in the coarser level. The subdivision rules specify a mask or stencil of vertices and assign weights to them, which is typically described by corresponding pictures. In case the same subdivision rules are valid for each step of refinement, the scheme is called *stationary*. The rules can be encoded in a subdivision matrix that maps a set of neighboring vertices into vertices of the finer mesh. The most important requirements to subdivision rules are:

– A smooth limit surface should be obtained.

– A point on the finer level should be generated using only a small number of nearby points on the coarser level.

– A vertex should influence only a small region of the limit surface.

– The rules should be affinely invariant.

A variety of subdivision schemes is available. Their refinement rules are based on two different principles, according to [Zori99], *vertex insertion* and *corner cutting*. Rules based on vertex insertion generate new vertices, called *odd* vertices, by splitting edges, or by creating vertices within faces, and connecting the new vertices. Old vertices that are retained possibly change their position. They are called *even* vertices. Corner cutting rules generate new faces inside of old faces, connect the new faces, and discard the old vertices. Examples are the *Doo-Sabin* [DS78] and the *Midedge* [PR97] schemes. Vertex insertion schemes can be further classified based on two criteria, the mesh type (triangular or quadrilateral), and whether the limit surface interpolates or approximates the original vertices of the mesh. The table from [Zori99] shows this classification for most known stationary subdivision schemes.

	Triangular	Quadrilateral
Approximating	*Loop*	*Catmull-Clark*
Interpolating	*Modified Butterfly*	*Kobbelt*

One of the greatest advantages of subdivision schemes is that they easily support surfaces of *arbitrary topology* type, which means that the edges and vertices of the associated mesh can be connected so that vertices have an arbitrary number N of incident edges, called the *valency* of a vertex. Therefore, surfaces that have an arbitrary topological genus can be created. Compared to this, classic spline-based surfaces cannot be generated by control meshes that contain *extraordinary vertices*. In case of quadrilateral meshes, a vertex is called an extraordinary vertex, if it has a valency not equal to four, the other vertices are called *regular* vertices. For triangular meshes, vertices with a valency equal to six are regular, and vertices of other valencies are extraordinary.

Subdivision schemes for meshes of arbitrary topology are often related to spline-based surfaces defined on their regular counterpart.

The Catmull-Clark scheme

Such a subdivision approach is the Catmull-Clark scheme. It extends the subdivision rules that generate uniform bicubic B-spline surfaces to meshes of arbitrary topology. The rules of the Catmull-Clark scheme are given by [CC78]:

- For each face of the mesh, generate new *face points* by averaging all old points defining the face.
- Generate new *edge points* by averaging the midpoints of the old edge with the two new face points of the faces adjacent to the edge.
- Calculate the new *vertex points* as

$$\frac{1}{n}\mathbf{q} + \frac{2}{n}\mathbf{r} + \frac{n-3}{n}\mathbf{s},$$

  where

  q is the average of the new face points of all faces adjacent to the old vertex point,

  r is the average of the midpoints of all old edges incident on the old vertex point,

  s is the old vertex point, and where

  n is the valency of **s**.

The mesh is reconnected by the following method.

- Each new face point is connected to the new edge points of the edges defining the old face.
- Each new vertex point is connected to the new edge points of all old edges incident on the old vertex point.
- The new faces are those that are enclosed by new edges.

Given an initial mesh of arbitrary topology, after the first refinement all faces will be four-sided, and the number of extraordinary vertices will remain constant. Furthermore, after subdividing at least twice, each face contains at most one extraordinary vertex.

Away from extraordinary vertices, the Catmull-Clark scheme is equivalent to midpoint uniform B-spline knot insertion. For all faces of the mesh that contain no extraordinary vertex, the following is true: The 16 vertices surrounding such a face are the control points of a uniform bicubic B-spline patch that can be written in the form given by eq. (3.6). By further subdividing, the portion of the surface composed of bicubic patches grows, whereas the region consisting of patches that contain an extraordinary vertex shrinks. Therefore, according to the bicubic spline, the limit surface is C^2 everywhere except at extraordinary vertices, where it is C^1.

In case all faces of the mesh are quadrilateral, which can be achieved by subdividing it at least once, the rules of the Catmull-Clark scheme can also be expressed as masks shown in Figure 3.2.

The boundary Note that the Catmull-Clark rules were originally given only for closed meshes, since there were no specific rules for the boundary and for the corners of a mesh. To specify the behavior of the boundary of a mesh more clearly, in the first instance the subdivision rules could be modified in the following way:

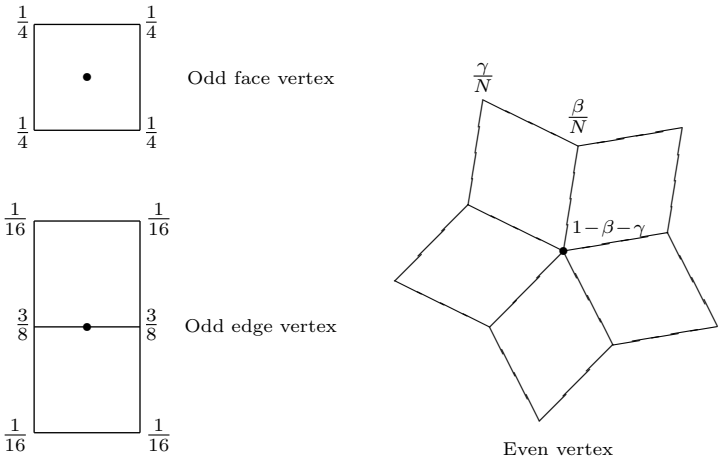

Figure 3.2: The masks of the Catmull-Clark scheme. Applicable to interior vertices of meshes with quadrilateral faces. $\beta = \frac{3}{2N}$ and $\gamma = \frac{1}{4N}$, where N is the valency of the vertex.

Figure 3.3: Boundary masks of the Catmull-Clark scheme

- Apply the rule for generating new edge points only to those edges of the old mesh that are not part of the boundary.
- Apply the rule for generating new vertex points only to those vertices of the old mesh that are not part of the boundary.

For regular meshes, i.e. meshes that contain only four-sided faces and contain no extraordinary vertex[1], this results in a uniform bicubic B-spline surface [Joy]. Consequently, the four boundary curves of that surface are uniform cubic B-spline curves.

In order to obtain such curves for more general meshes, it will be sufficient if the boundary of the mesh consists of a stripe that has a width of three regular faces. Each 4×4 vertices of that stripe are the control points of a boundary patch of the surface. A mesh yielding such a surface is shown in Figure 3.4.

In this manner, it is possible to obtain standard uniform B-spline curves as boundary curves for Catmull-Clark surfaces, or to prescribe such curves as a boundary interpolated by a Catmull-Clark surface. This method is used in the modeler described in the next chapter. Is is important to note that the boundary vertices of that kind of meshes are not the control points of the boundary curve of the surface.

The mesh can be prevented from shrinking towards the interior during refinement, by adding additional boundary rules illustrated by the masks in Figure 3.3. In this way, the boundary of the mesh is better approximated by the boundary of the limit surface. However, the resulting surface is formally not C^1 continuous [Zori99].

Since recently, so-called *combined subdivision schemes* allow a better control of the boundary. These schemes combine the known subdivision rules with modified rules for the boundary, so that arbitrary parametric boundary curves are interpolated by the surface. Such a combined scheme was presented by Levin [Levi99a, Levi99b] as a variant of the Catmull-Clark scheme. A drawback is that the possibility to treat

[1]Boundary vertices will be considered regular if their valency is equal to 3, and corner vertices will be regular, if their valency is 2.

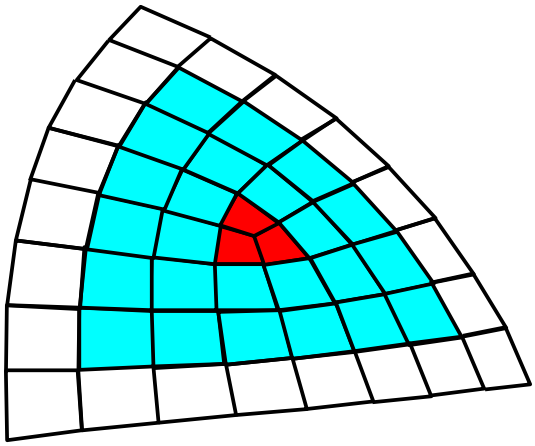

Figure 3.4: A control mesh resulting in a three-sided Catmull-Clark surface, containing one extraordinary vertex in the middle that is shared by the red faces. The 16 vertices surrounding a blue face define a bicubic B-spline patch. The boundary of those patches forms the surface boundary. Note that the faces of the control mesh, and not the patches are shown.

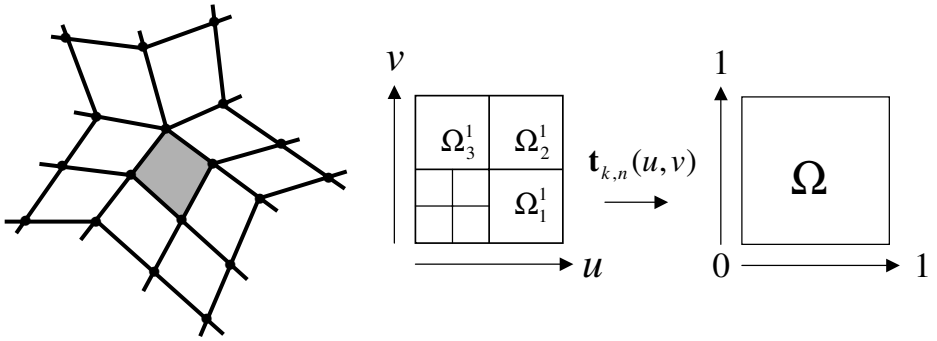

Figure 3.5: The shaded patch can be evaluated directly at arbitrary (u, v) using Stam's algorithm. (u, v) falls into a tile Ω_k^n of the patch domain. The $\mathbf{t}_{k,n}(u, v)$ map Ω_k^n into the unit square.

regions of the surface corresponding to regular parts of the mesh as bicubic B-spline patches is lost, as a result of the modified rules.

Exact evaluation of Catmull-Clark surfaces

Since recently, Catmull-Clark subdivision surfaces had to be evaluated by explicitly subdividing. Fortunately, Stam [Stam98] removed this restriction and presented a non-iterative method that allows to exactly evaluate Catmull-Clark surfaces at arbitrary parameter values.

In the Catmull-Clark subdivision scheme, the 16 vertices surrounding faces that do not contain extraordinary vertices are the control points of a uniform bicubic B-spline patch that can be evaluated directly. All faces that contain extraordinary vertices cannot be evaluated as uniform B-splines. By subdividing the initial control mesh at least twice, extraordinary vertices are isolated and each face is a quadrilateral, containing at most one extraordinary vertex with a valency N. Such a face, illustrated in Figure 3.5, can be evaluated directly using Stam's algorithm.

The key idea behind Stam's algorithm is to compute the eigenstructure, i.e. the

eigenvectors and eigenvalues, of the subdivision matrix of the Catmull-Clark scheme. Let $\bar{\mathbf{V}}$ denote the matrix whose columns are the corresponding eigenvectors. Note that $\bar{\mathbf{V}}$ is invertible [Stam98]. The shape of the patches is determined by their $K = 2N + 8$ control points, collected in the matrix $\mathbf{D}^T = (\mathbf{d}_1, \ldots, \mathbf{d}_K)$.

The patches can be evaluated directly in terms of their control points transformed by

$$\hat{\mathbf{D}} = \bar{\mathbf{V}}^{-1}\mathbf{D}. \tag{3.7}$$

Let $\mathbf{p}_i^T$ denote the rows of $\hat{\mathbf{D}}$. Then the patch can be evaluated as

$$s(u, v) = \sum_{i=1}^{K} \mathbf{p}_i B_i(u, v), \tag{3.8}$$

defined over the unit square $\Omega = [0, 1] \times [0, 1]$. Let the u, v-axes be oriented according to Figure 3.5. The K basis functions $B_i(u, v)$ are defined by their restrictions to the tiles Ω_k^n that arise from subdividing the face.

$$B_i(u, v)\big|_{\Omega_k^n} = (\lambda_i)^{n-1} x_i(\mathbf{t}_{k,n}(u, v), k) \tag{3.9}$$

λ_i denotes the i^{th} eigenvalue of the eigenstructure. The $x_i(u, v)$ are real-valued bicubic splines whose 16 coefficients only depend on the valency of the irregular vertex.

The $\mathbf{t}_{k,n}(u, v)$ map the tile Ω_k^n of the patch domain into the unit square (see Figure 3.5) and are given by:

$$\begin{aligned} \mathbf{t}_{1,n}(u, v) &= (2^n u - 1, 2^n v), \\ \mathbf{t}_{2,n}(u, v) &= (2^n u - 1, 2^n v - 1), \\ \mathbf{t}_{3,n}(u, v) &= (2^n u, 2^n v - 1). \end{aligned} \tag{3.10}$$

The tedious task of computing the eigenstructure of the subdivision matrix only has to be performed once. Stam has precomputed these eigenstructures, including the coefficients of the splines $x_i(u, v)$, up to some maximum valency of $N = 50$ and has made them available in [Stam99b].

Provided that these structures have been read from a file, a patch around an extraordinary vertex is evaluated by performing the following steps:

1. Project the control points surrounding the patch into the eigenspace, using eq. (3.7). Note that this step is only needed whenever the patch is evaluated the first time or after an update of the mesh.
2. Determine the tile into which the parameter (u, v) falls, obtaining k and n.
3. Map (u, v), lying in Ω_k^n, onto the unit square, using the transformation $\mathbf{t}_{k,n}(u, v)$.
4. Evaluate the splines $x_i(\mathbf{t}_{k,n}(u, v), k)$, using the given control coefficients from [Stam99b]; and compute the surface point.

Sketching with subdivision surfaces

A property that makes subdivsion surfaces very attractive for sketching objects from scratch is that they support arbitrary topology meshes. This will free the designer from managing constraints between neighboring patches, if the sketching method supports describing the model as a single connected surface. The initial mesh of a subdivision surface may be sketched using various methods, similar to polygonal surfaces.

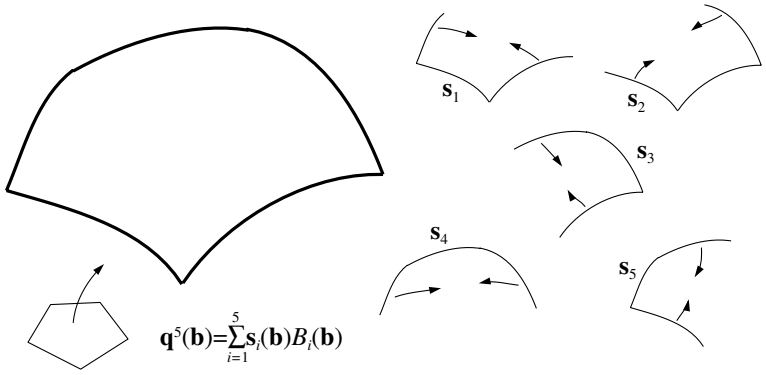

Figure 3.6: A surface of Kuriyama results from blending subsurfaces formed by sweeping crosssection curves across a guide curve

The elaboration of shape based on subdivision surfaces benefits from two aspects. By locally refining the mesh, shape details can elegantly be integrated into the representation. For subdivision schemes based on spline surfaces, such as Catmull-Clark surfaces and Loop surfaces, deformation methods based on variational modeling are available that may be applied to larger parts of the surface, or may be used to fair the surface. These methods work efficiently, and therefore are very suitable for use in an interactive modeler.

Furthermore, the performance of rendering methods benefits from the relationship to splines as well. This relationship has led to the development of the before mentioned non-iterative evaluation algorithms for Catmull-Clark and Loop surfaces [Stam98, Stam99a]. This allows to keep the complexity of the set of triangles used for rendering independent from the model complexity, which is useful for achieving an interactive frame rate in virtual environments.

3.2.5 Surfaces based on curve networks

The surfaces of Kuriyama

A task-based approach to represent a surface has been developed by Kuriyama [Kuri94] in order to more adequately meet the needs of designers. The intention of his surface approach is to support an intuitive and user-friendly method for the quick input of surfaces at the early stages of the design process. His approach generates surfaces from a network of curves that have arbitrary parametric forms. The network can have arbitrary topology such as t-connected or multiple intersections. The surface generated from a network consists of multisided patches defined on a multivariate coordinate system. Even two-sided and open-sided patches, i.e. a domain that lacks a closing curve, are supported.

The method of Kuriyama will be summarized here, since it seems indeed to be an adequate solution to the problem of quickly sketching surfaces. Therefore, the surfaces of Kuriyama have been implemented in the modeling system that is described in the subsequent chapters.

Suppose that within a curve network, a closed loop of m curves $\mathbf{c}_i(t)$ surrounds a patch, where $m \geq 2$ and $i = 1, \ldots, m$. Let the parameter interval of the curve $\mathbf{c}_i(t)$ that corresponds to the part of the loop be $[0, \Delta_i]$.

An m-sided patch $\mathbf{q}^m(\mathbf{b})$ (see Figure 3.6) can then be written as a combination

of m subsurfaces $\mathbf{s}_i(\mathbf{b})$, using an n^{th}-order blending function $B_i^n(\mathbf{b})$, as follows:

$$\mathbf{q}^m(\mathbf{b}) = \sum_{i=1}^{m} \mathbf{s}_i(\mathbf{b}) B_i^n(\mathbf{b}), \tag{3.11}$$

which is defined on an m-lateral domain polygon P, except $m = 2$, where a 4-sided domain polygon is used.

Every point $\mathbf{p} \in P$ is mapped to the vector $\mathbf{b} = (b_1(\mathbf{p}), \ldots, b_m(\mathbf{p}))$ of generalized barycentric coordinates by the functions

$$b_i(\mathbf{p}) = \frac{\pi_i(\mathbf{p})}{\sum_{i=1}^{m} \pi_i(\mathbf{p})}$$

with

$$\pi_i(\mathbf{p}) = \frac{\prod_{i=1}^{m} \alpha_i(\mathbf{p})}{\alpha_{i-1}(\mathbf{p})\alpha_i(\mathbf{p})},$$

where $\alpha_i(\mathbf{p})$ denotes the signed area of the triangle $\mathbf{p}\mathbf{p}_i\mathbf{p}_{i+1}$, whose sign is defined to be positive if $\mathbf{p}$ lies inside the domain P.

The blending function $B_i^n(\mathbf{p})$ is defined by

$$B_i^n(\mathbf{b}) = \frac{(b_i b_{i+1})^n}{\sum_{k=1}^{m} (b_k b_{k+1})^n}$$

The degree n can be chosen by the designer. This function has singular points at the vertices of P, which can be removed by adopting the following values at those points:

$$B_i^n(\mathbf{b}) = \begin{cases} 0 & \text{if} \quad b_{j \neq i,i+1} = 1 \\ 1/2 & \text{if} \quad b_{j=i,i+1} = 1 \end{cases}$$

The subsurfaces $\mathbf{s}_i(\mathbf{b})$ are constructed from the triple of neighboring curves $\mathbf{c}_{i-1}(t)$, $\mathbf{c}_i(t)$, and $\mathbf{c}_{i+1}(t)$, by sweeping the *crosssection curves* $\mathbf{c}_{i-1}(t)$ and $\mathbf{c}_{i+1}(t)$ along the *guide curve* $\mathbf{c}_i(t)$. It is assumed for the following that $\mathbf{c}_i(0) = \mathbf{c}_{i-1}(0)$ and $\mathbf{c}_i(\Delta_i) = \mathbf{c}_{i+1}(0)$.[2] The subsurfaces $\mathbf{s}_i(\mathbf{b})$ are defined by introducing local parameters (u_i, v_i) with respect to the edge of the domain corresponding to $\mathbf{c}_i$ by

$$\mathbf{s}_i(u_i(\mathbf{b}), v_i(\mathbf{b})) = \hat{\mathbf{c}}_{i-1}(v_i)\mathbf{M}_{i,i-1}(u_i) + \hat{\mathbf{c}}_{i+1}(v_i)\mathbf{M}_{i,i+1}(u_i) + \mathbf{c}_i(u_i), \tag{3.12}$$

with

$$\hat{\mathbf{c}}_j(v_i) = \mathbf{c}_j(v_i) - \mathbf{c}_j(0), \quad j = i-1, i+1.$$

The matrices $\mathbf{M}_{i,j} \in \mathbb{R}^{3\times 3}$ represent a general sweep of the crosssection curves along the guide curve, combined with a blending function. In case of a translational sweep, the definition of a subsurface can be simplified to

$$\mathbf{s}_i(u_i(\mathbf{b}), v_i(\mathbf{b})) = \Big(1 - \frac{u_i}{\Delta_i}\Big)\hat{\mathbf{c}}_{i-1}(v_i) + \frac{u_i}{\Delta_i}\hat{\mathbf{c}}_{i+1}(v_i) + \mathbf{c}_i(u_i),$$

which is taken here as the definition of $\mathbf{s}(u_i, v_i)$ for the sake of simplicity.

The local coordinate functions $u_i(\mathbf{b})$ and $v_i(\mathbf{b})$ are given by

$$u_i = \begin{cases} \Delta_i \sum_{k=1}^{m/2} b_{i+k} & m \text{ even} \\ \Delta_i \sum_{k=1}^{(m-1)/2} b_{i+k}/(1 - b_{i+(m+1)/2}) & m \text{ odd} \end{cases}$$

$$v_i = \left(\Delta_{i-1}\left(1 - \frac{u_i}{\Delta_i}\right) + \Delta_{i+1}\frac{u_i}{\Delta_i}\right) \sum_{k=2}^{m-1} b_{i+k}$$

Note that the operators $+$ and $-$ on the subscripts of b are defined cyclically so that the subscripts always are in $\{1, \ldots, m\}$.

[2] $\mathbf{s}_1(\mathbf{b})$ and $s_m(\mathbf{b})$ are constructed from $\mathbf{c}_m(t)$, $\mathbf{c}_1(t)$, $\mathbf{c}_2(t)$, or from $\mathbf{c}_{m-1}(t)$, $\mathbf{c}_m(t)$, $\mathbf{c}_1(t)$, respectively, assuming a corresponding coincidence of the curve ends.

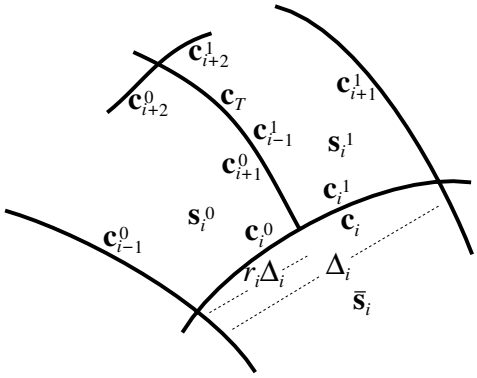

Figure 3.7: A t-connected intersection, after [Kuri94]

Modifications for singular topology The definition of eq. (3.12) preserves the condition of geometric continuity of a subsurface $\mathbf{s}_i$ with an adjacent subsurface $\bar{\mathbf{s}}_i$, according to [Kuri94]. However, a network of sketched curves tends to contain types of curve intersections so that the crosssection curves shared by $\mathbf{s}_i$ and $\bar{\mathbf{s}}_i$ are not successively parameterized, which destroys the geometric continuity of adjacent subsurfaces.

Therefore, modified methods are proposed by Kuriyama for conditions of *singular topology*, such as *t-connected intersections* (a curve ends in the middle of another curve) and for *multiple intersections* (more than two boundary curves intersect at a common vertex).

T-connected intersections Let the curve end of $\mathbf{c}_T$ be connected to the middle of $\mathbf{c}_i$, splitting $\mathbf{s}_i$ into two subsurfaces $\mathbf{s}_i^0$ and $\mathbf{s}_i^1$, as shown in Figure 3.7. These subsurfaces cannot have geometric continuity with $\bar{\mathbf{s}}_i$ if they are constructed by the standard method.

The following subsurfaces are generated, so that the final surface obtained by blending has geometric continuity with $\bar{\mathbf{s}}_i$:

1. Create $\mathbf{s}_i$ as if $\mathbf{c}_T$ did not exist.

2. Split $\mathbf{s}_i$ into two subsurfaces at $u_i = r_i\Delta_i$:

$$\mathbf{s}_i^0(u_i, v_i) := \mathbf{s}_i(r_i u_i, v_i)$$
$$\mathbf{s}_i^1(u_i, v_i) := \mathbf{s}_i((1 - r_i)u_i + r_i\Delta_i, v_i)$$

3. Use $\mathbf{c}_T$ as a guide curve in constructing $\mathbf{s}^0_{i+1}$ and $\mathbf{s}^1_{i-1}$, and as a crosssection curve for $\mathbf{s}^0_{i+2}$ and $\mathbf{s}^1_{i+2}$.

Multiple intersections Consider a guide curve $\mathbf{c}_i$ that has a multiple intersection with two crosssection curves $\mathbf{c}^1_{i-1}$ and $\mathbf{c}^2_{i-1}$ that are independently parameterized, see Figure 3.8. In order to obtain geometric continuity for the two adjacent subsurfaces $\mathbf{s}_i$ and $\bar{\mathbf{s}}_i$ across $\mathbf{c}_i$, $\mathbf{c}^1_{i-1}$ and $\mathbf{c}^2_{i-1}$ are replaced with a common curve $\hat{\mathbf{c}}_{i-1}$ that is continuous at the intersection and is given by

$$\hat{\mathbf{c}}_{i-1}(t) = \frac{\sum_{k=1}^{\lambda} \mathbf{c}^k_{i-1}(t)}{\lambda},$$

where λ denotes the number of curves $\mathbf{c}^k_{i-1}$ that meet at the multiple intersection.

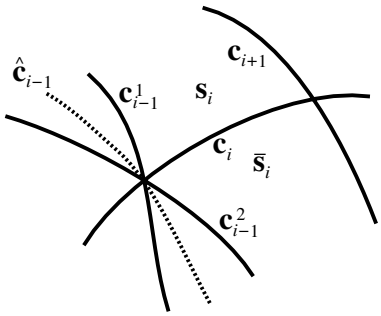

Figure 3.8: A multiple intersection, after [Kuri94]

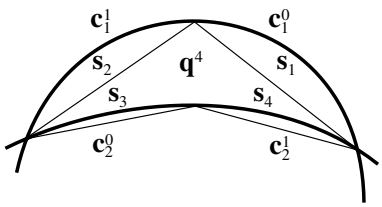

Figure 3.9: A 2-sided patch defined on a 4-sided domain, after [Kuri94]

This modification of crosssection curves used to construct $\mathbf{s}_i$ and $\overline{\mathbf{s}}_i$ ensures that they have geometric continuity across $\mathbf{c}_i$ of the same order as that of $\hat{\mathbf{c}}_{i-1}$. However, note that the final surface does not have geometric continuity at the multiple intersection.

2-sided patch Let two boundary curves $\mathbf{c}_1$ and $\mathbf{c}_2$ form a 2-sided loop, as shown in Figure 3.9. From the definition of the barycentric coordinates for the domain polygon it can be seen that a 2-sided domain is not supported. Therefore, a quadrilateral domain is constructed by splitting $\mathbf{c}_1$ and $\mathbf{c}_2$ in the middle, obtaining four boundary curves:

$$\mathbf{c}^0_j(t) := \mathbf{c}_j\left(\frac{t}{2}\right), \qquad \mathbf{c}^1_j(t) := \mathbf{c}_j\left(\frac{\Delta_j + t}{2}\right), \qquad j = 1, 2.$$

Four subsurfaces $\mathbf{s}_i$, $i = 1, 2, 3, 4$, are constructed using the split curves, according to the following table, where $\mathbf{c}_{left}$ denotes the left crosssection curve, $\mathbf{c}_{guide}$ the guide curve, and $\mathbf{c}_{right}$ the right crosssection curve.

i	$\mathbf{c}_{left}$	$\mathbf{c}_{guide}$	$\mathbf{c}_{right}$
1	$\mathbf{c}^1_2$	$\mathbf{c}^0_1$	$\mathbf{c}^0_2$
2	$\mathbf{c}^1_2$	$\mathbf{c}^1_1$	$\mathbf{c}^0_2$
3	$\mathbf{c}^1_1$	$\mathbf{c}^0_2$	$\mathbf{c}^0_1$
4	$\mathbf{c}^1_1$	$\mathbf{c}^1_2$	$\mathbf{c}^0_1$

The blending function B^n_i used to generate the final surface $\mathbf{q}^4$ is also constructed on a quadrilateral domain.

Sketching curve networks

As curve network is actually a sketching method for the quick input of shape. What makes curve networks particularly attractive for use in the initial design stages is that complete models may be designed from scratch, contrary to the common methods of sketching B-spline surfaces. The curve network corresponds naturally to the method of describing a model by drawing its principal curves, or contour curves. From a designer's point of view, this is a very intuitive way to specify a shape.

An ideal method of constructing a surface from a curve network would not impose restrictions on the surface domains, on the type of connections in the curve network, or on the parameterization of the boundary curves.

The surfaces of Kuriyama fulfill these requirements, which makes them a suitable choice for the task of shape *creation* from scratch. However, *elaboration* of free-form models naturally is based on sculpting surfaces, which is not directly supported by the surfaces of Kuriyama. On the other hand, it is always possible to deform the curves and update the adjacent surfaces, which is achievable at interactive frame rates. This has been demonstrated successfully by our modeler that uses the surfaces of Kuriyama.

Combining curve networks with deformable surfaces would increase the usefulness of curve networks as a means for quick shape input considerably. Since curve networks are likely to contain non-four-sided domains and arbitrary connections (see section 3.2.5), using a subdivision surface approach could be a solution. An interpolating scheme would have the advantage that the curve network and the Kuriyama surfaces could be reasonably well sampled. The sampled triangle mesh could also be used in a discrete fairing approach. Fitting in Catmull-Clark surfaces would allow to take advantage of efficient evaluation and rendering methods, as well as to make use of efficient deformation and fairing algorithms. For that reason, Catmull-Clark surfaces have been chosen for the modeler that is described in the next chapter. How they can be integrated into the curve network will be shown, closely approximating the drawn curves and, if desired, the surfaces of Kuriyama.

Chapter 4

A Modeler for Spline-based Conceptual Free-Form Styling

In the next 3 chapters, a modeling system for conceptual free-form styling in a virtual environment is described, which has been implemented at the Responsive Workbench. The modeler consists of components for drawing curves (section 4.3), connecting curves and editing curve networks (section 4.4), and creating the surfaces (section 4.5). Curve and surface deformation techniques for use in virtual environments, which are integrated in the modeler, are introduced in chapter 5.

These parts describe the pure functionality of the modeler. If the system were implemented on a desktop system, the design of the user interface would be based on familiar concepts, so that the description of the modeler could end with chapter 5. However, the interaction style in a workbench-like configuration is completely different from a desktop environment. The user interface to the functionality provided with the modeler, including 3D interaction and system control techniques, is therefore introduced in chapter 6.

4.1 Aim

The goal pursued by the described modeler is to enable the design of complete conceptual models from scratch. The quality level of the sketched models should be higher than that obtained from just stitching together several spontaneous drawing strokes, forming a cluttered bunch of polygons. The surface quality should be characterized by pleasant shapes, surface smoothness, and continuous surface transitions. The models should already resemble realistic objects, and should be recognizable as the result of a creative, but also constructive, process, rather than looking like abstract, artistic attempts. Despite that, they should be creatable within a reasonable amount of time.

Moreover, the model should fulfill certain geometric constraints, such as symmetry or planarity, where appropriate. Another desirable property would be that the connectivity information is an integral part of the model representation. For example, when modeling an airplane, the wings should be connected to the fuselage not only geometrically by moving parts together, but also in the topological structure of the model. This would ease the task of further elaboration considerably.

The surface sketching functionality should provide both enough flexibility and exactness sufficient for the design of such models.

4.2 Approach

The basic assumption underlying the design of the modeler is that surface sketching in a 3D environment for models like those described in section 4.1 is a rather difficult task, which cannot be performed by all users in a convincing manner, and moreover, requires suitable tools assisting even experienced users in sculpting a shape. The inherent difficulty in surface sketching stems from the fact that surfaces exhibit a large number of degrees of freedom, which cannot all be defined simultaneously and satisfactorily if a high-quality surface is desired.

The approach of the described modeler is to reduce the complexity of the task by subdividing it into more feasible tasks, combining their outputs automatically into a reasonable result to start with, and provide intuitive tools to further elaborate it. The resulting surface sketching process that has to be performed by a user of the system consists of the following stages:

1. Consider how the model can be described as a network of principal curves, or contour curves.
2. Draw the contour curves of the model, outlining the shape of its surface parts.
3. Connect the curves to define the topology.
4. Let the system generate a surface that exhibits a shape according to the shape of the surrounding curves.
5. Deform the curves and surfaces to work out the final shape of the model.

The tools related to 2. and 3. are described in the following sections, whereas the curve and surface deformation tools (5.) as well as methods to compute reasonable initial surface shapes (4.) deserve special attention and are described in chapter 5.

4.3 Drawing curves

A curve can be drawn with the modeler by performing a direct drawing stroke with the hand, holding a stylus, corresponding to the intended curve shape.

A path consisting of a 3D point set that is formed by moving the stylus through space is tracked and recorded. A new point is appended to the path each time the distance to the last path point exceeds a minimum distance and the elapsed time when that last point has been stored exceeds a minimum time frame. If the new point is too close to the last point or the time frame has not been reached, the new point will be temporarily appended to the path, overwriting the last position in order to avoid a jerky drawing process.

The minimum distance between two path points, and also the minimum time can be adjusted by the user in such a way that the resulting piecewise linear point path samples the drawn curve appropriately. Generally, increasing the speed of the drawing gesture performed by the hand means that the minimum time should be reduced in order to get a sufficient number of samples. Increasing the minimum time is used to avoid generating too many data points. The minimum distance can be set to enhance or relax the exactness at which the path is sampled.

However, the electromagnetically tracked data points are not recorded as raw data. Before adding a point to the path, it undergoes a filtering step that uses a Gaussian filter kernel positioned in such a way that its maximum coincides with the last tracked point. The parameters of the filter can be adjusted by the user. In this way, jittering caused by the tracking system (see section 1.2.3) can be substantially reduced.

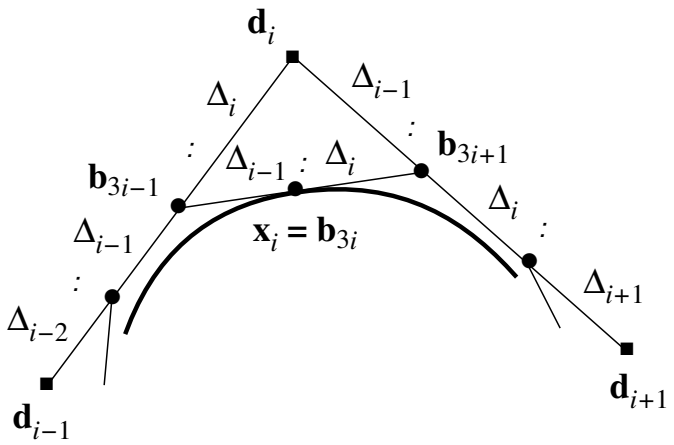

Figure 4.1: The A-Frame: relation between the data points $\mathbf{x}$, Bézier points $\mathbf{b}$, and B-spline points $\mathbf{d}$

In each application frame (see section 1.2.4 for a description of the stages of the software rendering pipeline), a new cubic B-spline curve is interpolated from all points in the point path, using standard cubic spline interpolation as described in the following paragraph. Since this method globally interpolates over all data points, the curve shape is changed globally as well. However, previously drawn segments of the curve remain visually stable, as soon as a short distance to the end of the curve has been reached.

Cubic spline interpolation Suppose at a certain application frame the point path contains the tracked points $\mathbf{x}_0, \ldots, \mathbf{x}_n$. A cubic B-spline curve is generated from these data points by cubic spline interpolation, which is described in [Fari97] and will be repeated here. Cubic B-spline curves have been presented in section 3.2.3. Let a Cubic B-spline curve that is created from the $n+1$ data points be written as

$$\mathbf{d}(u) = \sum_{i=-1}^{n+1} \mathbf{d}_i N_i^3(u), \tag{4.1}$$

where the cubic basis functions $N_i^3(u)$ are defined on the knot vector

$$U = \{u_{-2}, \ldots, u_{n+2}\}.$$

Note that the end knots are assumed to have multiplicity 3, i.e. $u_{-2} = u_{-1} = u_0$ and $u_n = u_{n+1} = u_{n+2}$.

Two tasks now must be solved. Firstly, a parameterization U has to be found, since it is in general not given. Secondly, $\mathbf{d}(u)$ shall interpolate the data points, so that $\mathbf{d}(u_i) = \mathbf{x}_i$; $i = 0, \ldots, n$.

The approach to solve the interpolation problem is based on the observation that every B-spline curve can be written as a piecewise Bézier curve, so that $\mathbf{x}_i = \mathbf{b}_{3i}$, $i = 0, \ldots, n$, where $\mathbf{b}_{3i}$ is the junction point of two cubic Bézier segments, see Figure 4.1. From analyzing the relationship between the Bézier control points, the data

points, and the B-spline control points, a linear system of the form

$$\begin{pmatrix} 1 & & & & & & & \\ \alpha_1 & \beta_1 & \gamma_1 & & & & & \\ & & \ddots & & & & & \\ & & & & \alpha_{n-1} & \beta_{n-1} & \gamma_{n-1} & \\ & & & & & & 1 \end{pmatrix} \begin{pmatrix} \mathbf{d}_0 \\ \mathbf{d}_1 \\ \vdots \\ \mathbf{d}_{n-1} \\ \mathbf{d}_n \end{pmatrix} = \begin{pmatrix} \mathbf{y}_0 \\ \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_{n-1} \\ \mathbf{y}_n \end{pmatrix}$$

is obtained, where the $\mathbf{y}_i$ are related to the data points by

$$\begin{aligned} \mathbf{y}_0 &= \mathbf{b}_1, \\ \mathbf{y}_i &= (\Delta_{i-1} + \Delta_i)\mathbf{x}_i, \qquad i = 1, \ldots, n-1, \\ \mathbf{y}_n &= \mathbf{b}_{3n-1}, \end{aligned}$$

using the forward difference operator $\Delta_i = u_{i+1} - u_i$. The matrix elements are given by

$$\begin{aligned} \alpha_i &= \frac{(\Delta_i)^2}{\Delta_{i-2} + \Delta_{i-1} + \Delta_i}, \\ \beta_i &= \frac{\Delta_i(\Delta_{i-2} + \Delta_{i-1})}{\Delta_{i-2} + \Delta_{i-1} + \Delta_i} + \frac{\Delta_{i-1}(\Delta_i + \Delta_{i+1})}{\Delta_{i-1} + \Delta_i + \Delta_{i+1}}, \\ \gamma_i &= \frac{(\Delta_{i-1})^2}{\Delta_{i-1} + \Delta_i + \Delta_{i+1}}, \end{aligned}$$

$i = 1, \ldots, n-1$, where $\Delta_{-1} = \Delta_n = 0$.

The first and last control points coincide with the first and last data points, i.e.

$$\mathbf{d}_{-1} = \mathbf{x}_0, \qquad \mathbf{d}_{n+1} = \mathbf{x}_n.$$

Notice that there are two free parameters left, namely the two Bézier points determining the tangent vectors at the ends of the curve, $\mathbf{y}_0 = \mathbf{b}_1$ and $\mathbf{y}_n = \mathbf{b}_{3n-1}$. There are several methods that generate these end conditions. One appropriate choice is the *Bessel end condition* that sets the tangent vector $\mathbf{b}_1 - \mathbf{b}_0$ equal to the tangent vector of the interpolating parabola through the first three data points, so that

$$3\mathbf{b}_1 = \left(1 + \frac{\Delta_1}{\Delta_0 + \Delta_1}\right)\mathbf{x}_0 + \frac{\Delta_0 + \Delta_1}{\Delta_1}\mathbf{x}_1 - \frac{\Delta_0^2}{\Delta_1(\Delta_0 + \Delta_1)}\mathbf{x}_2.$$

Similarly,

$$3\mathbf{b}_{3n-1} = \left(1 + \frac{\Delta_{n-1}}{\Delta_{n-2} + \Delta_{n-1}}\right)\mathbf{x}_{n-2} + \frac{\Delta_{n-2} + \Delta_{n-1}}{\Delta_{n-1}}\mathbf{x}_{n-1} - \frac{\Delta_{n-2}^2}{\Delta_{n-1}(\Delta_{n-2} + \Delta_{n-1})}\mathbf{x}_n.$$

For an equidistant parameterization, that is, $\Delta_i = c \in \mathbb{R}$ for all $i = 0, \ldots, n-1$, the linear system simplifies to

$$\begin{pmatrix} 1 & & & & & & & \\ \frac{3}{2} & \frac{7}{2} & 1 & & & & & \\ & 1 & 4 & 1 & & & & \\ & & & \ddots & & & & \\ & & & & 1 & 4 & 1 & \\ & & & & & 1 & \frac{7}{2} & \frac{3}{2} \\ & & & & & & & 1 \end{pmatrix} \begin{pmatrix} \mathbf{d}_0 \\ \mathbf{d}_1 \\ \mathbf{d}_2 \\ \vdots \\ \mathbf{d}_{n-2} \\ \mathbf{d}_{n-1} \\ \mathbf{d}_n \end{pmatrix} = \begin{pmatrix} \mathbf{b}_1 \\ 6\mathbf{x}_1 \\ 6\mathbf{x}_2 \\ \vdots \\ 6\mathbf{x}_{n-2} \\ 6\mathbf{x}_{n-1} \\ \mathbf{b}_{3n-1} \end{pmatrix}$$

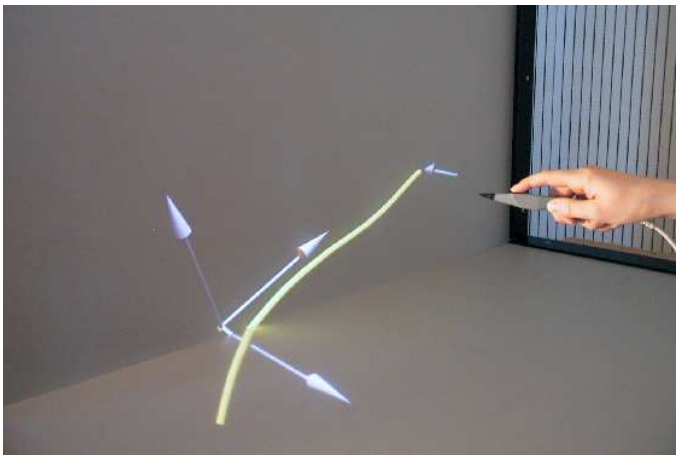

Figure 4.2: Drawing a curve

However, setting simply $u_i = i$ does not take into account the spacing of the data points, and it can generate very poor shapes, such as overshoots. Many solutions have been proposed to finding an appropriate parameterization; one choice that is simple and sufficient in practice is the *centripetal parameterization*, given by

$$\frac{\Delta_i}{\Delta_{i+1}} = \sqrt{\frac{\|\Delta \mathbf{x}_i\|}{\|\Delta \mathbf{x}_{i+1}\|}}, \tag{4.2}$$

where the operator Δ is defined for points in the same way as for knots. Note that the knot vector is not uniquely defined by this equation, since affine parameter transformations are allowed.

Notation For convenience, in the remaining parts of this chapter, let the indices of the control points of eq. (4.1) start at 0, and let a cubic B-spline curve with m control points be written as

$$\mathbf{d}(u) = \sum_{i=0}^{m-1} \mathbf{d}_i N_i^3(u), \tag{4.3}$$

where the indices of the knots and the basis functions change accordingly.

4.3.1 Drawing in space

The purpose of drawing curves directly in 3D is to outline the shape of a model by specifying its boundary. Drawing arbitrary space curves that vary in all 3 dimensions simultaneously is rarely required, since most industrial shape designs look rather regularly shaped.

Nevertheless, it is evident that the modeler requires from the designer a certain level of drawing and artistic skills – as required also when using a piece of paper and a pencil, or other traditional methods. It is assumed here that an experienced designer would be able to sweep out virtual space curves, applying his drawing skills by using large-scale gestures in 3D, although this assumption is currently not sufficiently substantiated. When drawing a space curve at the Responsive Workbench, the extra third dimension poses the difficulty that a curve might exhibit unwanted variations of shape when turned towards another direction. From the observations of designers using immersive modeling tools [DBW+00] it can however

be reported that drawing a curve in space is fairly achievable, and produces the expected results. Some designers who tested the system described here at the Responsive Workbench were even able to draw and connect space curve networks without having used the system before.

4.3.2 Drawing on virtual planes

Modeling and assembling with curves is further alleviated by the fact that not all contours of an object need to be drawn as space curves. In many cases, planar curve drawing or subdividing a surface by projecting a curve onto it, which is deformed afterwards, is more appropriate.

To draw a planar curve, the tracked data points are projected onto a virtual plane, and the curve is interpolated through the data afterwards. A drawing plane, for which the user can specify the position and orientation, is displayed in the modeling space.

4.3.3 Drawing curves on the surface

A curve that is being three-dimensionally drawn can be constrained to lie on the surface of the model by interpolating a cubic B-spline curve trough data points projected onto the surface. Therefore, the curve can follow arbitrary directions on the surface, approximating a true surface curve.

Point projection is implemented using Newton iteration. Let $\mathbf{s}$ denote a surface and (u, v) a parameter value on the rectangular domain of the surface. Newton iteration can be used to minimize the distance of a data point $\mathbf{x}$ and a surface point $\mathbf{s}(u, v)$ by iteratively approximating a solution to the equations

$$\begin{aligned}\mathbf{r}(u, v) &= \mathbf{s}(u, v) - \mathbf{x},\\ f(u, v) &= \mathbf{r}(u, v)^T \mathbf{s}_u(u, v) = 0,\\ g(u, v) &= \mathbf{r}(u, v)^T \mathbf{s}_v(u, v) = 0.\end{aligned}$$

Newton iteration exhibits a strong local convergence, provided that a good start value (u_0, v_0) is given. The algorithm, provided by [PT97], is given on page 51.

The two tolerances ϵ_1 and ϵ_2 can be used to indicate convergence of the Newton iteration. ϵ_1 is a measure of distance, and ϵ_2 is a measure to indicate a zero cosine between $\mathbf{r}(u_i, v_i)$ and the two derivative vectors at the surface point.

The algorithm generates a path of surface points only within the surface part that the start value (u_0, v_0) belongs to, since the whole surface of the model is not uniformly parameterized. In order to support surface curves that cover several surface parts of the model, a new start value has to be provided for the surface part adjacent to the boundary crossed by the point path.

To find a start value, a heuristic method is used. The whole surface of the model is sampled, and the distances of the surface points to $\mathbf{x}$ are computed. A surface part and the start value (u_0, v_0) is chosen that yield the closest distance to $\mathbf{x}$.

It is important to find a good start value in order to achieve reliable convergence, which is not guaranteed by the above algorithm. In case the maximum number of iterations is exceeded, no point is added to the point path on the surface. Instead, a new tracked data point is taken and the algorithm is started again with a new start value.

4.3.4 Rendering curves

The B-spline curve is what is visible to the user as the drawn curve. OpenGL provides efficient drawing routines for NURBS curves that are implemented on the

$i := 0$

while i has not exceeded the maximum number of iterations

Evaluate $\mathbf{s}(u_i, v_i)$, $\mathbf{s}_u(u_i, v_i)$, $\mathbf{s}_v(u_i, v_i)$, $\mathbf{s}_{uu}(u_i, v_i)$, $\mathbf{s}_{vv}(u_i, v_i)$, $\mathbf{s}_{uv}(u_i, v_i)$.

$\mathbf{r}(u_i, v_i) := \mathbf{s}(u_i, v_i) - \mathbf{x}$

if points are coincident, that is $|\mathbf{r}| \leq \epsilon_1$

return $\mathbf{s}(u_i, v_i)$ as a surface point

endif

if the cosines are close to zero, that is $\frac{|\mathbf{s}_u^T \mathbf{r}|}{|\mathbf{s}_u| \cdot |\mathbf{r}|} \leq \epsilon_2$ and $\frac{|\mathbf{s}_v^T \mathbf{r}|}{|\mathbf{s}_v| \cdot |\mathbf{r}|} \leq \epsilon_2$

return $\mathbf{s}(u_i, v_i)$ as a surface point

endif

No convergence criterium is fulfilled; compute the parameter for the next iteration, using

$f_u := \mathbf{s}_u^T \mathbf{s}_u + \mathbf{r}^T \mathbf{s}_{uu}$

$f_v := \mathbf{s}_u^T \mathbf{s}_v + \mathbf{r}^T \mathbf{s}_{uv}$

$g_u := f_v$ as long as $\mathbf{s}_{uv} = \mathbf{s}_{vu}$

$g_v := \mathbf{s}_v^T \mathbf{s}_v + \mathbf{r}^T \mathbf{s}_{vv}$

if $f_u g_v - g_u f_v = 0$

halt the iteration without generating a surface point

endif

Generate new parameters, namely

$u_{i+1} := u_i + \frac{f_v g - g_v f}{f_u g_v - g_u f_v}$

$v_{i+1} := v_i + \frac{g_u f - f_u g}{f_u g_v - g_u f_v}$

if (u_{i+1}, v_{i+1}) is out of the domain of the surface part

halt the iteration without generating a surface point

endif

if the parameters do not change significantly, i.e.
$|(u_{i+1} - u_i)\,\mathbf{s}_u + (v_{i+1} - v_i)\mathbf{s}_v| \leq \epsilon_1$

return $\mathbf{s}(u_{i+1}, v_{i+1})$ as a surface point

endif

$i := i + 1$

endwhile

Table 4.1: Algorithm for point projection using Newton iteration [PT97]

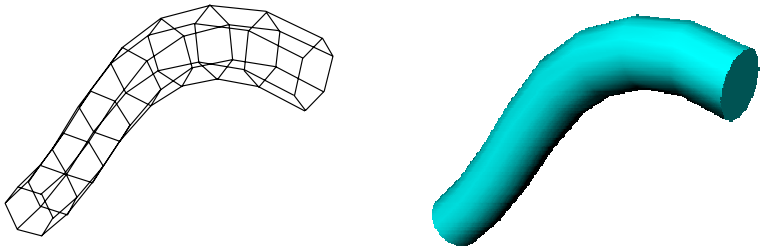

Figure 4.3: A curve is represented as a smooth shaded tube of meshed limbs. The faces of the limbs are shown as quadrilaterals.

geometry engines of the graphics board. However, a curve is represented just as a thin pixel curve by these algorithms. In order to interact with a curve directly in a virtual environment, e.g. to support selection and grabbing, a curve representation that enables a better spatial perception would be much more appropriate. Therefore, the modeler implements the representation of splines as tubes that have a thickness defined by the user. Compared to a thin wire representation, tubes exhibit realistic shading effects. In particular, they seem to benefit the representation of curve networks used as a skeleton for the surface.

A tube consists of connected limbs surrounding the curve. Each limb is formed by a closed triangle mesh with a hexagonal cross section, approximating a cylinder, as shown in Figure 4.3. This results in 14 vertices and 14 vertex normals for each mesh, since the mesh represents 12 triangles. Vertices and normals are reused in successive meshes by using index lists. The vertex normals allow a smooth shading of the tube.

4.4 Connecting curves

4.4.1 Constructing a curve network

As described in section 4.2, the purpose of drawing curves in space, on a plane, or on the surface is to construct a curve network outlining a skeletal model. A new curve can be drawn freely, but it has to be connected to the existing curve network afterwards, thereby altering its shape slightly. This means that the resulting net curve is not identical to the curve initially drawn. However, this solution has turned out to be practical, since in a sketching environment, exactly reproducing user input is not what is primarily needed. Instead, a method has been developed that frees the user from directly specifying the intersection points with the network, at the same time leaving the whole freedom available when drawing a new curve. It just has to pass the regions where connections are supposed to be created at distances below a threshold value.

Connecting a drawn curve to the network

Let a cubic B-spline curve $\mathbf{c}(u)$, $u \in \mathbb{R}$ be given in the form of eq. (4.3), and let the newly drawn curve $\mathbf{c}_{draw}(u)$ be defined over the knot vector $U = \{u_{-2}, \ldots, u_{l+2}\}$, $l \in \mathbb{N}$, using triple end knots, i.e. $u_{-2} = u_{-1} = u_0$, and $u_l = u_{l+1} = u_{l+2}$.

On the net curves, which $\mathbf{c}_{draw}(u)$ approaches, a number m of points $\{\mathbf{q}_1, \ldots, \mathbf{q}_m\}$, each having a minimal distance to a corresponding point $\mathbf{r}_j$ on $\mathbf{c}_{draw}(u)$, $j \in \{1, \ldots, m\}$, are estimated, see Figure 4.4. The final curve $\mathbf{c}_{net}(u)$ is an approximation to $\mathbf{c}_{draw}(u)$ interpolating the network points $\{\mathbf{q}_1, \ldots, \mathbf{q}_m\}$. It is created

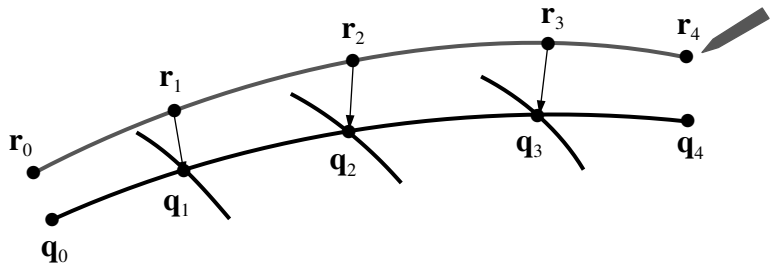

Figure 4.4: Connecting a drawn curve to the network. The connection points are $\mathbf{q}_1$, $\mathbf{q}_2$, and $\mathbf{q}_3$.

and inserted into the network automatically from the new curve after the drawing sweep has been completed by the following steps:

1. Compute the network connection points $\{\mathbf{q}_1, \ldots, \mathbf{q}_m\}$ and the associated point set $\{\mathbf{r}_1, \ldots, \mathbf{r}_m\}$ on $\mathbf{c}_{draw}(u)$.

2. Estimate end points $\mathbf{q}_0 = \mathbf{c}_{draw}(u_0) + \mathbf{q}_1 - \mathbf{r}_1$ and $\mathbf{q}_{m+1} = \mathbf{c}_{draw}(u_l) + \mathbf{q}_m - \mathbf{r}_m$ to be appended to the connection points, which correspond to the ends of $\mathbf{c}_{draw}(u)$. Set $\mathbf{r}_0 = \mathbf{c}_{draw}(u_0)$ and $\mathbf{r}_{m+1} = \mathbf{c}_{draw}(u_l)$.

3. Interpolate a curve $\mathbf{c}_q(t)$ through $\{\mathbf{q}_0, \ldots, \mathbf{q}_{m+1}\}$ over the knot vector $T = \{t_{-2}, \ldots, t_{m+1}\}$, where $t_{-2} = t_{-1} = t_0$, and $t_{m-1} = t_m = t_{m+1}$. Let T be created from the $\mathbf{q}_i$ by centripetal or chord length parameterization, see eq. (4.2).

4. Interpolate a curve $\mathbf{c}_r(t)$ through $\{\mathbf{r}_0, \ldots, \mathbf{r}_{m+1}\}$, using the same knot vector.

5. Affinely map T to the domain of U, i.e. so that $t_{-2} = u_{-2}$, and $t_{m+1} = u_{l+2}$. Merge the knot vectors T and U, mutually inserting missing knots.

6. $\mathbf{c}_{net}(u) = \mathbf{c}_{draw}(u) + \mathbf{c}_q(u) - \mathbf{c}_r(u)$.

7. Insert $\mathbf{c}_{net}(u)$ into the curve network, and discard $\mathbf{c}_{draw}(u)$.

As an alternative, a variational method that minimizes the overall shape deviation from the original curve could be used by defining a corresponding curve energy functional. To force the target curve to interpolate the network points $\{\mathbf{q}_1, \ldots, \mathbf{q}_m\}$, interpolation constraints must be included into the system. See chapter 5 for more information on variational modeling.

A Strategy for creating curve networks

When determining the network connection points and the associated points on the drawn curve, it must be taken into account that curve networks are constructed to describe the skeleton of models. For the construction of reasonably connected networks, it seems to be useful to constrain the way new curves are connected to the network by distinguishing between the following types of connection points, which are illustrated in Figure 4.5, and assigning them a priority.

1. The end of the drawn curve and the end of another curve (create new network connection)

2. The end of the drawn curve and an existing network connection (use existing connection)

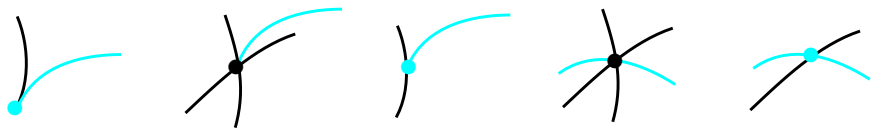

Figure 4.5: The order of connection types used to search the network (from left to right). The existing parts of the network, consisting of curves and connections, is shown in black, whereas the drawn curve and new connections are colored.

3. The end of the drawn curve and the inner range of another curve (create new network connection)
4. The inner range of the drawn curve and an existing network connection (use existing connection)
5. The inner range of the drawn curve and the inner range of another curve (create new network connection)

A connection of the new curve to the network will be created only, if a user-defined minimal distance to previously created connections on that curve is maintained. The distance is measured on the knot vector of the curve to allow loops. By searching for connections in an order corresponding to the priorities, the following connection strategy is automatically employed in the modeler when inserting a new curve.

- Create a model boundary by connecting open curve ends to the network and by connecting curve ends with each other.
- Avoid small loops and short segments by using existing connection points.

When a new connection point is created, the parameter values of the involved curves corresponding to that point are inserted as knots into the knot vectors of those curves. Hence, a connection point shared by a curve is also a junction point between two cubic segments of that curve.

4.4.2 Editing network curves

Deforming a curve already connected to the network means that either the influence range of the deformation must be restricted to the curve piece between two connections or end points or that the network has to adapt to the deformed shape in order to maintain the connectivity. For a greater flexibility, it seems to make sense to support both deformation modes, since deformations covering a larger range of a curve are likely to occur.

In the latter case, which is illustrated in Figure 4.6, all curves that intersect the deformed curve range must be shifted towards their new connection points. The range of recursive adaptation on the adapting curves ends at the neighboring connection points with the net, if they exist, or otherwise at the curve end points. These points remain unchanged in case that they are not itself connected with the deformed curve range.

A deformation tool should ensure geometric continuity of the deformed curve piece with neighboring curve pieces that remain unchanged. This can be achieved by fixing the first three control points of the first cubic curve segment at the beginning of the deforming curve piece, and the last three control points of the last segment.

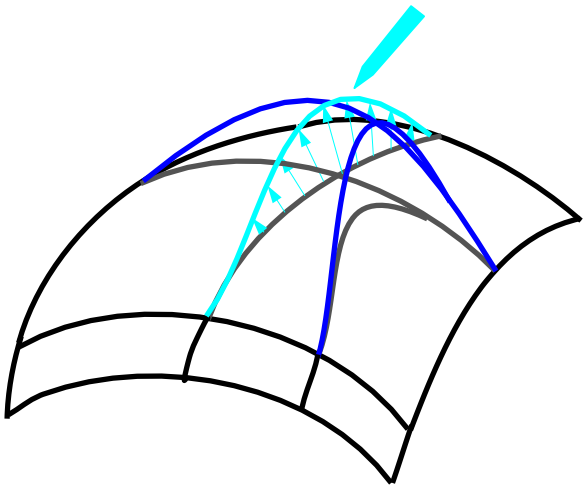

Figure 4.6: Deforming a network curve. The part being modified covers two curve pieces and is shown in light color. The adapting parts of the network are shown in dark color. The unchanged parts are black, and the previous state of the curves is shown in grey. The range of adaptation ends at the neighboring connection points or at the curve ends.

4.5 Creating the surface

Drawing and geometrically connecting a curve network is only a partial solution to the task of constructing a skeleton for a model surface. Fitting surface parts into a curve network generally cannot be solved uniquely. Additional information has to be derived from the user input to define the topology of the curve net, which specifies which curve pieces form the boundary for a part of the surface.

4.5.1 Extracting the topology

The curve network that is the result of drawing and connecting curves can be interpreted as an edge-vertex graph, where the words *edge*, *vertex*, and *graph* are used in the graph theoretical sense. An edge corresponds to a curve piece between two network connections or curve end points, whereas a vertex corresponds to a connection or curve end. Topology extraction means finding a unique set of cycles within the edge-vertex graph in order to assign faces to them. This has remained a difficult problem in the field of geometric modeling; whereas the other direction, deriving the edge-vertex graph from a collection of patches that form a surface, is straightforward. Methods have been developed to convert an edge-vertex graph to a face-based model, but these methods either do not guarantee that the found solution is unique, or they impose strong restrictions on the input graph.

Finding cycles in an edge-vertex graph

Some methods rely on a purely topological approach, which makes them generally applicable to several kinds of geometry. Hanrahan [Hanr82] constructs a unique embedding of an edge-vertex graph on a closed orientable surface, which is equivalent to determining the faces comprising the volume model represented by the graph. Unfortunately, his method will produce a unique embedding if and only if the graph is a planar graph that is triply connected, e.g. the graph shown in Figure 4.7. Being N-connected means that there exist n sequences of edges that do not share any vertices, between any two vertices. Either restriction, graph planarity

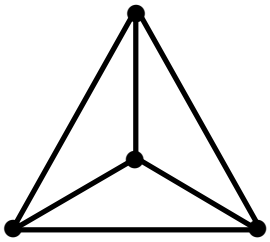

Figure 4.7: A 3-connected planar graph

(i.e. a possible embedding on a sphere), and the 3-connectedness seem to be much too strong to be imposed on a sketching system for curve networks.

For that reason, Elsas et al. [vEvdH95] present a method to extract the topology of an irregular network of sketched 3D curves, similar to the curve network described here. The modeler into which the method is included is a desktop system using a graphics tablet and a pen as input devices. Curves are sketched in a 3D environment; connections need not to be explicitly specified. Therefore, in the first step of their method, an edge-vertex graph is generated by geometrically identifying curves that are supposed to be connected. In the second step, they input the edge-vertex graph into a topological search algorithm that finds a set of minimal cycles, i.e., cycles that can't be split, within the graph.

Again, the main problem is that the set of cycles found by the algorithm is not necessarily unique. Instead, the result depends on the order in which edges emanating from a vertex are searched. Another problem is that a surface resulting from a minimal cycle might not be what was intended by the designer.

Topology extraction as a task for the designer

In summary, dropping the restrictions planarity and 3-connectedness means that a unique solution is not available. In this case a fully automatic topology extraction algorithm would possibly generate surfaces that won't correspond to the design intent. The applicability of such algorithms in the context of the described modeler therefore seems to be questionable. At least they should be complemented with geometric information and with additional user input in order to help resolving the ambiguities.

These ambiguities exist in the graph-theoretical sense, but they actually mean the specification of the design intent by the user. As a consequence, topology extraction in curve networks turns out to be a design problem, which can be most adequately solved by user input. The user somehow needs to specify the cycles with a pointing device, which can be achieved by selecting successive curve pieces in the simplest case, or by confirming or rejecting surface parts obtained from cycle candidates nominated by a search algorithm. In Figure 4.8 an example is shown.

Modeling topology in desktop systems

Currently available systems that rely on curve networks are mostly desktop systems with 2D input, such as the Fast Shape Designer by vanDijk [vD93, vD94], or desktop systems with spatial input and a stereo display such as 3-Draw by Sachs et al. [SRS91]. (The modeler from Sachs et al. does not support surfaces and therefore does not need a topology extraction module.)

Specifying cycles on a 2D desktop system seems to be obstructive because of the following reasons: the small scale at which the model is displayed means that individual elements such as curve pieces are located very close to each other so that

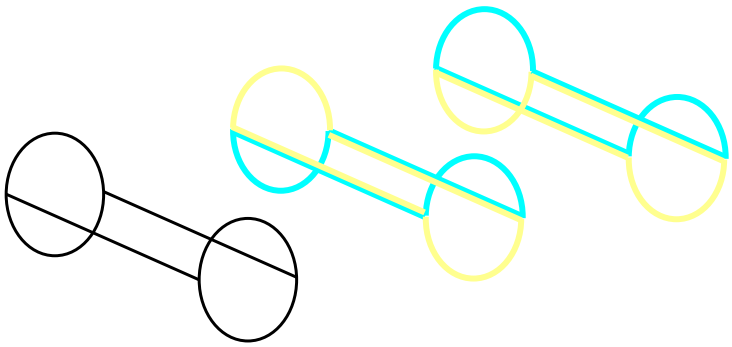

Figure 4.8: From the sketched curve network (left), a surface can be created in two different ways (middle and right). The surface consists of two parts, yellow and green, which are determined by their surrounding boundary pieces.

selecting cycles could be prone to error. Cycles that form back faces or occluded faces of the model can often not directly be localized by the pointing device in order to select the pieces of a cycle. The model has to be turned permanently to reach occluded cycles or lines. This task would benefit from a two-handed user interface, which would allow to distribute the task of turning the model and selecting parts onto both hands.

Defining the topology in such a way is therefore an issue of user interface design as well, and it is an argument for pursuing two-handed interaction also for desktop systems, in addition to the arguments given in the dissertation of Gribnau [Grib99]. See chapter 6 for two-handed interaction in virtual environments.

Modeling topology in virtual environments

Such inherently three-dimensional user input tasks are predestined for projection-based VEs, such as the Responsive Workbench, which display objects at a larger scale and therefore allow direct interaction with the hands. The topology extraction methods developed for our modeler rely on that principle. Loops[1] to fit in surfaces are found interactively by using the following simple algorithm, implemented by I. Nikitin [NW99]. In an initialization step, the (squared) distances of every curve piece in the network to the pointer are computed. Then, the following steps are performed.

1. Search for a curve piece that has a minimal distance to the pointer and store it as the first element in a sequence.

2. Select curve pieces connected to the last piece based on a minimal distance to the pointer and append them to the sequence until one of the following conditions, which terminate the algorithm, occurs:

   - the starting connection point has been reached (the sequence forms a cycle), or
   - the search has reached an open curve end (a cycle has not been found), or
   - the search has returned to a connection that is already included in the sequence (mismatched cycle).

The algorithm permanently searches the network for a cycle, giving feedback to the user by highlighting the curve pieces belonging to it, until the user validates the

[1] The terms *loop* and *cycle* are used here interchangeably.

found cycle by pressing a button. In this way, the user controls the process of cycle selection by moving the pointer interactively until the desired cycle has been found. In order to force the algorithm to find a minimal cycle, the user can point into the corresponding loop of curve pieces such that the pointer has a small distance to all of them. By observation, this method, although being very simple, has turned out to work very well in practice, because of its particular ease of use in a virtual environment.

In many cases however, it is not possible to position the pointing device such that the intended loop can indeed be selected, especially, when models become more complex. In these cases, the user can switch to the simplest possible method that is selecting successive curve pieces that form a loop with the pointer. This trivial method can easily be applied in a virtual environment, since direct spatial selection of curve pieces is possible and the size of the loops is appropriate, compared to the size of the pointing device or the hand. In section 6.3.2 it is explained how two-handed interaction at the Responsive Workbench benefits the task of topology extraction.

4.5.2 Fitting in surface parts

Each loop found by topology extraction is used independently from neighboring loops to fit in a *surface part*. In an interactive modeling session, the designer modifies the topology of a curve network frequently. After inserting new curves, moving curves or deleting curves, surface parts have to be subdivided or removed. Generating surface parts separately therefore is more efficient and more flexible compared to computing a whole surface out of a given network.

In the implemented modeler, the designer can choose between two different kinds of surfaces. The surfaces form Kuriyama [Kuri94], who also proposes curve networks, are one possible choice. His surfaces interpolate the boundary curves and have geometric continuity with neighboring surfaces. Since they are constructed from sweeps of curves along their neighboring guide curves (see section 3.2.5 for a summary of Kuriyama's approach), they reproduce the shape outlined by the surrounding curves, which is a very useful property. In addition, they can be defined over N-sided domains, $N \geq 2$. However, that kind of surface can only be deformed indirectly, since it does not possess a control point representation.

To take advantage of both properties, a surface shape that resembles the sketched shape of the curves, and the possibility of direct surface deformations the modeler supports Catmull-Clark surfaces that use the surfaces from Kuriyama as a reference surface (see section 5.3.4).

In the following table it is shown what kind of surface is fitted into a loop with a given number l of curve pieces, and how many sides N the surface domain has.

l	N	surface
2	4	Kuriyama
3	3	Catmull-Clark or Kuriyama
4	4	Bicubic spline or Kuriyama
5	5	Catmull-Clark or Kuriyama
> 5	$= l$	Kuriyama

Table 4.2: Fitting in N-sided surface parts

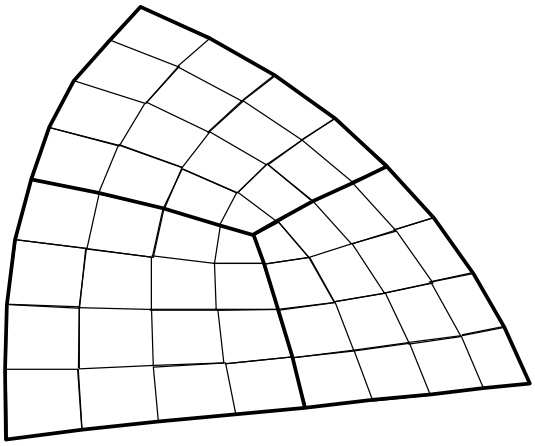

Figure 4.9: The mesh topology of a 3-sided Catmull-Clark surface part. Three 4-sided subdomains are defined. The boundary pieces generated by this mesh each consist of 6 cubic curve segments.

Fitting in Kuriyama surfaces

The curve pieces of a Kuriyama surface that is fitted into a loop directly define the surrounding curves for the surface, since the approach of Kuriyama works with arbitrary parametric curves (section 3.2.5).

Fitting in bicubic B-Spline surfaces or Catmull-Clark surfaces

Spline-based surfaces are fitted into a loop in case of 3, 4, or 5 boundaries[2]. The actual purpose of using the Catmull-Clark approach is to support non-4-sided domains in the curve network. The subdivision capabilities are not currently used, instead an initial control point mesh corresponding to a user-defined resolution is computed. The constructed topology of the control point mesh is a rectangular grid in case of 4 sides, obtaining a tensor-product bicubic B-spline surface part, and consists of N composite rectangular grids, obtaining a Catmull-Clark surface part, in other cases (see Figure 4.9). As a result, one irregular point with a valency of N arises in the middle of the mesh.

Meshes are interpreted as a collection of control points for uniform bicubic B-spline patches, each defined by 16 control points, except the N irregular faces sharing the middle vertex. The fact that uniform bicubic B-spline patches are reproduced is a characteristic of the original scheme by Catmull and Clark [CC78].

Each tensor product uniform bicubic patch has the representation

$$\mathbf{d}(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} \mathbf{d}_{ij} N_i(u) N_j(v), \tag{4.4}$$

$(u,v) \in [0,1] \times [0,1]$, with control points $\mathbf{d}_{ij}$. The 16 basis functions $N_i(u)N_j(v)$

[2]N-sided domains with $N > 5$ can be supported in the same way, but are currently not implemented

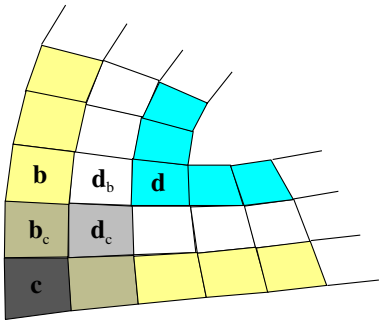

Figure 4.10: The patches of a surface part around the boundary. Note that the control mesh is not visible here.

are products of the univariate uniform B-spline basis,

$$
\begin{aligned}
N_0(t) &= \frac{1}{6} - \frac{1}{2}t + \frac{1}{2}t^2 - \frac{1}{6}t^3 \\
N_1(t) &= \frac{2}{3} - t^2 + \frac{1}{2}t^3 \\
N_2(t) &= \frac{1}{6} + \frac{1}{2}t + \frac{1}{2}t^2 - \frac{1}{2}t^3 \\
N_3(t) &= \frac{1}{6}t^3,
\end{aligned}
\tag{4.5}
$$

$u, v, t \in [0, 1]$, and $t = u$ or $t = v$. These univariate basis functions can be constructed using the recurrence from Mansfield, de Boor, and Cox [Fari97] together with a uniform knot vector with all knots having simple multiplicity.

The boundary The disadvantage of this representation is that the control points of the boundary of the mesh are not the control points of the boundary curve of the surface.

In order to control the boundary more easily, in the tensor product approach triple end knots are typically used [Fari97], which changes the basis functions of all boundary patches and of all patches adjacent to the boundary patches. Moreover, the first and last two rows and columns of control points now define the boundary curves and the cross boundary derivatives.

Interpreting a Catmull-Clark surface part as a collection of patches, it is possible to control the boundary of such a surface part in the same way. This results in six kinds of patches, not including patches around an extraordinary vertex, shown in Figure 4.10.

$$\mathbf{b}(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} \mathbf{b}_{ij} M_i(u) N_j(v), \tag{4.6}$$

$$\mathbf{c}(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} \mathbf{c}_{ij} M_i(u) M_j(v),$$

$$\mathbf{b}_c(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} \mathbf{b}_{c,ij} M_i(u) L_j(v),$$

$$\mathbf{d}_b(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} \mathbf{d}_{b,ij} L_i(u) N_j(v),$$

$$\mathbf{d}_c(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} \mathbf{d}_{c,ij} L_i(u) L_j(v),$$

$(u,v) \in [0,1] \times [0,1]$, with the basis functions $L_i(t), M_i(t), N_i(t), t = u$ or $t = v$, $N_i(t)$ given by eq. (4.5), and

$$\begin{aligned} L_0(t) &= \frac{1}{4} - \frac{3}{4}t + \frac{3}{4}t^2 - \frac{1}{4}t^3, \\ L_1(t) &= \frac{7}{12} + \frac{1}{4}t - \frac{5}{4}t^2 + \frac{7}{12}t^3, \\ L_2(t) &= N_2(t), \\ L_3(t) &= N_3(t), \end{aligned} \tag{4.7}$$

$$\begin{aligned} M_0(t) &= (1-t)^3, \\ M_1(t) &= 3t - \frac{9}{2}t^2 + \frac{7}{4}t^3, \\ M_2(t) &= \frac{3}{2}t^2 - \frac{11}{12}t^3, \\ M_3(t) &= N_3(t). \end{aligned} \tag{4.8}$$

4.5.3 Surface transitions

A common method to achieve a visually smooth transition between two surfaces that share a common boundary is to ensure that the C^r condition is satisfied. Two surfaces are said to be C^r if they are r times continuously differentiable across their common boundary curve, and if these derivatives up to order r agree there. In order to fulfill this condition, a basic requirement when using spline-based surfaces, like the Catmull-Clark surface parts chosen here, is that the boundary patches of adjacent surface parts must be compatible, regarding the degree, a common domain, and the number of patches adjacent to the boundary. In this case, the C^r condition can be expressed as a set of relations between the control points of the neighboring surface patches.

With regard to that, restrictions are imposed on the chosen mesh topology of a surface part fitted into a loop. The same number of cubic curve segments is needed for all boundary curve pieces of all surfaces parts, resulting in a uniform mesh topology for all surface parts of the model, which is shown in Figure 4.9 on page 59 for a 3-sided surface part.

Constructing a C^0 transition With that chosen configuration, the boundary control points of a surface part that is represented according to Figure 4.10 and eq.

(4.6) form a uniform cubic B-spline curve,

$$\mathbf{x}_b(t) = \sum_{k=0}^{n-1} \mathbf{p}_{k,b} N_{k,b}^3(t), \tag{4.9}$$

which must be shared by adjacent surface parts, in order to obtain a C^0 transition. Additionally, the network connection points must coincide with the end points of the boundary curve. Note that $\mathbf{x}_b(t)$ uses triple end knots.

The $\mathbf{x}_b(t)$ should approximate the curves originally constructed by the user as accurately as possible. Therefore, they are created from the originally drawn non-uniform curves using variational methods as described in section 5.2.6. In this section, it is assumed that they are available.

Constructing a C^1 transition A smooth transition between two surface parts can be constructed using the fact that across the common boundary, the first two control points of each row on a boundary patch control the cross boundary derivative, according to Farin [Fari97]. Let two surface parts of the form shown in Figure 4.10 be stitched together.

Consider two patches $\mathbf{b}^l(u,v)$ and $\mathbf{b}^r(u,v)$ that are defined over a common domain $[u_0,u_1]\times[v_0,v_1]$ and $[u_1,u_2]\times[v_0,v_1]$, respectively, with $u_0=0, u_1=1$, and $u_2=2$. Let further have $\mathbf{b}^l(u,v)$ and $\mathbf{b}^r(u,v)$ control nets $\{\mathbf{b}^l_{ij}\}$ and $\{\mathbf{b}^r_{ij}\}$, respectively, then the C^1 condition becomes

$$\frac{\partial}{\partial u}\mathbf{b}^l(u,v)\bigg|_{u=1} = \frac{\partial}{\partial u}\mathbf{b}^r(u,v)\bigg|_{u=1}. \tag{4.10}$$

This must be fulfilled for all patches that share an edge along the boundary, i.e. also for the other patch types $\mathbf{b}_c$ and $\mathbf{c}$ shown in Figure 4.10. According to eq. (4.6), eq. (4.10) can then be reduced to

$$\begin{aligned}
\mathbf{b}^l_{3j} - \mathbf{b}^l_{2j} &= \mathbf{b}^r_{1j} - \mathbf{b}^r_{0j},\\
\mathbf{b}^l_{c,3j} - \mathbf{b}^l_{c,2j} &= \mathbf{b}^r_{c,1j} - \mathbf{b}^r_{c,0j},\\
\mathbf{c}^l_{3j} - \mathbf{c}^l_{2j} &= \mathbf{c}^r_{1j} - \mathbf{c}^r_{0j},
\end{aligned}$$

$j=0,\ldots,3$. Additionally, the boundary control points of the "left" and the "right" side must coincide, and they should be equal to the control points $\mathbf{p}_{k,b}$ of the corresponding segment of the boundary curve $\mathbf{x}_b(t)$.

$$\begin{aligned}
\mathbf{b}^l_{3j} &= \mathbf{b}^r_{0j},\\
\mathbf{b}^l_{c,3j} &= \mathbf{b}^r_{c,0j},\\
\mathbf{c}^l_{3j} &= \mathbf{c}^r_{0j},
\end{aligned}$$

$j=0,\ldots,3$.

It is therefore easy to ensure a visibly continuous surface across a boundary by interpolating a cross boundary tangent vector from the two derivative vectors of opposite boundary curves, see Figure 4.11, provided that those curves are itself at least C^1 continuous.

There remains, however, one problem related to the interpolation of cross boundary derivatives. At the corner patch, two different control points $\mathbf{c}_{11}$ are generated by interpolating between the curve derivatives in each of the two directions. They correspond to an ambiguous mixed partial derivative $\frac{\partial^2}{\partial u \partial v}\mathbf{c}(0,0) \neq \frac{\partial^2}{\partial v \partial u}\mathbf{c}(0,0)$, also referred to as the twist vector, at the corner. Finding appropriate twist vectors has no straightforward solution [Fari97]. One possibility is simply to use zero twists,

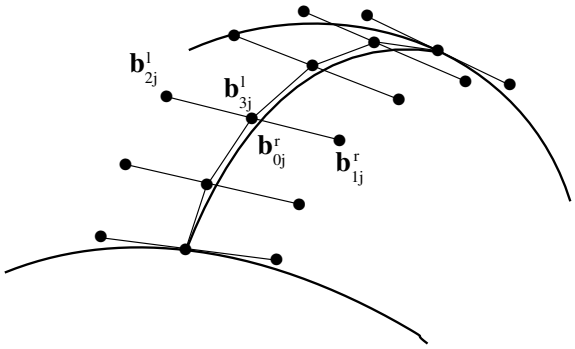

Figure 4.11: A C^1 continuous transition across the boundary of two surface parts. The shown control points must be collinear and must all be in the same ratio $1:1$.

which ensures a C^1 continuous surface for the configuration shown in Figure 4.11. The price to pay for is that "flat spots" can occur closely to the connection points, and moreover, that the interpolated cross boundary derivatives are not reproduced in the vicinity of the corners.

In case more than two curves meet at a connection point or the two curves are not C^1 continuous there, only a C^0 transition can be achieved at that point, or across the boundary, respectively.

A disadvantage with the solution to the boundary patch representation is that five different sets of basis functions must be maintained in addition to the basis functions of the remaining inner and extraordinary patches. For the purpose of variational modeling (see chapter 5), a separate implementation to compute the energy for each set of basis functions would be necessary. It would therefore be desirable to deal only with two kinds of patches, namely inner patches and extraordinary patches. This can be achieved by transforming the representations of eqs. (4.6) back to the patch representation for simple knots (4.4), after interpolating the cross boundary tangent vectors. The following simple calculations are used:

$$\begin{aligned}
\mathbf{d}^r_{0j} &= 6\mathbf{b}^r_{0j} - 6\mathbf{b}^r_{1j} + \mathbf{b}^r_{2j}, \\
\mathbf{d}^r_{1j} &= \frac{3}{2}\mathbf{b}^r_{1j} - \frac{1}{2}\mathbf{b}^r_{2j}, \\
\mathbf{d}^r_{2j} &= \mathbf{b}^r_{2j}, \\
\mathbf{d}^r_{3j} &= \mathbf{b}^r_{3j},
\end{aligned} \tag{4.11}$$

$j = 0, \ldots, 3$, and similarly for the "left" patch control points $\mathbf{d}_{ij,l}$. It can be shown that applying the calculations (4.11) to the representation (4.6) indeed yields the standard representation (4.4) for a boundary patch, i.e. $\mathbf{b}(u,v) = \mathbf{d}(u,v)$. The equations (4.11) can directly be derived from Barsky and Thomas [BT81], who present transformations among several spline formulations.

In order to transform a corner patch $\mathbf{c}(u,v)$, the equations (4.11) must be applied twice. The result of the first transformation for all rows of control points is taken as the input for the second step that is applied to the new columns. Due to the tensor product approach, applying (4.11) first to the columns and to the rows afterwards yields the same result.

4.5.4 Rendering surfaces

In section 3.1.1, the technical requirements of sketching in a virtual environment, which a surface approach should fulfill, were identified. It was argued that there

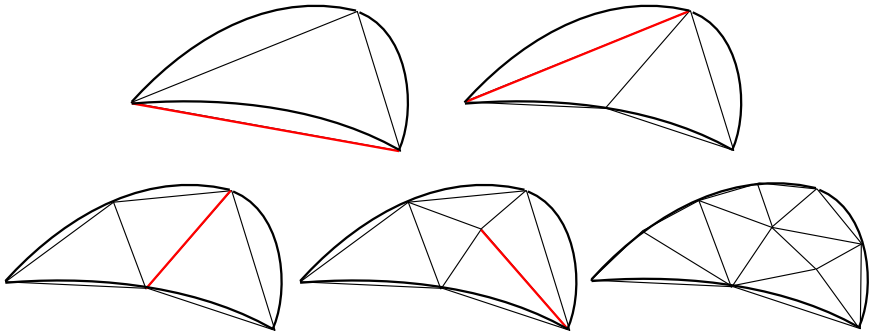

Figure 4.12: Triangulation of a surface using bisection of triangles. Four steps are shown. In each step, the edge that is subdivided next is shown in red. The last picture shows the triangulation after a few more steps. Note that bisecting an edge sometimes requires that the neighboring triangle is also bisected, in order to avoid cracks.

is a need for a rendering method that allows adaptation of the complexity of the tesselated surface representation to the load of the application and draw processes, in order to achieve interactive frame rates.

Thanks to the work of Stam [Stam98], a non-iterative evaluation algorithm for Catmull-Clark surfaces is available that is applicable to the surface parts used in the implemented modeler. This algorithm is described in section 3.2.4 on page 38. Using this evaluation procedure in combination with a flexible multilevel-triangulation method allows us to choose the quality of surface representation, i.e. the triangulation, to be completely independent from the subdivision level of a surface part.

The rendering algorithm is very simple. The goal is to obtain a set of triangles approximating the surface with reasonable accuracy. An error estimator decides whether a triangle will be refined using bisection on the side that deviates most from the surface, see Figure 4.12. The estimator evaluates the distance of the triangle sides to the surface, and, additionally, the distance of the middle of the triangle to the surface. By adjusting the tolerances, the quality of the surface representation can be controlled. During surface deformations, the error criterion is relaxed in order to keep the frame rate nearly constant. After the user has sculpted the surface, a single triangulation step based on a stricter criterion is performed.

Chapter 5

Curve and Surface Deformation Tools

This chapter introduces deformation tools for spline-based curves and surfaces, which have been implemented as components of the modeling system described collectively in chapter 4, 5, and chapter 6. Regarding the development of deformation tools, it was taken into account that users interact with their hands directly in space; usually not supported by force-feedback devices. Although the tools have been implemented in a workbench environment, they may also applied for interactive deformation tasks in other VE-based modeling systems as well.

The purpose of this chapter is to describe the tools in mathematical terms, and to show how the tools are applied in a virtual environment.

5.1 Introduction

This is achieved by high-level modeling tools, which means that these methods completely hide the mathematical representation from the designer, such as control points and knot vectors. This characteristic predestinates them for use in a virtual environment based sketching system.

5.1.1 High-level curve and surface modeling

Variational methods support creating fair, pleasant shapes by minimizing the energy of a curve or surface under given constraints. They have been introduced for surfaces by Welch and Witkin [WW92]. Wesselink and Veltcamp [VW95, WV95, Wess96] present variational modeling techniques and tools for curves and surfaces. Usually these methods require two steps. In the first step, positional or directional constraints, including boundary conditions are imposed on an object. In the second step, the remaining free shape parameters of the object are specified by energy minimization. This process is then repeated until the desired shape is found. The shape of the object can be altered by changing the constraints, or adding energy terms that have certain deformation effects.

The goal of this work is to use the variational modeling approach in a more interactive way. The idea is to use shaping tools that steadily adjust the influence of energy contributions. This depends on hand gestures using spatial input devices. Deformation methods should make use of the fact that the hands are interacting in space. Thereby, it makes sense to not restrict the view on physical metaphors, in order to unfold the possibilities offered by the ability of unconstrained hand movement in a VE. For curves, appropriate solutions are presented in section 5.2.

Another approach closely related to variational modeling is based on the simulation of *elastic deformations* when a force is applied to a curve or a surface. In case variational modeling is already available, linear elastic deformations for small forces can be implemented in a relatively easy way. The Hookean law holds in that case therefore, the energy matrix can be used as the stiffness matrix of the system. The full dynamic case is more complicated and involves more computation time. Work in this field is presented by Qin et al. [QMV98] for Catmull-Clark surfaces and by Terzopoulos [TQ94] for NURBS curves and surfaces.

5.1.2 Energy minimization

Variational modeling is a modeling approach based on minimizing a fairness functional defined on a curve or surface. The goal is to achieve a fair, pleasant shape. Although there is not an exact definition of what is meant by "fair", it can be derived from the literature that a fair shape should not exhibit inflection points, flat spots, buckles or bumps. The shape should be somehow visually pleasing, which can be achieved by choosing an appropriate energy functional that measures the fairness, as discussed e.g. by Greiner and Seidel [GLW96, GS97]. For curves, a combination of the bend and the stretch energy is an appropriate choice, considering the above mentioned fairness criterions. Similarly, for surfaces the so called *thin plate* or bend energy, combined with the stretch energy, is commonly used. However, the exact formulations of the bend or stretch energies are far too complicated for interactive use and are therefore replaced by approximations. For a discussion on the advantages and disadvantages of these widely used approximations, see e.g. [Wess96].

For spline-based curves and surfaces, i.e. curves and surfaces that can be written in the canonical representations

$$\mathbf{c}(t) = \sum_{i=0}^{n-1} \mathbf{d}_i N_i(t) \tag{5.1}$$

$$\mathbf{s}(u,v) = \sum_{i=0}^{n-1} \mathbf{d}_i M_i(u,v), \tag{5.2}$$

simple approximations to the bend and stretch energies are available that have the advantage of being quadratic in the control points. The resulting minimization system leads to a linear system that can be solved efficiently, which is an important factor for the use of this method in an interactive modeling system.

In eqs. (5.1) and (5.2), $\mathbf{c}(t)$ describes a curve with n control points $\mathbf{d}_i$ and basis functions $N_i(t)$ defined over a parameter t, whereas $\mathbf{s}(u,v)$ denotes a surface $\mathbf{s} : \Omega \to \mathbb{R}^3$ with n control points $\mathbf{d}_i$ and bivariate basis functions $M_i(u,v)$ defined on the rectangular domain $\Omega \subset \mathbb{R}^2$ and $(u,v) \in \Omega$.

In addition to the *internal* bend and stretch energy functionals responsible for achieving a fair shape, user-defined modeling operators result in *external* energy terms that are added to the overall curve or surface energy E. The modeling operators are used to achieve certain deformation effects, such as attracting a curve or surface towards a point, a line, or a plane, as introduced by Wesselink [Wess96]. They contribute energy terms that are quadratic or linear in the control points.

The quadratic parts of E can be assembled into an energy matrix $\mathbf{A} \in \mathbb{R}^{3n \times 3n}$, which itself contains the internal energy matrix $\mathbf{A}_i$ and the external energy matrix $\mathbf{A}_e$, i.e.

$$\mathbf{A} = \mathbf{A}_i + \mathbf{A}_e,$$

$\mathbf{A}_i, \mathbf{A}_e \in \mathbb{R}^{3n \times 3n}$.

If all the control points $\mathbf{d}_i$ are collected in the concatenation vector $\mathbf{c} \in \mathbb{R}^{3n}$, a quadratic programming problem with linear constraints

$$\text{minimize} \quad \mathbf{c}^T\mathbf{A}\mathbf{c} + \mathbf{b}^T\mathbf{c} + k \tag{5.3}$$

$$\text{such that} \quad \mathbf{D}\mathbf{c} = \mathbf{e}, \tag{5.4}$$

will be obtained where $\mathbf{b} \in \mathbb{R}^{3n}$ contains the linear contributions of the energy terms in E, and k contains the constant parts. Eq. (5.4) with $\mathbf{D} \in \mathbb{R}^{m\times 3n}, \mathbf{e} \in \mathbb{R}^m$ comprises the m linear constraints, which can be used to fix the boundary of the curve or surface.

The problem (5.3) can be solved efficiently by various linear equation solving algorithms, which will not be discussed here. Refer e.g. to the book of Luenberger [Luen84], or to [Wess96], where the most common means are briefly summarized.

In many cases (e.g. for all deformation tools presented in this work, see 5.2.2), the equations of the energy contributions can be split in three independent equations for the coordinates of the control points $\mathbf{d}_i$. This results in three independent minimization problems, lowering the dimension of the problem by a factor three.

In variational modeling, it is necessary to apply constraints to the curves or surfaces subject to energy minimization. This is easy to see, since the energy of an unconstrained curve can be made zero by shrinking the curve so that all its points collapse to a single point. Various kinds of constraints exist but most importantly; positional, directional, continuity, and interpolation. See [Wess96] for a detailed description. In order to keep the minimization problem solvable within a reasonable amount of time, constraints that are linear in the control points are preferable.

5.2 Curve shaping tools

For the following, we assume that a curve or a curve segment is given by eq. (5.1). The tools described in this section are implemented for non-uniform cubic B-spline curves. The simple approximations for the bend energy

$$E_b = \int_{t_0}^{t_1} \|\ddot{\mathbf{x}}_{min}(t)\|^2 dt \tag{5.5}$$

and the stretch energy

$$E_s = \int_{t_0}^{t_1} \|\dot{\mathbf{x}}_{min}(t)\|^2 dt \tag{5.6}$$

for curves as given in [WV95] are then quadratic in the control points $\mathbf{d}_i$ of eq. (5.1). They are given by

$$E = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} a_{ij}\mathbf{d}_i^T\mathbf{d}_j, \tag{5.7}$$

where

$$a_{ij,b} = \int_{t_0}^{t_1} \ddot{N}_i(t)\ddot{N}_j(t)dt \tag{5.8}$$

are the matrix elements for the bend energy ($E = E_b$), and

$$a_{ij,s} = \int_{t_0}^{t_1} \dot{N}_i(t)\dot{N}_j(t)dt \tag{5.9}$$

are the matrix elements for the stretch energy ($E = E_s$). t_0 and t_1 denote the curve parameter interval where the curve $\mathbf{x}_{min}$ is supposed to be minimized. E_b and E_s are parts of a weighted sum that describes the internal energy of a curve,

$$E_i = w_b E_b + w_s E_s. \tag{5.10}$$

In order to allow more control over the shape of a curve, energy terms that cause certain deformation effects, referred to as attractors, contribute to the external energy E_e of a curve, so that the total energy of the curve is given by

$$E = w_i E_i + w_e E_e. \tag{5.11}$$

5.2.1 Improvements on variational modeling

Having such tools at hand, designers are able to influence the fairing process in a way that the curve adopts the desired shape. That kind of usage of variational methods, as proposed e.g. by [Wess96], can be interpreted rather as a curve creation process, than as an interactive deformation process on a given curve. Positional and directional end conditions are prescribed in the first step, and following the minimization step; eventually biased by additional attracting or repelling energy terms. As a result, a curve designed from scratch is obtained, for which all constraints and energy contributions are known. Therefore, they can be used or modified accordingly for refining its shape.

However, in case of curves given e.g. by a drawing process, within the region covered by the minimization shape details tend to decrease. Previously defined features are wiped out by such a procedure.

Shape preserving

Since obtaining a perfectly smooth curve certainly is not always the desired result of applying a deformation tool, energy terms that could *preserve* the shape of a curve would be very useful. For that purpose, the following energy terms are introduced here.

$$E_{p0} = \int_{t_0}^{t_1} \|\dot{\mathbf{x}}_{min}(t) - \dot{\mathbf{x}}_{ref}(t)\|^2 dt \tag{5.12}$$

$$E_{p1} = \int_{t_0}^{t_1} \|\ddot{\mathbf{x}}_{min}(t) - \ddot{\mathbf{x}}_{ref}(t)\|^2 dt \tag{5.13}$$

Suppose that $\mathbf{x}_{min}(t)$ and $\mathbf{x}_{ref}(t)$ both have n control points $\mathbf{p}_i$ and $\mathbf{q}_j$, respectively. Then both E_{p0} and E_{p1} can be written in the form

$$E_{p0|p1} = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} a_{ij}\mathbf{p}_i^T\mathbf{p}_j + \sum_{i=0}^{n-1}\mathbf{b}_i^T\mathbf{p}_i + c$$

with

$$a_{ij,p0} = \int_{t_0}^{t_1} \dot{N}_i(t)\dot{N}_j(t)dt,$$

$$a_{ij,p1} = \int_{t_0}^{t_1} \ddot{N}_i(t)\ddot{N}_j(t)dt,$$

and

$$\mathbf{b}_{i,p0} = -2\sum_{j=0}^{n-1}\mathbf{q}_j \int_{t_0}^{t_1} \dot{N}_i(t)\dot{N}_j(t)dt,$$

$$\mathbf{b}_{i,p1} = -2\sum_{j=0}^{n-1}\mathbf{q}_j \int_{t_0}^{t_1} \ddot{N}_i(t)\ddot{N}_j(t)dt.$$

The constant term c is given by

$$c_{p0} = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\mathbf{q}_i^T\mathbf{q}_j \int_{t_0}^{t_1} \dot{N}_i(t)\dot{N}_j(t)dt,$$

$$c_{p1} = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} \mathbf{q}_i^T \mathbf{q}_j \int_{t_0}^{t_1} \ddot{N}_i(t)\ddot{N}_j(t)dt.$$

A combination of both E_{p0} and E_{p1}

$$E_p = w_{p0}E_{p0} + w_{p1}E_{p1} \tag{5.14}$$

seems to produce the best results. These energy terms tend to minimize the variations between two curves. If $\mathbf{x}_{ref}(t)$ is initialized with the original curve and $\mathbf{x}_{min}(t)$ is set to the results of the minimization, features of the original curve will be preserved. The degree of shape preserving relative to other deformation effects, e.g. smoothing, can be controlled by using appropriate weights.

Locality

As it has been shown, multiplying it with a weight factor can control the influence of an energy term. It is therefore possible e.g. to reduce a smoothing effect in favor of preserving the shape. However, the corresponding weight setting is constant on the whole curve. Interactive deformation tools would greatly benefit from a more flexible approach. A combination of energy terms with weight functions defined on the parametric domain of the curve would allow to spread the influence of an energy term along the curve unevenly.

As an application, "local" bend and stretch energy terms E_{bl} and E_{sl} are proposed in this dissertation as follows:

$$E_{bl} = \int_{t_0}^{t_1} w_{bl}(t)\|\ddot{\mathbf{x}}_{min}(t)\|^2 dt, \tag{5.15}$$

$$E_{sl} = \int_{t_0}^{t_1} w_{sl}(t)\|\dot{\mathbf{x}}_{min}(t)\|^2 dt, \tag{5.16}$$

where $w_{bl}(t)$ and $w_{sl}(t)$ are user defined weight functions for controlling the amount of smoothing along the curve. Both E_{bl} and E_{sl} can be written as

$$E_{bl|sl} = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} a_{ij}\mathbf{p}_i^T \mathbf{p}_j$$

with

$$a_{ij,bl} = \int_{t_0}^{t_1} w_{bl}(t)\ddot{N}_i(t)\ddot{N}_j(t)dt, \tag{5.17}$$

$$a_{ij,sl} = \int_{t_0}^{t_1} w_{sl}(t)\dot{N}_i(t)\dot{N}_j(t)dt. \tag{5.18}$$

In this manner, it is easy to combine energy terms to develop new interactive, intuitive curve deformation tools. In sections 5.2.3 and 5.2.4, examples how this can be achieved are presented.

5.2.2 Setting up the minimization system

All energy functionals that are used in this work, including those for surfaces, can be written in the form

$$E = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} a_{ij}\mathbf{p}_i^T \mathbf{p}_j + \sum_{i=0}^{n-1} \mathbf{b}_i^T \mathbf{p}_i + c, \tag{5.19}$$

with n control points $\mathbf{p}_i$, quadratic coefficients a_{ij}, vector-valued linear coefficients $\mathbf{b}_i$, and a constant c. Note that this allows us to split the problem (5.3) in three independent systems for each coordinate.

The quadratic contributions

The structure of the energy matrices and how they are set up is described in detail by Wesselink [Wess96]. Suppose the quadratic coefficients a_{ij} are arranged in an $n \times n$ matrix, i.e. $\bar{\mathbf{A}} = (a_{ij})$. A cubic B-spline curve is interpreted as a sequence of cubic segments, each defined by four control points and basis functions. The energy of a curve is the sum of the energy of its segments. Therefore, for the whole curve, the matrix $\bar{\mathbf{A}}$ can be constructed by starting with an $n \times n$ zero matrix $\bar{\mathbf{A}}$ and then adding the 4×4 matrices corresponding to the segments to $\bar{\mathbf{A}}$ so that they overlap appropriately. In case of uniform splines, each segment contributes the same amount of internal energy to the system, which means that the submatrices $(a_{ij,b})$ and $(a_{ij,s})$ of a segment, once they are calculated, can be read from a file when the application starts.

In our case, $\bar{\mathbf{A}}$ can directly be used in

$$\text{minimize} \quad \mathbf{c}_x^T \bar{\mathbf{A}} \mathbf{c}_x + \mathbf{b}_x^T \mathbf{c}_x + k, \tag{5.20}$$

$$\text{such that} \quad \bar{\mathbf{D}} \mathbf{c}_x = \mathbf{e}_x, \tag{5.21}$$

$\mathbf{b}_x \in \mathbb{R}^n, \bar{\mathbf{D}} \in \mathbb{R}^{m \times n}, \mathbf{e}_x \in \mathbb{R}^m$, where $\mathbf{c}_x \in \mathbb{R}^n$ is the concatenation of all x-coordinates of $\mathbf{c}$; same for the y- and z-coordinates.

In the general case, i.e. to set up the minimization system (5.3), the energy of a curve must be expressed in the concatenation vector $\mathbf{c}$, which requires an expanded $3n \times 3n$ matrix $\mathbf{A}$. It is constructed by replacing each element a_{ij} of $\bar{\mathbf{A}}$ with the 3×3 matrix

$$\begin{pmatrix} a_{ij} & 0 & 0 \\ 0 & a_{ij} & 0 \\ 0 & 0 & a_{ij} \end{pmatrix}, \tag{5.22}$$

resulting in a banded matrix $\mathbf{A}$ with upper and lower bandwidth of 9 [Wess96].

The linear contributions

To set up (5.20), the x, y, z-coordinates of the linear coefficients $\mathbf{b}_i$ can be assembled into $\mathbf{b}_x, \mathbf{b}_y, \mathbf{b}_z$, respectively. In the general case, to set up the term $\mathbf{b}^T\mathbf{c}$ in eq. (5.3), the $\mathbf{b}_i$ can simply be concatenated to the vector $\mathbf{b} \in \mathbb{R}^{3n}$.

Note that the constant part c is not needed to solve the problems (5.3) and (5.20). It can therefore be omitted.

5.2.3 A curve smoother

With the ideas introduced in section 5.2.1, a curve smoother has been developed that interactively allows controlling the amount as well as the location of smoothing. It utilizes a kind of "flat-iron" metaphor. The pointer is being moved along the curve until the desired shape has been achieved. The current location of the pointer causes the maximal effect of the smoother. By moving the pointer back and forth, the curve is being smoothed out gradually. Beside the region of the influence of the pointer, the curve shape is preserved. It has proven very easy to control the exactness of the smoothing process in that way. Especially for larger deformations, the amount of arm movement can be reduced to pointing to the location where the effect is intended to increase.

Optionally, the current distance of the pointer to the curve is taken into account. A small distance causes a short smoothing interval on the curve, which increases when the pointer is moved away from the curve. The overall smoothing effect decreases over time as long as the curve is active.

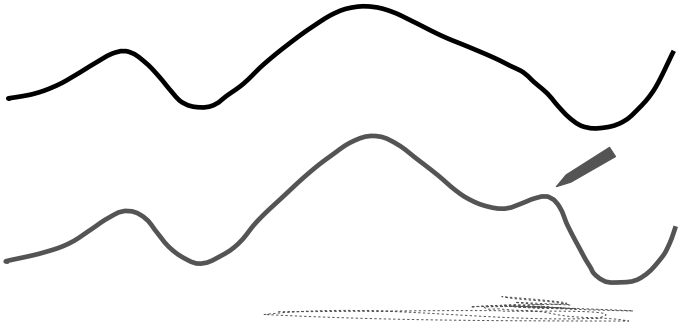

Figure 5.1: Smoothing a curve locally. The original curve is shown in grey. A dotted path indicates how the pointer is moved along the curve. A curve region gets smoother by continuously pointing at it.

An interactive curve smoother can be assembled from the following energy terms.

$$E_{smooth} = w_b E_{bl} + w_s E_{sl} + w_p E_p, \tag{5.23}$$

where $w_{bl}(t)$ and $w_{sl}(t)$ are chosen to be an exponential weight function of Gaussian shape with its maximum at the parametric location t_q, where the designer points at the curve:

$$w_{bl}(t) = w_{sl}(t) = e^{-a(t-t_q)^2}. \tag{5.24}$$

Here, $a > 0$, w_b, and w_s have to be chosen by the designer. This interactive smoother works in the following way. First, a curve $\mathbf{x}_{min}$ is selected that is supposed to be smoothed. In each frame of the application process, the curve parameter t_q is determined from the location of the input device which should be reasonable close to the curve. The current state of $\mathbf{x}_{min}(t)$ is copied into $\mathbf{x}_{ref}(t)$, to calculate the shape preserving term E_p, and a new E_{smooth}, which defines the matrix $\mathbf{A}$ for the minimization step. The solution produces a curve which is slightly smoother at the location of the pointer, but still retains its details beside the maximum of the Gaussian. This curve is taken as $\mathbf{x}_{ref}(t)$ for the next application frame. The curve is gradually smoothed out, depending on the weights. Figure 5.1 shows an example.

5.2.4 A curve sharpener

The curve sharpener can be interpreted as the inverse operation to the curve smoother. It seems to be peculiar to create a sharpening operation with variational methods. However, it is easy to implement the desired effect by simply choosing negative weights w_b and w_s in equation (5.23) for the weight function of the internal energy.

The curve sharpener is applied in the same way as the curve smoother, but with the effect that initial details on the curve are elaborated. As opposed to the smoother, the sharpening effect increases over time. This means that the whole process does not converge. Usually the sharpening tool is deactivated before this critical situation occurs. Figure 5.2 shows a curve that has been locally sharpened.

5.2.5 A curve dragger

A useful curve drag tool would allow attraction of a segment towards the pointer location, directly corresponding to the hand movement. The basic shape details of the dragged curve should be preserved as it has been initially drawn. The influence

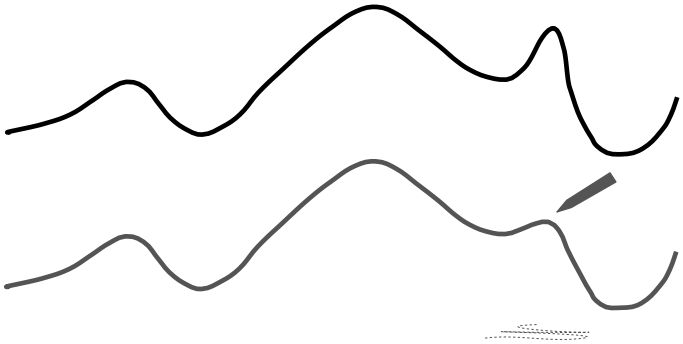

Figure 5.2: Sharpening a curve locally. Compared to Figure 5.1, the influenced region is shorter.

range of the curve dragger can be chosen to be narrow or wide, corresponding to a peak- or a dent-like result. A curve segment, fixed by its two connection points in the curve network, can be warped around as desired. The use of the curve dragger can benefit from two-handed interaction. In this mode, the non-dominant hand is changing the orientation of the model while the dominant hand drags the curve (see chapter 6).

Attractors have been proposed by [WV95] to pull a curve segment towards a given point $\mathbf{q}$, towards a line or a plane, or to pull a point at the location t_q on the curve towards $\mathbf{q}$. As an example, the *point to point attractor* is given.

$$\begin{aligned} E_{pp} &= \|\mathbf{x}_{min}(t_q) - \mathbf{q}\|^2. \\ &= \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} a_{ij}\mathbf{p}_i^T\mathbf{p}_j + \sum_{i=0}^{n-1}\mathbf{b}_i^T\mathbf{p}_i + c \end{aligned} \tag{5.25}$$

with

$$a_{ij} = N_i(t_q)N_j(t_q), \qquad \mathbf{b}_i = -2N_i(t_q)\mathbf{q}, \qquad c = \mathbf{q}^T\mathbf{q}.$$

To create a curve drag tool which preserves the details and features of the selected segment (Figure 5.3), E_{pp} is combined with a shape preserving term E_p selected from the equations (5.12), (5.13), or (5.14), introduced in section 5.2.1:

$$E_{drag} = w_{pp}E_{pp} + w_pE_p. \tag{5.26}$$

The way the curve dragger works is a little bit different from the smoothing and sharpening procedures. When the dragger is activated with a curve $\mathbf{x}_{min}(t)$, $\mathbf{x}_{ref}(t)$ in eq. (5.14) is initialized with $\mathbf{x}_{min}(t)$ in order to preserve its details. In the following frames of the application process, only $\mathbf{q}$ in eq. (5.25) needs to be updated with the current pointer position.

Advantages

Note that these deformation tools produce similar results as the simulation of a force effect on a curve. However, the way of controlling smoothing and sharpening a curve is rather indirect compared to directly pulling a curve. Especially for larger deformations, the amount of arm movement can be reduced to pointing to the location where the effect is supposed to increase. Moreover, it seems to be easier to control the exactness of smoothing and sharpening processes in this way rather than with direct deformations. By applying the intuitive "flat-iron" metaphor, the effect of smoothing out the curve can be distributed in a very elegant way.

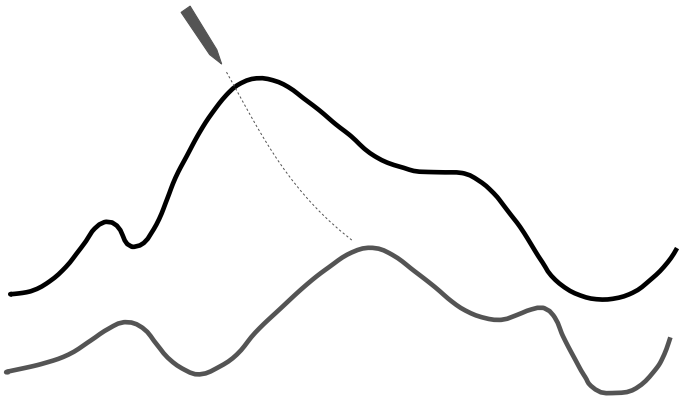

Figure 5.3: Dragging a curve segment

For deformation tools applied to virtual objects, force feedback devices are often demanded to overcome the feeling of interacting in the air. However, such devices involve serious drawbacks such as disturbing stereo perception by reaching into the viewing frustrum or restricting the work area, as stated also by Schkolne et al. [SPS01]. An indirect mapping of hand movements to the smoothing or sharpening effect, as it has been developed here, elegantly circumvents the problem of missing force feedback.

5.2.6 Computing boundary curves

As is has been described in section 4.5.3, surface parts are constructed merely from their surrounding boundary. Each part of the boundary is created as a uniform cubic B-spline curve that approximates the drawn network curve as closely as possible. For the similar purpose of attracting a curve towards a reference curve, appropriate energy terms have already been introduced, refer to equation (5.14). Note that the drawn curve can have a non-uniform parameterization.

Let the segment of the drawn curve and the approximating curve used as part of the surface boundary be denoted by $\mathbf{x}_{draw}(t)$ and $\mathbf{x}_{min}(t)$, respectively, and further assume that the parameter domain of $\mathbf{x}_{draw}(t)$ has been linearly transformed to that of $\mathbf{x}_{min}(t)$, so that $t \in [t_0 \dots t_1]$. With

$$\mathbf{x}_{min}(t) = \sum_{i=0}^{n-1} \mathbf{p}_i N_i(t), \tag{5.27}$$

$$\mathbf{x}_{draw}(t) = \sum_{j=0}^{m-1} \mathbf{q}_j M_j(t), \tag{5.28}$$

where $N_i(t)$ are cubic basis functions on a uniform knot vector, and $M_j(t)$ are non-uniform cubic basis functions, the following energy term can be used to create $\mathbf{x}_{min}(t)$:

$$\begin{aligned} E_a &= \int_{t_0}^{t_1} \|\dot{\mathbf{x}}_{min}(t) - \dot{\mathbf{x}}_{draw}(t)\|^2 dt \\ &= \sum_{i_0=0}^{n-1} \sum_{i_1=0}^{n-1} a_{i_0 i_1} \mathbf{p}_{i_0}^T \mathbf{p}_{i_1} - 2 \sum_{i=0}^{n-1} \mathbf{b}_i^T \mathbf{p}_i + c, \end{aligned}$$

with

$$a_{i_0 i_1} = \int_{t_0}^{t_1} N_{i_0}(t) N_{i_1}(t) dt$$

$$\mathbf{b}_i = \sum_{j=0}^{m-1} \mathbf{q}_j \int_{t_0}^{t_1} N_i(t) M_j(t) dt$$

$$c = \sum_{j_0=0}^{m-1} \sum_{j_1=0}^{m-1} \mathbf{q}_{j_0}^T \mathbf{q}_{j_1} \int_{t_0}^{t_1} M_{j_0}(t) M_{j_1}(t) dt$$

Fixing curve ends

Usually it is required that a curve reproduces certain constraints when deformation tools are applied to it. It is, for example, in most cases desirable to fix the positions and tangent vectors at the ends of a deforming curve segment. For general parametric curves, positional as well as directional constraints are needed to achieve this, as described by Wesselink [Wess96].

Fortunately, controlling the behavior of B-spline curves at the ends is easy since the shape parameters are the control points. Due to the local support of the B-spline basis, positional constraints applied to control points at the ends are sufficient. In case a standard cubic B-spline representation with triple end knots and corresponding basis functions at the ends is given for $\mathbf{x}_{min}(t)$, the control point constraints can be written as

$$\mathbf{d}_i = \mathbf{p}_i, \qquad i = 0, 1, n-2, n-1, \tag{5.29}$$

where $\mathbf{d}_i$ is a control point of $\mathbf{x}_{min}(t)$ determined by the minimization (5.20), and $\mathbf{p}_i$ is its prescribed position. In case a curve region consisting of inner segments is subject to minimization, and the inner knots are simple, at least the first three control points of the first cubic segment and the last three control points of the last segment should be fixed, since the corresponding basis functions have a support of 4 knot spans. Note that eq. (5.29) can be split into three independent coordinate equations. They can be expressed in terms of the concatenation vectors $\mathbf{c}_{x|y|z}$ by setting the appropriate places of the constraint matrix $\bar{\mathbf{D}}$ in eq. (5.21) to 1 and the corresponding places of $\mathbf{e}_{x|y|z}$ to the coordinates p_{ix}, p_{iy}, and p_{iz}.

5.3 Surface shaping tools

In the following, a surface shall be given by eq. (5.2). As for curves, a weighted sum E_i of simple approximations to the *thin plate* or bend energy E_b and to the stretch energy E_s is commonly used as the internal energy contribution to the fairness functional:

$$E_b = \int_\Gamma (\mathbf{s}_{uu}^T \mathbf{s}_{uu} + 2\mathbf{s}_{uv}^T \mathbf{s}_{uv} + \mathbf{s}_{vv}^T \mathbf{s}_{vv}) du dv, \tag{5.30}$$

$$E_s = \int_\Gamma (\mathbf{s}_u^T \mathbf{s}_u + \mathbf{s}_v^T \mathbf{s}_v) du dv, \tag{5.31}$$

$$E_i = w_b E_b + w_s E_s. \tag{5.32}$$

Here, $\Gamma \subseteq \Omega$ denotes the region for the optimization (5.3). If the canonical surface representation (5.2) is substituted into eqs. (5.30) or (5.31), a quadratic expression in the control points $\mathbf{d}_i$ is found, so that these functions can be very efficiently minimized:

$$E = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_{ij} \mathbf{d}_i^T \mathbf{d}_j, \tag{5.33}$$

Concatenating the control points in the vector

$$\mathbf{c} = (d_{00}, d_{01}, d_{02}, \ldots, d_{n-1,0}, d_{n-1,1}, d_{n-1,2})^T,$$

eq. (5.33) can be written in matrix form,

$$E = \mathbf{c}^T \mathbf{A} \mathbf{c}, \tag{5.34}$$

with $\mathbf{A}$ obtained from (a_{ij}) by applying (5.22). For the bend energy (5.30) and the stretch energy (5.31), $a_{ij,b}$ and $a_{ij,s}$ can be written as

$$a_{ij,b} = \int_\Gamma (M_{i,uu} M_{j,uu} + 2 M_{i,uv} M_{j,uv} + M_{i,vv} M_{j,vv}) du dv, \tag{5.35}$$

$$a_{ij,s} = \int_\Gamma (M_{i,u} M_{j,u} + M_{i,v} M_{j,v}) du dv. \tag{5.36}$$

Similar to eq. (5.11), attractors contribute to an external energy E_e and can be used to control the shape resulting from the minimization process. The total energy of the surface becomes, as in the curve case

$$E = w_i E_i + w_e E_e, \tag{5.37}$$

where w_i and w_e control the relative influence of the internal and external contributions.

Using the concatenation vector $\mathbf{c}$, a minimization problem (5.3) under linear constraints (5.4) is obtained. The constraints can be used to fix the control points of the boundary.

It would be useful to have similar deformation tools available for surfaces as introduced for curves in section 5.2. In fact, it is possible to define analogous energy terms to perform local smoothing or sharpening, including shape preserving behavior. Unfortunately, a much larger set of control points is needed to define a surface shape, and to provide the necessary degrees of freedom in order to usefully apply these tools. Such a high number of control points prevent interactive frame rates. Therefore, currently the implemented deformation tools are restricted, so that a constant energy matrix results.

For surfaces, a different approach to implement shape preserving behavior is used. From the internal energy and the control points, the equilibrium

$$\mathbf{b}_q = \mathbf{A}_i \mathbf{c} \tag{5.38}$$

is computed and added to the right hand side of the linear system used to solve the minimization problem (5.3). In eq. (5.3),

$$\mathbf{b} = \mathbf{b}_e - w_q \mathbf{b}_q, \tag{5.39}$$

is set where $\mathbf{b}_e$ contains the linear contributions of all attractors in the system. The weight w_q controls how much the result approaches the minimal-energy surface. $w_q = 1$ results in the original surface, whereas $w_q = 0$ causes the minimal-energy surface (assumed $\mathbf{b}_e = \mathbf{0}$).

5.3.1 A surface smoother

The surface smoother is designed in a way that stopping the process when appropriate can control the degree of smoothness. In contrast to the curve case, weighted

energy functions are not implemented. The energy term is simply the combination of the bend and stretch energies:

$$E_{smooth} = w_i E_i. \tag{5.40}$$

In addition, an equilibrium is calculated, in order to delay the smoothing process over time. In each frame of the application process, w_q in eq. (5.39), initialized with 1, is multiplied by a factor $0 < f < 1$. The result becomes the new w_q, such that the influence of $\mathbf{b}_q$ is decreasing. Note that $\mathbf{b_e} = \mathbf{0}$, since no external contributions are used. Because the energy matrix $\mathbf{A}$ needs to be initialized only once, the smoother can be used interactively. In each frame, after calculating $\mathbf{b}_q$ and $\mathbf{b}$, only the solving step needs to be performed, resulting in a slightly smoother surface than in the previous frame.

5.3.2 A surface sharpener

A sharpening tool can be derived from the smoother by simply using an $f > 1$. This causes the whole surface to "inflate", until stopped by the user. This is to avoid an "explosive–like" behaviour of the sharpening process.

5.3.3 A surface dragger

The surface dragger can be used to deform a surface region similar to curves. As a shape preserving term, the equilibrium $\mathbf{b}_q$ from eq. (5.38) is used. In addition to an internal energy term, a *point to point attractor* for surfaces is needed, which is defined in [Wess96] as

$$\begin{aligned} E_{pp} &= \|\mathbf{s}(u_0, v_0) - \mathbf{p}\|^2 \\ &= \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} a_{ij}\mathbf{d}_i^T\mathbf{d}_j + \sum_{i=0}^{n-1}\mathbf{b}_i^T\mathbf{d}_i + c, \end{aligned}$$

with

$$a_{ij} = M_i(u_0, v_0)M_j(u_0, v_0), \qquad \mathbf{b}_i = -2M_i(u_0, v_0)\mathbf{p}, \qquad c = \mathbf{p}^T\mathbf{p}.$$

Here, (u_0, v_0) is the parametric location of the surface point which is supposed to be attracted towards $\mathbf{p}$. The total energy in case of surface dragging becomes

$$E_{drag} = w_i E_i + w_{pp} E_{pp}. \tag{5.41}$$

Recall that $\mathbf{b}_q$ is computed only from E_i; the quadratic contribution from E_{pp} does not influence the equilibrium $\mathbf{b}_q$. It has to be initialized when the dragger starts working and retains its value in the following frames. Only E_{pp}, and therefore $\mathbf{b}_e$ and $\mathbf{b}$ (see eq. (5.39)) have to be updated before the solution can be calculated.

5.3.4 Computing initial surface shapes

Although interactive surface deformation tools are available, it is obvious that supporting the shape creation process with reasonable initial shapes would be very helpful. For a modeler based on curve networks, it seems to be appropriate to create surface shapes according to their surrounding curves, since in many cases, an energy-minimal surface does not exhibit the desired shape. Consider e.g. a simple curve network that outlines a part as shown in Figure 5.4. The surface sags in the middle, which is created by minimizing its thin-plate energy. In many cases, a stretched shape might be more intuitive. Therefore, in the modeler described in this work, the user can choose between two methods for the creation of the initial surfaces.

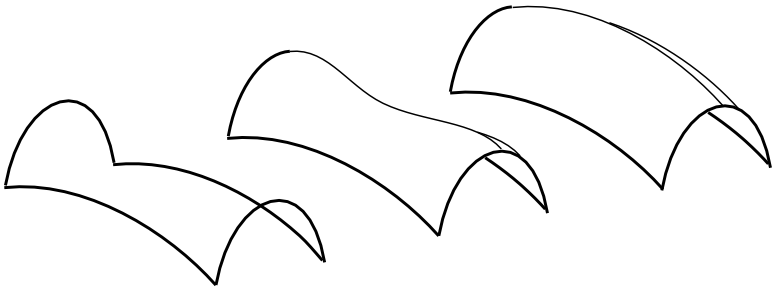

Figure 5.4: A surface that has a minimal energy (middle), created under the boundary constraints of the curve network (left), often might not correspond to the intended result. The surface on the right has a shape that better reproduces the outline.

Fixing the boundary

The transition between neighboring surface parts should reproduce the transitions of the surrounding curves.

Analogously to the curve case (see section 5.2.6), a narrow boundary area can be constructed within a surface part, by prescribing a stripe of control points around the surface part, with a width of three. Each setting of a control point involves a control point constraint similar to (5.29), which is added to the constraint system (5.4). This way, the boundary curves of the surface part, as well as the cross boundary derivatives can be determined. It has been shown in section 4.5.3, that the stripe of control points can be set in such a way that boundary curves are interpolated. These uniformly parameterized curves $\mathbf{x}_{min}(t)$ have the representation (5.27) and are itself approximations to the drawn network curves $\mathbf{x}_{draw}(t)$ from eq. (5.28), see section 5.2.6.

Furthermore, two surfaces can be made C^1 continuous across their common boundary, if the corresponding two opposite boundary curves are at least C^1 continuous as well, see section 4.5.3.

Initial shapes with minimized energy

In case a fair shape in the sense of variational modeling is what is desired, this can be achieved by using a fairness norm, such as (5.32),

$$E_i = w_b \int_\Gamma (\mathbf{s}_{uu}^T \mathbf{s}_{uu} + 2\mathbf{s}_{uv}^T \mathbf{s}_{uv} + \mathbf{s}_{vv}^T \mathbf{s}_{vv}) dudv + w_s \int_\Gamma (\mathbf{s}_u^T \mathbf{s}_u + \mathbf{s}_v^T \mathbf{s}_v) dudv,$$

as the only energy contribution. Due to the used constraints, the shape of the surrounding curves of the network only influences the boundary area of the surface part, as described in the previous paragraph.

Initial shapes based on reference surfaces

The surfaces of Kuriyama [Kuri94] (see section 3.2.5) have been proposed to better reproduce sketched principal curves of a network. Although the approach of Kuriyama does not allow direct deformations, his surfaces can be used as reference surfaces to fit in Catmull-Clark surface parts that adopt a similar shape. A *region to region* attractor, as defined by Wesselink [Wess96] can be used for this task:

$$E_{ref} = \sum_{d=1}^{N} \int_{\Gamma_d} \|\mathbf{s}_d(u,v) - \mathbf{r}_d(u,v)\|^2 dudv.$$

Here, the N-sided domain Γ has been split into N rectilinear subdomains Γ_d, according to Figure 4.9 on page 59. The $\mathbf{s}_d(u,v)$ represent the rectangular regions of the Catmull-Clark surface part written in the canonical surface representation (5.2), whereas the $\mathbf{r}_d(u,v) = \mathbf{q}_d^N(\mathbf{b}_d(u,v))$ depicts the part of the Kuriyama surface defined on Γ_d. $\mathbf{b}_d(u,v)$ maps the rectilinear (u,v) coordinates into barycentric coordinates needed to evaluate $\mathbf{q}_d^N$ according to eq. (3.11). The N parts $E_{ref,d}$ of the sum E_{ref} are given by

$$\begin{aligned} E_{ref,d} &= \int_{\Gamma_d} \mathbf{s}^T\mathbf{s} du dv - 2\int_{\Gamma_d} \mathbf{s}^T\mathbf{r}_d du dv + \int_{\Gamma_d} \mathbf{r}_d^T\mathbf{r}_d du dv \\ &= \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} a_{ij}\mathbf{d}_i^T\mathbf{d}_j - 2\sum_{i=0}^{n-1}\mathbf{b}_i\mathbf{d}_i + c \end{aligned}$$

with

$$\begin{aligned} a_{ij} &= \int_{\Gamma_d} M_i(u,v)M_j(u,v) du dv, \\ \mathbf{b}_i &= \int_{\Gamma_d} M_i(u,v)\mathbf{r}_d(u,v) du dv, \\ c &= \int_{\Gamma_d} \mathbf{r}_d^T(u,v)\mathbf{r}_d(u,v) du dv \end{aligned}$$

The terms a_{ij} and $\mathbf{b}_i$ are used to set up the matrix $\mathbf{A}$ and the vector $\mathbf{b}$ in the minimization problem (5.3). Note that the constant part c is not needed to solve (5.3). It can therefore be omitted.

How the a_{ij} are computed depends on the actual basis functions of the surface representation (5.2). For Catmull-Clark surface parts used in the modeler described here, solutions are provided in the next section. The terms $\mathbf{b}_i$ impose more difficulties, since the surfaces of Kuriyama are represented in terms of generalized barycentric coordinates. For the sake of simplicity, the integrals $\mathbf{b}_i$ are approximated numerically by means of the gaussian quadrature formulas, see [PTVF92].

5.4 Energy terms for Catmull-Clark surfaces

Catmull-Clark surface parts, as constructed for the modeler described in this work (see 4.5.2), consist of uniform bicubic patches, each defined by 16 control points and basis functions, and N patches surrounding the extraordinary middle vertex in case of N-sided domains, $N = 3$ or $N = 5$. Similar to curves, the total energy of a surface part is the sum of all energy contributions of its patches. Therefore, to calculate a certain energy functional, the energy matrices (a_{ij}) for two kinds of patches are needed.

5.4.1 Energy terms for uniform bicubic patches

A bicubic B-spline patch can be written in tensor product form,

$$\begin{aligned} \mathbf{b}(u,v) &= \sum_{k=0}^{3}\sum_{l=0}^{3} \mathbf{d}_{kl} N_k(u) N_l(v) \\ &= \sum_{i=0}^{15} \mathbf{d}_i N_{i\%4}(u) N_{i/4}(v), \end{aligned} \tag{5.42}$$

$(u,v) \in [0,1] \times [0,1]$.

Here, the $M_i(u,v)$ from the canonical representation (5.2) are replaced with products of univariate, uniform B-spline basis functions $N_{i\%4}(u)N_{i/4}(v)$, given in eq. (4.5), where "%"and "/"stand for the remainder and the division respectively. The elements of the energy matrices $(a_{ij,b})$ of eq. (5.35), and $(a_{ij,s})$ of eq. (5.36), can then be represented using products of elements of the curve energy matrices, as shown for one of the terms that appears in eq. (5.35):

$$\begin{aligned} a_{ij,uu} &= \int_\Gamma M_i(u,v)_{uu} M_j(u,v)_{uu} dudv \\ &= \int_0^1 \int_0^1 (N_{i\%4}(u)N_{i/4}(v))_{uu} (N_{j\%4}(u)N_{j/4}(v))_{uu} dudv \\ &= \int_0^1 N_{i/4}(v)N_{j/4}(v)dv \int_0^1 N_{i\%4}(u)_{uu} N_{j\%4}(u)_{uu} du \end{aligned} \tag{5.43}$$

The other parts of the sum in eq. (5.35) and eq. (5.36) are computed analogously. For convenience, let them be named according to their subscript, indicating the differentiation, e.g.

$$\begin{aligned} a_{ij,uv} &= \int_\Gamma 2M_i(u,v)_{uv} M_j(u,v)_{uv} dudv, \quad \text{or} \\ a_{ij,u} &= \int_\Gamma M_i(u,v)_u M_j(u,v)_u dudv. \end{aligned}$$

Since the basis functions are the same for each uniform bicubic patch, the energy submatrix is the same for all those patches.

5.4.2 Energy terms for patches around an extraordinary vertex

In his paper about exact evaluation of Catmull-Clark surfaces [Stam98], Stam presents a formula to evaluate a patch, containing exactly one extraordinary vertex that has a valency of N, at arbitrary parameter values. In Figure 3.5 on page 38 it can be seen for which configuration the method of Stam is valid.

The algorithm of Stam however is not only useful for the evaluation of patches. Since basis functions for extraordinary patches are obtained, so that those patches can be written in the canonical representation (5.2),

$$s(u,v) = \sum_{i=1}^{K} \mathbf{p}_i B_i(u,v), \tag{5.44}$$

their energy can be written in the form

$$E_{ext} = \sum_{i=1}^{K} \sum_{j=1}^{K} e_{ij} \mathbf{p}_i^T \mathbf{p}_j, \tag{5.45}$$

which is quadratic in the $K = 2N + 8$ coefficients $\mathbf{p}_i$. In case it would be possible to analytically derive solutions to the integrals e_{ij}, similar to eqs. (5.35) and (5.36), the energy contribution of an extraordinary patch would be obtained.

In the following, it is shown how the integrals e_{ij} can be derived, using the definition of the basis functions of eq. (3.9). Similar to the derivation of $a_{ij,uu}$, given by eq. (5.35), the first part of the sum under the integral of the bend energy, $e_{ij,uu}$, is taken as an example.

$$\begin{aligned} e_{ij,uu} &= \int_{\Omega} B_i(u,v)_{uu} B_j(u,v)_{uu} du dv \\ &= \sum_{n=1}^{\infty} \sum_{k=1,2,3} \int_{\Omega_k^n} B_i(u,v)_{uu} B_j(u,v)_{uu} du dv. \end{aligned}$$

By substituting $B_i(u,v)\big|_{\Omega_k^n} = (\lambda_i)^{n-1} x_i(u_{kn}, v_{kn}, k)$ with $(u_{kn}, v_{kn}) = \mathbf{t}_{k,n}(u,v)$, as defined by eq. (3.10) we obtain:

$$e_{ij,uu} = \sum_{n=1}^{\infty} \sum_{k=1,2,3} \lambda_i^{n-1} \lambda_j^{n-1} \int_{\Omega_k^n} x_i(u_{kn}, v_{kn}, k)_{uu} x_j(u_{kn}, v_{kn}, k)_{uu} du dv.$$

Mapping each tile Ω_k^n, which form the integration domain, to the unit square, as done by the transformation $\mathbf{t}_{k,n}(u,v)$, yields

$$e_{ij,uu} = \sum_{n=1}^{\infty} \sum_{k=1,2,3} \frac{\lambda_i^{n-1} \lambda_j^{n-1}}{4^n} \int_0^1 \int_0^1 x_i(u_{kn}, v_{kn}, k)_{uu} x_j(u_{kn}, v_{kn}, k)_{uu} du_{kn} dv_{kn}.$$

Note that the integrand does no longer depend on n. Now, the chain rule is applied in order to obtain a differentiation with respect to u_{kn}. For the sake of simplicity, the indices are dropped after that:

$$e_{ij,uu} = \sum_{n=1}^{\infty} 4^n \lambda_i^{n-1} \lambda_j^{n-1} \sum_{k=1,2,3} \int_0^1 \int_0^1 x_i(u,v,k)_{uu} x_j(u,v,k)_{uu} du dv$$

By expanding the bicubic spline $x_i(u,v,k)$ into its control coefficients c_{il} and basis functions $M_l(u,v)$, and expanding $x_j(u,v,k)$ into d_{jm} and $M_m(u,v); l,m = 0 \ldots 15$, we obtain

$$\begin{aligned} e_{ij,uu} = \sum_{n=1}^{\infty} 4^n \lambda_i^{n-1} \lambda_j^{n-1} \sum_{k=1,2,3} \sum_{l=0}^{15} \sum_{m=0}^{15} c_{ikl} d_{jkm} \\ \int_0^1 \int_0^1 M_l(u,v)_{uu} M_m(u,v)_{uu} du dv. \end{aligned}$$

According to eq. (5.43), this can be reduced to the univariate case:

$$\begin{aligned} e_{ij,uu} = \sum_{n=0}^{\infty} 4(4\lambda_i\lambda_j)^n \sum_{k=1,2,3} \sum_{l=0}^{15} \sum_{m=0}^{15} c_{ikl} d_{jkm} \\ \left. \int_0^1 N_{l/4}(v) N_{m/4}(v) dv \int_0^1 N_{l\%4}(u)_{uu} N_{m\%4}(u)_{uu} du \right\} = a_{lm,uu} \end{aligned}$$

The $e_{ij,uu}$ exist if the geometric series converges. Therefore,

$$e_{ij,uu} = \frac{4}{1 - 4\lambda_i\lambda_j} \sum_{k=1,2,3} \sum_{l=0}^{15} \sum_{m=0}^{15} c_{ikl} d_{jkm} a_{lm,uu} \qquad \text{if } |4\lambda_i\lambda_j| < 1.$$

The terms $e_{ij,vv}$ and $e_{ij,uv}$ are obtained simply be replacing $a_{lm,uu}$ with $a_{lm,vv}$, or with $a_{lm,uv}$, respectively. Similarly,

$$e_{ij,u} = \frac{1}{1-\lambda_i\lambda_j} \sum_{k=1,2,3} \sum_{l=0}^{15} \sum_{m=0}^{15} c_{ikl} d_{jkm} a_{lm,u} \qquad \text{if } |\lambda_i\lambda_j| < 1,$$

analogously for $e_{ij,v}$. Additionally, a term $e_{ij,0}$ arising from non-derivated basis functions might be needed, e.g. for the *region-to-region* attractor to create initial surface shapes described in section 5.3.4:

$$e_{ij,0} = \frac{1}{4-\lambda_i\lambda_j} \sum_{k=1,2,3} \sum_{l=0}^{15} \sum_{m=0}^{15} c_{ikl} d_{jkm} a_{lm,0} \qquad \text{if } \left|\frac{\lambda_i\lambda_j}{4}\right| < 1.$$

Here, $a_{lm,0}$ represents integrals over products of non-derivated basis functions, similar to eq. (5.43).

The drawback of this solution is that there are several such terms that are undefined because the geometric series does not converge. Those terms are set to 0 in the energy matrices $\bar{\mathbf{E}}_{ext} = (e_{ij})$ of the patch.

Remember that the energy is expressed in terms of control points $\mathbf{p}_i$ projected into the eigenspace of the subdivision matrix of the Catmull-Clark scheme. Unfortunately, this representation of the energy, written in terms of *projected* control points, is not very useful, since the control points of a patch around an extraordinary vertex are shared by other patches, including ordinary bicubic B-spline patches. Since a common concatenation vector is needed to set up the minimization system (5.3), it would not be possible to assemble an energy matrix for the whole surface, using just the matrix $\bar{\mathbf{E}}_{ext}$. A representation of the energy needs to be found that can be expressed in terms of the original control points. This can be achieved by transforming back the $\mathbf{p}_i$, using the inverse of the matrix of eigenvectors $\bar{\mathbf{V}}$ known from equation (3.7), and combining this with $\bar{\mathbf{E}}_{ext}$:

$$\bar{\mathbf{A}}_{ext} = (\bar{\mathbf{V}}^{-1})^T \bar{\mathbf{E}}_{ext} \bar{\mathbf{V}}^{-1} \tag{5.46}$$

allows to write the energy of a patch around an extraordinary vertex in terms of the original control points $\mathbf{d}_i$:

$$E_{ext} = \sum_{i=1}^{K} \sum_{j=1}^{K} a_{ij,ext} \mathbf{d}_i^T \mathbf{d}_j. \tag{5.47}$$

Using the concatenation vector $\mathbf{c} = (d_{11}, d_{12}, d_{13}, \ldots, d_{K1}, d_{K2}, d_{K3})^T$, and expanding $\bar{\mathbf{E}}_{ext}$ to $\mathbf{E}_{ext}$ and $\bar{\mathbf{V}}^{-1}$ to $\mathbf{V}^{-1}$, as in (5.22), E_{ext} can be rewritten in matrix form as

$$E_{ext} = \mathbf{c}^T (\mathbf{V}^{-1})^T \mathbf{E}_{ext} \mathbf{V}^{-1} \mathbf{c}.$$

In fact, now that the energy contributions of extraordinary patches can be exactly calculated without iteration, it is possible to handle them in the same way as the energy contributions from ordinary B-spline patches. Moreover, $\bar{\mathbf{A}}_{ext}$ needs to be computed only once for a given valency and can be read from a file on startup of the application. The necessary control coefficients of the splines $x(u, v, k)$ are available from [Stam99b] as well as the elements of $\mathbf{V}^{-1}$ and the eigenvalues λ.

Note that, at a time when the basis functions were unknown, Halstead et al. [HKD93] derived energy contributions of these Catmull-Clark patches using a method similar to that Stam [Stam98] used to find his basis functions, see section 3.2.4.

Chapter 6

The User Interface

The spatial and temporal resolutions of display systems used for current VE systems, including projectors and screens, have reached a satisfying level that meets user requirements. However, computing and graphics power have grown much faster. Apparently, there is a strong discrepancy between the offered quality of those output components of the user interface, and the usability of current VE-based interaction methods. They are still too underdeveloped to support creative, highly interactive applications, such as geometric modeling or industrial design.

This chapter proposes solutions for input components of VE-based user interfaces, especially with regard to system control for geometric modeling applications. 3D interaction techniques have been developed and implemented for the modeler described in this work, and are presented in that context. Clearly, the usage of the modeling tools presented in previous chapters is an issue of 3D interaction as well. In this chapter, methods of how to handle and access that functionality and how to control the application as a whole are determined.

The concepts proposed in this chapter are independent of how particular modeling tools work. Moreover, they are general in the sense that they are not necessarily coupled to the specific modeling approach pursued in this work, and are applicable even to other fields.

6.1 Introduction to 3D interaction

Foley and Silbert [FS89] define the *human-computer interface* (or *user interface*) as the determination of all user inputs into a computer, the determination of all computer outputs to the user, and the determination of sequences of the inputs-outputs made accessible to the user.

A user interface enables *interaction* with a computer program through its hardware and software components for input and output. *Three-dimensional* or *3D interaction*, also referred to as *spatial interaction* means that the raw data that form the user input is generated in space or that the computer output is sent to a 3D environment. In turn, the user perceives the feedback information in this environment.

Several *interaction modes* (section 6.1.1) can be made available in a VE to perform the different *interaction tasks* (section 6.1.2) of an application. The interaction modes and the interaction tasks are described independently from each other. Note that often a choice of interaction modes is available to perform a specific interaction task.

6.1.1 Interaction modes

In a VE, other modes or methods of interaction than in a desktop environment are typically used. A classification of interaction modes is given by Bullinger et al. [BBB97], who defines *formal language interaction*, *direct manipulative interaction*, *natural language interaction*, and *gestic interaction*.

In [BKJ+00], Kruijff [Krui00] gives a similar classification of interaction methods. He describes the four categories: *graphical menus*, *voice commands*, *gestural interaction*, and *tools*.

The interaction modes described in this work are partly based on the work of Bullinger et al., and also partly on [BKJ+00]. The context of geometric modeling requires a generalization of some of the modes. Moreover, taking into account recent work on 3D interaction, it seems to be useful to include new modes, e.g. *prop-based interaction*.

Although the following modes are defined in the context of VEs, some of them do not necessarily require that a VE is present, and principally could be applied in other work environments as well.

Formal language interaction

Formal language interaction is based on programming languages, command languages, or formal interrogation languages [BBB97]. The complex structure of the input can only be handled using a keyboard, and a computer screen, and therefore does not correspond with user interaction in a VE.

Note that higher levels of interaction modes can partly rely on formal language interaction. For example, graphical menus, widgets, or gesture recognition engines execute scripts written in the underlying formal language, as in the VE framework "Avango" [Tram99]. It uses a Scheme interpreter as a frontend available at runtime. The state of all objects in the scene graph is completely defined by a set of attributes that are directly accessible through Scheme bindings. This makes the formal language layer very powerful, although the full functionality is rarely needed during interactive sessions.

Voice-based interaction

Voice-based interaction is realized by interpreting spoken words as commands. It can be very powerful, because it is hands-free and natural [Krui00]. In complex applications, it offers an additional channel used parallel to the main interaction mode. Relying solely on voice-based interaction is not recommended, since it cannot be used in noisy environments. Furthermore, the speech recognition engines are not always reliable.

In [BBB97], *natural language interaction* is discussed. Although natural language interaction potentially offers possibilities of expressions not realizable by other forms of interaction, it is practically restricted to be based on simple phrases or single words. Therefore, it seems to be more appropriate to use the term voice-based interaction instead.

Direct manual interaction

Direct manual interaction plays the most important role in VEs, especially in interactive applications such as geometric modeling. It shall be defined by the following criteria:

– User input is formed from the positional and orientation data obtained by hand tracking. It is mapped to manipulations or transformations of virtual

objects, or state-changing functions defined on the application, or results in the creation of new objects.

– The way in which the hands interact is based on familiar metaphors, or on artificial concepts also called *magic* that deviate from the real world [Poup00].

Earlier descriptions of this kind of interaction are centered around making use of familiar metaphors of daily life to manipulate objects, and only act on existing objects, such as the *direct manipulative interaction* mode defined in [BBB97]. Developers of 3D interfaces often favor familiar metaphors of daily life, although they are not always adequate or applicable in a VE. Furthermore, the absence of constraints of the physical world allows the development of artificial concepts for manual interaction. The advantages and disadvantages of this approach are discussed by Poupyrev [Poup00]. In section 6.4, spatial interaction techniques will be developed that do *not* adopt metaphors of the real world, although they are hand-based.

Usually, direct manual interaction is implemented by using the hand with a hand-held input device or with a glove[1]. Issuing a command or performing manipulation is initiated by bringing a virtual pointer that follows the hand close to an object. Eventually, the interaction procedure requires that additional commands be issued e.g. by pressing buttons.

Note that the direct manual interaction mode is not restricted to interaction tasks such as object manipulation or creation. It is also commonly used to perform *system control* tasks, e.g. when interacting with menus in a VE.

Gestural interaction

In the *gestic* [BBB97] or *gestural* [Krui00] *interaction* mode, commands, instructions, as well as manipulative or even creative actions are formed by hand movements or hand signals. In the simplest case, gestures express simple commands to change the system state. Used in this manner, gestural interaction can be compared with the usage of sign language [Krui00]. Complex gestural interaction is presented by Hummels et al. [HPO+97, HSO97], who describe how gestures could be used to create initial shapes, or deform predefined shapes.

Hand movements that are used to describe geometric attributes of primitive objects, e.g. pointing out the length of a cylinder or the radius of a sphere, or that are used for other simple tasks such as gripping objects, seem to fit better under direct manual interaction than under gestural interaction, although interpreted as gestic interaction in [BBB97]. Thus, the categorization of interaction modes is not very strict, and there is a continuous transition between different modes. The main difference between direct manual and gestural interaction is, with gestural interaction, the hand movements usually must match a certain 2D or 3D path in order to be recognized as a gesture. Principally, only a limited set of gestures is available. In contrast to this, in direct manual interaction, a hand movement directly acts on an object, and the path formed by the hand is arbitrary. Hence, gesture recognition engines are not necessary.

Related to gestural interaction is *postural interaction*. As pointed out by [Krui00], there is a significant difference between gestures and postures. A posture refers to how the hand is held, and is therefore static. A gesture means that the hand can move, thereby changing its posture. The pinch glove is an input device that supports postures formed by pinching, i.e. bringing the thumb together with a finger. The pinch glove has contacts at the fingertips that register pinching events and forward them to the application. This is used by Cutler et al. [CFH97] to switch between tools at the Responsive Workbench.

[1] Strictly speaking, the term "direct" therefore is not really correct.

Prop-based interaction

Prop-based interaction makes use of special-purpose input devices, referred to as props. Often, a physical prop resembles a certain class of virtual objects, to support a particular effective and intuitive interaction with that class of objects. For example, consider a plastic torso used together with a hand held device in a medical simulation application. Prop-based interaction was introduced by Hinckley et al. [HPGK94], who present a 3D interface for visualizing and interacting with neurosurgical data.

More generally speaking, the shape of the prop and its handles, switches, or buttons, are highly adapted to a specific interaction task. This way, the so-called Cubic Mouse [FPW+00], see Figure 1.6 on page 13, allows controlling cutting planes intuitively. The user moves three sticks that are associated with the local coordinate system of the object attached to the device.

Interaction using external devices

Separate output devices that are integrated in the VE system setup can be a suitable solution to 3D interaction tasks that are difficult to accomplish directly in the VE. Using a conventional computer at another workplace for controlling the VE is not the goal of this interaction mode. Instead, an additional display such as a pen-PC or a touch screen is located close to the user. This provides additional information and functionality that can be accessed by the hand or even by the same input device used within the VE. Complicated interaction procedures could be much easier configured using features of these devices compared to implementing them in the VE.

Furthermore, all interaction widgets that are part of the scene graph of a VE have one drawback in common. If the amount of geometry exceeds a critical limit, the usability of the interaction widgets will seriously suffer from a decreasing frame rate. This is especially true if performing the tasks requires fine motor skills. In such cases, external and independent devices benefit 3D interaction significantly.

Multimodal interaction

LaViola [LaVi00] defines *multimodal interaction* as the combination of multiple input modalities (i.e. interaction modes) to provide the user with a richer set of interactions compared to traditional unimodal interfaces. Interaction modes can be combined in various ways. They may complement each other, result in redundant input, the user can choose between equivalent modes, or can issue commands concurrently using different modes [LaVi00]. *Combined interaction* is similarly defined in [BBB97].

6.1.2 Interaction tasks

Most of the universal interaction tasks in a VE are of spatial nature. In [BKJ+00], *navigation, selection, manipulation,* and *system control* are identified as universal interaction tasks, used as building blocks to compose more complex tasks. This might apply to most standard VE application domains that have been established so far. Considering geometric modeling however, one essential interaction task, namely *creation*, should be added to that classification, see below.

Navigation

In an immersive VE, *navigation* is the most important task. Navigation means specifying or changing the current location of the observer relative to the virtual scene, so that the part of the scene that is contained in the viewing frustrum changes.

Navigation consists of two subtasks. *Wayfinding*, similar to the real world, is a cognitive task, which means acquiring knowledge about the current position and deciding the direction to go within the virtual world. *Travel* is an input task and refers to the actions necessary to move from place to place [BKJ+00].

In semi- or non-immersive VE systems, like the Responsive Workbench, navigation does not play an important role, since in most cases the complete scene can be easily overlooked. Note that transformations applied to the scene graph are not considered as navigation tasks, although it can have the same effect on the position of the user relative to the scene.

Selection

Selection, or *picking*, is simply the specification of objects within the VE. Selection is often used together with other interaction tasks, e.g. an object is selected in order to move it to another position. The implementation of selection techniques can be problematic, since it is not always possible to reach an object with the hand or with a virtual pointer. Objects may be too distant or overlap each other therefore special techniques need to be developed.

Manipulation

Manipulation means the change of object attributes. Some examples include the transformation matrix, the material, or shape parameters. Often, interaction techniques used for manipulation in VEs are based on familiar metaphors. Simple manipulations are performed directly; e.g. grabbing the object and releasing it at the target position. Interaction techniques for more complex tasks, such as sculpting the shape of objects, are not so straightforward. A possible approach is to develop so-called magic [Poup00] sculpting techniques. The use of the hands in such techniques deviates from their use in the real environment. For example, consider the curve shaping tools presented in section 5.2: curves are indirectly smoothed or sculpted by indicating where the tool is supposed to influence the curve.

Creation

For VE-based design applications, *creation* needs to be considered as a separate task dedicated to designing objects from scratch. Creation means instantiating a new object of a certain class and specifying its relevant attributes. The creation tasks of the modeler described in this work can be found in chapter 4.

In simple cases, where objects with reasonable initial attribute values are available, creation corresponds to the two subtasks instantiation and manipulation. Shape attributes that imply a large number of degrees of freedom, such as control point vectors, cannot be specified that way. Objects containing such attributes have to be created using special creation techniques. Compared to manipulation, fewer numbers of constraints are usually imposed upon creation.

System control

System control, also referred to as *application control*, is interaction to change the state of the system, the state of the application, or the mode of interaction, see also [BKJ+00]. Issuing corresponding commands to the system usually does this. System control tasks might involve other interaction tasks, such as selecting items and clicking on icons.

Unfortunately, the 2D interaction style of desktop systems cannot simply be transferred to a VE, since the extra third dimension involves many more degrees of freedoms compared to a 2D environment [Krui00].

6.1.3 Interaction at the Responsive Workbench

A particular VE system is usually designed to meet the requirements of a restricted class of applications. The resulting setup already implies its basic interaction methods, which have to be taken into account for developing user interfaces. In the context of this work, the Responsive Workbench (section 1.2.2) is the most important setup. It is therefore important to understand the particular characteristics of workbench interaction before developing the user interface components of the modeler.

The manipulation space

At the workbench, the part of the scene contained in the viewing frustrum, and the region that is within arm's reach, are largely superimposed. Additionally, the boundary areas of a workbench VE can easily be reached, since the user can move to the sides and move his head over the table. Furthermore, objects can have an appropriate scale, due to the size of the projection screens. This is particularly helpful for the direct manual interaction mode.

The hands are instrumented with simple, tracked input devices, such as the pen-like stylus, and control objects "directly", at close distance. Note, however, that normally the virtual cursors that represent the reference locations of interaction are slightly distant from the input devices. This is necessary to alleviate the disturbances of depth perception, caused by the hands and the input devices that visually interfere with the virtual scene. This is a drawback typical for back-projected environments.

Obviously, optimizing interaction methods based on the interplay of *both* hands is a very powerful approach for workbench interaction. Two-handed interaction will be examined in more detail in sections 6.2 and 6.3.

Use of the table top

Beside its main purpose, the tabletop has additional use.

The lack of resting positions for the arms is often regarded as a drawback factor that affects the usability of VE-based interfaces, because it exacerbates fatigue [MBS97]. Although workbench interaction suffers from this problem as well, the tabletop alleviates it by providing a resting position for the user's (non-active) hand and his body during interaction.

Furthermore, the border regions of the tabletop may serve as places to store occasionally used input devices; in particular, props. Beside direct manual interaction, prop-based interaction is currently the most common input mode used at the workbench. Using several props or changing the mode of interaction requires laying aside the old input device and picking up the new one from a place that should be within arm's reach.

It is interesting to note that Cutler et al. [CFH97] interpret the Responsive Workbench itself as a large physical prop, to which the model is anchored. The fact that the user has haptic contact with it should be further exploited and may very well lead to the development of appropriate interaction methods.

Predominant modes and tasks

The use of the workbench metaphor meets the needs of manually oriented applications, in which the hands directly manipulate virtual objects. This is the main mode of interaction at the workbench, which is, however, useful in areas other than for manipulation tasks as well.

In summary, the common spatial interaction tasks of workbench applications are selection, manipulation, creation, and system control. These are accomplished mainly in direct manual mode, and in prop-based mode. Navigation, a very common task in immersive VEs, is not essentially needed in workbench applications. This is because the dimensions of typical application scenarios normally fit within the given manipulation space, i.e. the Responsive Workbench is mainly used as a non-immersive environment.

One task particularly important in interactive applications is system control. Opposed to manipulation, or creation, for which metaphors from the physical world are often available, useful approaches to system control in workbench environments seem to be hard to find with the available interaction modes. Integrating system control into highly interactive tasks on the workbench as they appear in the modeler is the topic of section 6.4.

6.2 Two-handed interaction

User interfaces that accept simultaneous input from both hands have become very popular, especially in VEs. This kind of interaction is referred to as *two-handed* or *bimanual* interaction. Its principles are derived from how humans use their hands in reality, as explained in section 6.2.1. Direct manual interaction based on two-handed input is a powerful concept for virtual work environments, such as the Responsive Workbench. A general purpose user interface by Cutler et al. [CFH97] provides the basic ideas for bimanual workbench interaction. It is described in section 6.2.2.

6.2.1 Principles of two-handed interaction

The two-handed interaction schemes proposed so far for table-like environments are largely based upon the observations of Guiard [Guia87] of how humans distribute work between their hands in the real world. Although the work of Guiard examines real world situations, the results could even be used to derive artificial or magic concepts for direct manual interaction.

Guiard classifies manual activities into three categories; namely *unimanual*, *symmetric bimanual*, and *asymmetric bimanual* tasks. Unimanual tasks involve only one hand, e.g. opening or closing a water tap. Symmetric bimanual activities involve both hands performing a nearly identical action, e.g. driving a car, both hands holding the steering wheel. The most common activities turn out to be asymmetric bimanual tasks. The so-called non-dominant hand (for most individuals, the left hand) and the dominant hand (i.e. the right hand) have different, but coordinated roles. Consider handwork, or playing music instruments; these activities are characterized by asymmetric bimanual work. This kind of coordinated movement of both hands is based on the following principles:

- The non-dominant hand dynamically provides a spatial reference for the movements of the dominant hand.
- The dominant hand performs finer, more detailed motions at a higher spatial and temporal frequency, compared to the non-dominant hand.
- The non-dominant hand starts earlier; initiating an activity.

The framework of Guiard represents a guideline for the design of two-handed interfaces for desktop systems as well as for VEs. It has been shown by several studies that two-handed interaction minimizes the cognitive load, and enhances performance, see e.g. [KBS94], or, in the context of geometric modeling with desktop systems, refer to the work of Gribnau [Grib99].

6.2.2 Two-handed interaction at the Responsive Workbench

Cutler et al. [CFH97] propose a variety of two-handed 3D tools and interactive techniques for the Responsive Workbench, which are based upon the principles of Guiard. Their tools, described in the following table, are exceptional intuitive. Although all the tools are restricted to simple transformations of objects, the approach of Cutler et al. seems to be a very promising approach to workbench interaction. Similarly, the principles of Guiard are useful to derive methods for more complex manipulation and creation tasks: a two-handed interaction scheme for conceptual styling at the Responsive Workbench is presented in the next section.

One-handed	
Adjust scalar attributes	Scalar attributes of objects, such as the transparency, can be adjusted by turning the stylus around its axis, like turning a screwdriver.
Grab	Objects can be grabbed and moved freely, or their movement can be constrained to the tabletop.
Scale	Objects are scaled by moving the hand up or down, increasing, or decreasing the size, respectively.
Symmetric two-handed	
Scale	The size of objects is controlled by indicating it with both hands. The user moves the hands close to each other, or apart from each other.
Rotate	The object is rotated with both hands, like a turntable.
Asymmetric two-handed	
Rotate	The non-dominant hand positions the object while the dominant hand rotates it around its center, in a trackball-like manner.
Zoom	The non-dominant hand specifies a focal point, whereas the dominant hand moves away from it (zoom in) or towards it (zoom out).

6.3 Creation and manipulation

In this section, a non-symmetric two-handed interaction scheme of the modeler is presented. It is distinguished between actions of the non-dominant hand, the dominant hand, and two-handed actions. The focus is on the way the tools are used by the hands, whereas interaction methods for tool selection deserve especial attention in section 6.4.

The idea to utilize two-handed interaction for geometric modeling has been proposed before for desktop systems. In these applications, the hands are used according to the observations of Guiard. The non-dominant hand is responsible for setting the reference frame for the dominant hand, which edits the model. The THRED system [GLS95, SG97] is a free-form editor of this kind, which is based on polygonal surfaces. In 3-Draw [SRS91], the non-dominant hand adjusts a real

tablet, to which a tracking sensor is attached, while the dominant hand draws curves on the tablet (see also section 2.5).

6.3.1 Benefits of two-handed manipulation

Recently, Gribnau [Grib99] has examined two-handed operation for computer-aided conceptual modeling, using a standard computer screen together with spatial input. He conducted user studies where one-handed and two-handed manipulation tasks had to be performed. It was found that conceptual modeling would benefit from two-handed operation in 3D. Two-handed operation is faster because the designer does not have to divide the intended actions into successive 2D movements with the mouse [Grib99].

The studies of Gribnau covered only manipulation tasks. Creation was not considered. However, it seems to make sense to assume that creation tasks would benefit from two-handed support as well.

The experimental setup consisted of a computer screen running in mono mode, and input devices that were electromagnetically tracked. Similar to the other two-handed desktop-based modelers, an unnatural discrepancy between the limited display size and the space available for hand movement results. The Responsive Workbench provides a more adequate workspace for such kind of hand-based interaction. It is therefore assumed that the use of a workbench would benefit conceptual styling with two hands, compared to the use of a computer screen. This is currently not manifested by evaluative user studies.

6.3.2 A two-handed interaction scheme

Manipulation and creation tasks of the modeler are supported by a two-handed interaction scheme. According to the principles of two-handed interaction, aligning and positioning tasks are assigned to the non-dominant hand, whereas creation and manipulation tasks are assigned to the dominant hand. In addition to that, combined two-handed tasks exist.

Input devices

For the proposed interaction methods, simple input devices that generate 6-DOF tracking data and have at least one button are sufficient. The stylus, described in section 1.2.3, is a standard input device that resembles a pen and is equipped with one button. It is used by the dominant hand for creation and manipulation tasks, such as drawing and deforming curves. The non-dominant hand uses a device that has a shape similar to a mouse and is equipped with three buttons. At least one button is required.

Tools

A selectable item representing a function or an interaction task defined on an object class in the VE is referred to as a *virtual tool*, or shortly, as a *tool*. Usually, a tool has geometric representations, such as an icon to select it, and a hand held pointer, when it is active. A tool transfers hand motion into modifications of attributes of virtual objects, or creates new objects.

Virtual cursors

Usually, a virtual cursor or another piece of geometry is assigned to each tool. The cursor is activated when the corresponding tool is selected and follows the stylus. All interaction with virtual objects is based on evaluating the position and the

direction of the virtual cursor relative to an object. Depending on the tasks, the cursor must intersect the object, must be close to it, or has to point towards it. Therefore, the cursor looks like an arrow in order to visualize its position and also the direction into which the user is pointing.

Feedback

Activating tools, selecting objects, and applying tools to objects is supported by optical feedback. Usually, the involved objects or cursors are highlighted.

Tasks of the dominant hand

The following tasks, which are assigned to the dominant hand, can be interpreted as subtasks of creating surfaces. In some modelers, the creation of surfaces is a direct task, as in "Surface Drawing" by Schkolne et al. [SS99]. As it has been described previously (see section 4.2), the modeler of this work tries to simplify the main surface creation task by dividing it into several successive tasks that are applied to curves.

Draw a curve A direct 3D drawing stroke, accomplished with the dominant hand, yields a space curve, as described in section 4.3. The data points of a drawing stroke can be projected before interpolating a curve, resulting in a planar curve (see also section 4.3.2), or in a curve on the surface (see also section 4.3.3).

Drawing a curve is the most important creation task of the modeler. Every new design that starts from scratch requires that some curves representing parts of the spatial contour of the model are directly drawn. However, not all curves need to be directly created. Copying or mirroring them can reuse curves, as the model is evolving. In chapter 7, this process is described in detail by means of examples.

Draw a symmetric curve By selecting the tool for drawing symmetric curves, a symmetry constraint is imposed upon the drawing stroke. It is applied to the curve after the user has generated all data points. The given plane of symmetry should be perpendicular to the "main" drawing direction; which is from left to right, to produce useful results.

An obvious approach would be drawing just one half of the curve and generating the other half automatically. However, this method seems to be too "technically involved". In addition, it is easier to form a network of curves with full drawing strokes. The curve is made symmetric by averaging both halves afterwards.

Deform a curve or surface Deformation of curves and surfaces is accomplished by transforming the movements of the dominant hand into a change of the shape parameters, i.e. the control points. This is achieved using variational methods, which have been extended in this work for interactive usage, as described in chapter 5. The deformation tools for smoothing and sharpening curves are designed specifically for being used without force-feedback, as they are based on indirect, but intuitive movements of the hand, see section 5.2.

Select a loop A surface part is fit into a closed sequence of curve pieces by pointing directly into the loop with the stylus, as described in section 4.5.1. The found loop is highlighted and can be selected by pressing the button. This initiates the automatic calculation of a surface part.

Select pieces of a loop Pointing directly at its individual curve pieces, and clicking the stylus button specifies a loop. The selected curve pieces are highlighted. Subsequent pieces will be selectable only if they are connected to previously selected pieces. The selection of the closing curve piece initiates the calculation of the surface part that belongs to the loop.

Basic functions Basic functions, such as copy, mirror, move, or remove are implemented for curves. For surfaces, the only basic function is the remove function. Since the model is specified through the curve network this is possible. The dominant hand performs these tasks in the following way:

Copy a curve Bringing the cursor close to the curve and pressing the button select the relevant curve. A copy of the curve is attached to the hand, following it as long as the button is held pressed. When the button is released, the copied curve is inserted into the curve network at its current location using the algorithm described in section 4.4.1. All those loops that share a network curve intersected by the newly inserted curve become invalid. The corresponding connections are reset, and the associated surface parts are removed. The user has to define new loops in that region of the curve network.

Move a curve Moving a curve works in the same way, except that the selected curve is first removed from its initial position in the curve network. This means resetting the connections pointing to the curve and those pointing from the curve to its neighbors and to the loops. If surface parts share pieces of the selected curve, they are removed as well.

Mirror a curve To mirror a curve, the user selects the reference curve and clicks the stylus button. A copy of that curve that is symmetric with respect to a user-defined plane is created. It is inserted into the network automatically.

Remove a curve Similarly, curves are removed by clicking at them.

Select a tool For the purpose of selecting modeling tools, 3D widgets are available, which are handled by the dominant hand. The main reason for using the dominant hand is that orienting the model with the non-dominant should be continuously possible. The interaction widgets for tool selection are described in sections 6.4.2 and 6.4.3.

Drop the tool The current tool is automatically dropped when another tool is selected. In order to drop the tool without selecting another tool, the stylus is moved away from the workbench and the button is pressed. The half-space in front of the workbench is defined as the tool drop area. An alternative tool drop method that was suggested by Mine et al. [MBS97] is "throwing it over the shoulder" and pressing the button.

Tasks of the non-dominant hand

Position the model Positioning and orienting the model is the main task of the non-dominant hand. Orienting relevant parts of the model toward the user is supported by a clutch mechanism. Pressing the corresponding button on the input device attaches the model to the hand. The model follows the hand as long as the button is held pressed down. Releasing the button fixes the model at its current location.

Turning the model all the way around can be achieved by subsequently clutching the model to the hand. Notice that, although that method is based on the grabbing metaphor, the situation in the real world is completely different. The physical part would fall down immediately after releasing the grip. The fact that the virtual model stays at its position, which certainly is very helpful, is an example that implementing real-world behavior in a VE is not always desirable.

Position the model relative to a virtual drawing plane is a task needed for drawing planar curves. The virtual plane is activated when the tool for planar drawing is selected. The plane is inclined at an angle of 45° relative to the horizontal projection screen, resembling a drafting table for architects and designers. The virtual drawing plane always faces to the front of the workbench. This method allows the user to perform drawing strokes in the main drawing direction, parallel to the front. Therefore, the model has to be oriented relative to the drawing plane for appropriate placement of the new curves. The drawing plane being semi-transparent, allows the model to be aligned to it in an easier manner.

Scale the model Scaling the model appropriately is very useful for the insertion of small, short curves, or for the creation of curves within oversized parts of the model. The whole model is scaled until the size of the relevant parts is adapted to the size of usual drawing strokes.

The hand gesture used to scale up or down the model is moving the hand up or down, respectively, thereby pressing the second button. This metaphor has been developed and implemented by Fröhlich et al. [CFH97], and it allows to handle scaling very intuitively. If there is not a second button available, a scale tool must be activated first.

Drop the tool If there is a second button available on the device for the non-dominant hand, a tool that is attached to the editing hand can be dropped by pressing that button. The advantage of this method is that the user can switch between tools very quickly, compared to dropping the tool by moving away the arm. This can benefit the performance of complex tasks that require several tools.

Combined two-handed tasks

Combined two-handed tasks are tasks that are simultaneously performed with both hands. Principally, all tasks for the dominant hand could be used in a mode in which the non-dominant hand dynamically adjusts the model in order to achieve a continuously optimal perspective. However, this method is not useful for all tasks. For example, it seems to be very difficult to draw a space curve if the other hand is busy moving the model. Despite this, some combined tasks benefit from combining movements of both arms.

Drag a curve or surface The result of deforming a curve or a surface with the drag tool depends on the relative movements of both hands. In other words, the editing hand, which controls the deformation, could adopt a fixed position while the non-dominant hand moves away the entire model. This would thereby result in the deformation of the selected object.

Opposed to this, deforming a curve with the smooth tool, or with the sharpening tool (see section 5.2.3 and 5.2.4) requires finer hand movements, which are better suited to the dominant hand. Therefore in cases like these, the model should be kept fixed.

Copy or move a curve A frequently needed task is copying or moving a curve to a new location with the dominant hand. Simultaneously moving or turning the model with the non-dominant hand, until the new location is within reach can alleviate this.

Note that the model can be attached to the non-dominant hand simply by pressing the corresponding button, which activates the clutch mechanism. There is no need to explicitly select the model. On the other hand, a curve must be located and selected with the cursor each time a new grab sequence starts. If the new position of the curve is difficult to reach, it will be easier to turn and reposition the model rather than repeatedly grabbing the curve.

Select pieces of a loop Selecting all curve pieces that form a loop is a task that can require repeated adjustments of the model position. Repeatedly attaching the model to the non-dominant hand and aligning it appropriately, as described for the task of moving curves can achieve this. The process of extracting the topology of a curve network seems to benefit from two-handed interaction in VEs because the non-dominant hand can quickly provide a new reference frame for the dominant hand. Compare this to the situation in a desktop environment, which is discussed in section 4.5.1.

6.3.3 Summary

The tasks of the modeler referenced in this work are summarized in a table on page 95. As it can be seen, the corresponding tools available for modeling are numerous. Solutions to the problem of how to access this functionality in a comfortable and appropriate way are presented in the next section.

Dominant hand	
Draw a curve	A cubic interpolatory spline is created using the tracked data points. The curve can be a space curve, or a planar curve, or can be constrained to lie on the surface.
Draw a symmetric curve	The curve is made symmetric with respect to a given plane after the full drawing stroke is completed.
Deform a curve	A curve is locally *smoothed* or *sharpened* by moving the cursor back and forth along the curve. A curve is directly deformed by *dragging* it into the direction of the cursor.
Deform a surface	Surface parts are globally *smoothed* or *sharpened* as long as the button is held pressed. Surfaces are directly deformed by *dragging*.
Select a loop	To fit in a surface part, loops are pointed to with the cursor.
Select pieces of a loop	Loops are specified by selecting the curve pieces that form the loop.
Basic functions	Curves are copied or moved by selecting them and moving them, holding down the button. They are mirrored or deleted by selecting them and pressing the button.
Select a tool	Tools are selected using the hand menu, or the ToolFinger, refer to section 6.4.
Drop the tool	Tools are dropped by putting them aside, or by throwing them over the shoulder, pressing the button.
Non-dominant hand	
Position the model	The model is following the hand as long as the button is held pressed.
Scale the model	The model is scaled by moving the hand up or down, holding the second button pressed.
Drop the tool	Another possibility to drop a tool is pressing the third button.
Two-handed	
Drag a curve or surface	A curve or a surface is deformed based on the relative movement of both hands.
Copy or move a curve	Similarly, a curve is copied or moved, using both hands.
Select pieces of a loop	The curve pieces of a loop are selected with the stylus, after the model has been adjusted so that the curve pieces can easily be reached.

6.4 System control

The interaction techniques developed for intuitive, direct manipulation of geometry and data in VEs in many cases use metaphors from the physical world. The arguments for the suitability of VEs, such as the Responsive Workbench, for direct manipulation are largely based on this way of designing interaction.

From 2D desktop interaction, it can be seen that the user is involved in many overhead tasks, such as choosing a function, selecting an object, or issuing a command. For instance, consider a computer-aided design system, where the user wants to sculpt a free-form object. In order to perform this task, editing tools not only have to be applied to the model, they also have to be *selected*, or *switched*. Therefore, aside from the main task, the user frequently performs actions in which a command is applied to change either the mode of interaction or the application state. Such actions are defined as *system control*, or *application control* [Krui00], and they are always part of an application.

If you consider complex manipulation tasks in a VE within this context, it becomes clear that appropriate 3D interaction techniques for system control must be developed and integrated into the user interface. This must be done in order to fully exploit the potentials of virtual work environments for direct manual interaction. Contrary to manipulation, solutions for system control methods in VEs are far from being obvious. On a 2D desktop system, applications are normally controlled via a WIMP interface (Windows, Icons, Menus, and Pointers). It is commonly accepted as the standard interaction method. Consequently, the related concepts unaltered appear in many VE applications as well. However, inherently two-dimensional concepts cannot simply be transferred to VEs, since the extra third dimension involves many more degrees of freedoms, compared to a 2D environment [Krui00].

In this section, first, a summary on related work is given, and after that, two system control techniques are presented that have been developed for the use at the Responsive Workbench with the modeler. The *hand menu* (section 6.4.2) tries to utilize the traditional menu approach for interactive applications in VEs. The *ToolFinger* (section 6.4.3) is a novel approach that especially benefits the task of selecting tools, including switching between tools.

6.4.1 Related work

Although the system control components of the user interface crucially influence the usability of VE applications, specifically designed techniques are rare, even for table-like work environments.

Bowman et al. [BKJ+00] give a categorization about the existing methods, which include graphical menus, voice commands, gestures, and gestural interaction. For the purpose of system control, including tool selection, a few non-traditional approaches have been proposed.

Coquillart and Grosjean [GC01] propose a promising approach to system control. They transfer the principle of keyboard hotkeys to the Responsive Workbench, using a cube-shaped widget, the *Command & Control Cube* or *CCC*. It consists of a 3D grid of small cubes with which commands are associated. Positioning a selection pointer within the corresponding cube, using a tracked input device, can activate a command. The authors emphasize that this technique allows controlling the application even in "eyes-off" mode, due to its regular structure. The control cube is the first approach that introduces a shortcut paradigm to VE interaction.

The virtual palette [CW99], and a similar approach by Schmalstieg et al. [SES99], can be used for two-handed application control. The virtual palette resembles a transparent plate with a handle, held by the non-dominant hand. It is used together with a stylus in projection-based environments. With that configuration,

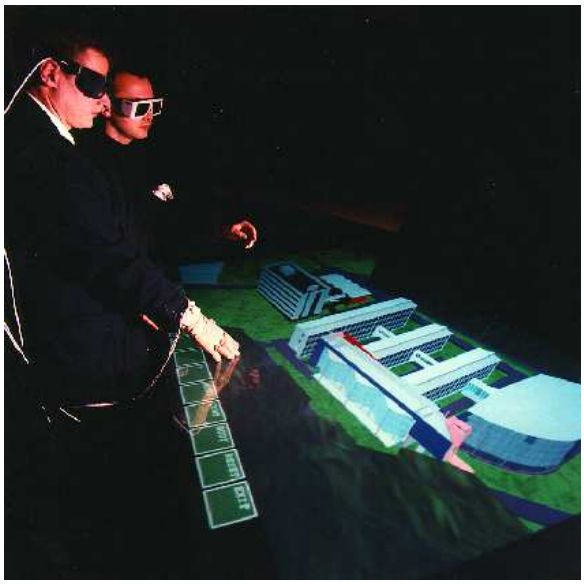

Figure 6.1: The use of a menu at the Responsive Workbench. The fields are located in the front of the display area. They are selected by moving the stylus inside. The architectural scenario is described in [KBF+95].

items can be selected on the palette using the stylus.

A pinch glove based menu for system control tasks is presented in [BW01], implemented in an immersive virtual environment. The menu texts appear close to each finger, oriented in the direction of the finger. Touching a finger with the thumb causes a tool selection.

Cutler et al. [CFH97], perform two-handed direct manipulation in a natural way on a Responsive Workbench, see also section 6.2.2. Both hands are instrumented with pinch gloves. Tool transitions are accomplished by pinching, or by picking up tools from a toolbox, located at the border of the display. In certain situations, transitions occur implicitly by reaching in with the second hand. As a result this causes switching from one-handed to two-handed mode.

Forsberg et al. [FJZ98], a modeling framework running on a table-sized display, called ErgoDesk, is presented. The interaction paradigm is based on physical props and multimodal input. Transitions between a variety of 2D and 3D interaction techniques are supported. To switch a prop, the user puts down one prop and picks up another. Transitions between virtual tools are accomplished either by speech recognition or by a drawn gesture.

6.4.2 The hand menu

The hand menu has been developed as a general-purpose technique for system control at the Responsive Workbench. It is based on the familiar concept of a menu. However, its use deviates considerably from 2D menus and from early menu techniques proposed for workbench environments.

The use of menus at the Responsive Workbench

The use of menus is familiar from 2D interaction, where items are placed on a screen and selected by positioning a cursor inside, using a mouse, and pressing mouse buttons. The first menus that appeared on the Responsive Workbench were used

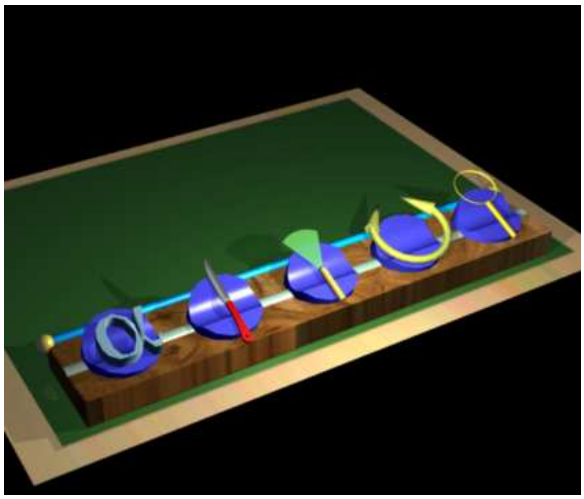

Figure 6.2: The design of the toolbar for the Responsive Workbench (due to B. Fröhlich and S. Mostafawy). A tool is selected by intersecting the icon with the stylus.

analogously. Two-dimensional items, resembling keys on a computer keyboard were placed on the display surface [KBF+95], as shown in Figure 6.1. Later, Fröhlich chose three-dimensional icons as selectable items [CFH97], which were arranged next to each other, forming a toolbar that appears to be above the front part of the workbench, as shown in Figure 6.2. The intent of the toolbar design is to resemble a set of physical tools laid out ready for grabbing.

With both menu variants the menu is placed at a fixed position on the display surface. Selections have to be made by positioning the stylus inside the selection region. Selecting a rectangular field belonging to a menu of the first type is facilitated by haptic contact. This is because the user touches the physical display surface with the stylus. On the other hand, to select a tool from the menu of the toolbar type, the user has to hit the corresponding icon geometry in three dimensions. Other pick policies, e.g. checking the bounding sphere of an icon, are also supported.

The *hand menu* proposed here differs from these early attempts in the following way. Instead of being fixed on the display area or within the 3D scene, the menu follows the hand of the user. Turning the hand and pressing the stylus button does a selection, thus there is no need to hit icons in space.

This approach is based on the following observations on workbench interaction. At the Responsive Workbench, the location where the user is performing the main task is in most situations away from the position of a fixed menu. Repeatedly reaching for menu items with the hand to switch tools can become a burden for the user. Moreover, the user is required to break his focus of attention in order to look at the menu. Another drawback of fixed menus is that they can occlude relevant parts of the scene, or that the menu itself is occluded. Mine et al. [MBS97] argue for specifying the position of a menu relative to the user or relative to a part of the body. It was found that the resulting proprioceptive cues, i.e. the person's sense of position and orientation of the body and the limbs, could significantly enhance user performance, compared to fixed widget positions. The hand menu concept follows these guidelines.

Similar methods have been presented for use in a desktop environment. Liang et al. [LG94, SG94] propose a ring menu that is following the hand that is holding a tracking sensor. The menu is represented as a circular object, on which several icons are placed. To select a tool the user can rotate the hand, until the desired icon

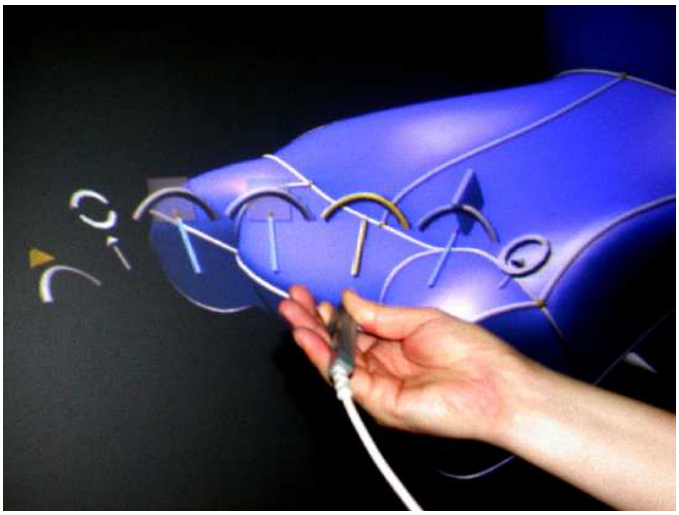

Figure 6.3: The hand menu interaction widget

falls into a selection basket, which is always oriented toward the screen. Although the ring menu was designed for use in a 2D system, its concepts provide a suitable basis for interaction with menus in VEs. Furthermore, concerning the problem of interference of interaction widgets with the model, VEs are superior for interaction with hand-held menus. Compared to standard computer screens, the much larger workspace of projection-based systems allows more flexibility with menu positioning. Moving the menu with the dominant hand, or aligning the model with the non-dominant hand is possible.

Design

The hand menu consists of a curved configuration of icons, each of which represents a tool, see Figure 6.3. The angle spanned by the menu should allow a comfortable turning of the wrist; otherwise it would be difficult for the user to select the outer icons. The radius of the curved menu should equal the distance of the wrist to the menu, so that the appearance of the menu suggests its usage.

Given a number of tools, the designer of the menu should specify that distance, together with the size of the icons, in such a way that a comfortable selection is possible. When adding more icons to an existing menu, the designer has two options. The size of the icons could be reduced, or the distance of the menu to the wrist could be enlarged. Reducing the icon size seems to be the preferable method, since the icons need not be hit explicitly for selection. Obviously, the maximum number of tools that can be handled in a reasonable way is limited; at most 10–12 tools seem to be appropriate.

Notice that the icons are slightly tilted upwards, taking account for the relative position of the user's head to the scene at the Responsive Workbench.

Usage

The hand menu is optimized for quick selections of items performed under the control of just the dominant hand. The non-dominant hand is still free to align the model.

After activation, the menu follows the dominant hand, so that in case the menu is occluded by another object, it can be dragged out easily. At the same time, the menu is attached to the hand in a way that allows the hand a simultaneous tool

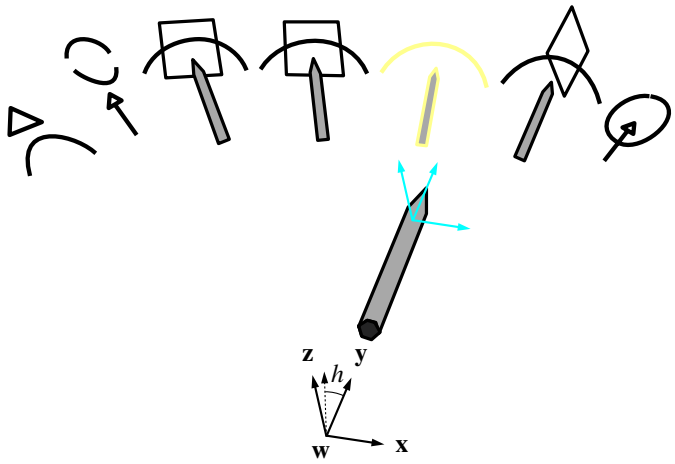

Figure 6.4: Handling the menu. The blue coordinate system of the stylus receiver is shifted towards the position of the wrist **w**. The menu is effectively attached to the coordinate system $\{\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{w}\}$. Thereby the rotation of the menu around the **z**-axis h, which is needed to select a tool, is frozen to 0, as well as its rotation around the y-axis, in order to keep the menu horizontal. With the shown direction of the stylus, the yellow tool is selected by evaluating h.

selection. The position and the pitch angle of the menu, given in the coordinate system of the stylus tracker, are connected to that tracker, whereas heading and roll are frozen to 0, see Figure 6.4. Hence, the menu is always oriented horizontally when following the hand; but can still be tilted.

The user selects a tool by turning the hand until the appropriate icon is highlighted. The icon is found by evaluating the heading angle and by clicking the stylus button afterwards. Note that there is no need for a pick ray to hit the icons. This selection method takes into account the fact that a selection of a tool is conceptually one-dimensional.

Turning the hand that way usually involves a slight change of the menu position as well, which is an unwanted effect. To properly isolate the menu movement from turning the hand around the wrist, the receiver coordinate system of the stylus, originally located close to the tip of the stylus, is shifted backwards to the direction of the stylus so that its origin coincides with the position of the wrist. An optimally stable position of the menu would be achieved if the shift vector were calibrated for each user separately.

Pressing a button on the input device activates the hand menu. Then, the menu is awaiting a tool selection, which is done by pressing the stylus button. Alternatively, pressing the button when no icon has been selected deactivates the menu. The menu disappears when a tool has been picked up, or when it has been deactivated. In case a tool has been selected, a cursor representing the chosen tool follows the hand. The button of the stylus or a button of the input device held by the non-dominant hand might be assigned to menu activation.

Advantages

In comparison to fixed menus, such as the toolbar placed at the front of the Responsive Workbench [CFH97], the hand menu has the following advantages:

– Icons are selected simply by turning the hand. There is no need for manual 3D selections, which would impose additional cognitive load on the user.

- The region of interaction with the menu and the manipulation space are co-located.
- The user can easily resolve occlusion situations of the menu with parts of the scene.

Applying the menu

Within the modeling application, the hand menu can be used to support all creation, manipulation, and system control tasks.

In order for manipulation of curves and surfaces to occur, each of which having its own set of tools, two different hand menus are configured according to the respective tool sets. Pointing at a curve, which highlights the curve, and pressing the menu activation button activates the curve menu. Pointing at an arbitrary surface activates the menu with the surface tools.

The tools used for creation, such as drawing a curve, and the system control tools, such as switching the rendering mode for surfaces, are put on a third menu. It appears when the user moves the stylus into an empty region and presses the activation button.

The hand menu is a general-purpose technique. However, dedicating special attention to the manipulation task seems to be necessary, considering the process of forming a model using many tools interchangeably. For this, the *ToolFinger* interaction widget is presented in the next section.

6.4.3 The ToolFinger

The most important action that frequently occurs while a virtual object is being manipulated is *selecting a tool*, including *switching between tools*. In applications such as geometric modeling, a large set of different editing tools can exist, which should be kept ready for instant selection and immediate application to an object.

For that purpose, a new interaction technique for projection-based VEs, called the *ToolFinger* [Wesc03], is proposed. The ToolFinger is an interaction widget that has been designed specifically to integrate tool selections and tool switches into complicated workflows that are typical for modifying and editing geometry. The ToolFinger supports the quick selection and immediate application of editing tools applied consecutively to objects in workbench-like VE systems.

Note that the ToolFinger is a software-based solution, due to the rather simple input devices that are common to most VE systems. The ToolFinger merely requires a hand-held input device with one button and tracked position and orientation. Best suited is a pen-like device, such as the commercially available and widely used stylus, or any similar device.

The ToolFinger technique integrates two actually separated tasks, namely tool selection, and tool application. The result is one seamless flow of action that especially benefits object manipulation using several tools. In traditional menu-based solutions, the user would have to interrupt an editing task and break his focus of attention each time he would need a new tool. In contrast to this, the ToolFinger reduces the sequence of interaction steps that are necessary to change the tool significantly. The approach pursued by the ToolFinger is that a tool is selected very close to the region of space where it is applied to an object. Secondly, tool application immediately and seamlessly must follow tool selection. Moreover, all this is performed by just one hand, usually the dominant hand, so that the non-dominant hand is free to position and orient the scene appropriately, thus enabling two-handed interaction.

The key idea of the ToolFinger approach is to subdivide a selection pointer into several segments. These are also referred to as sections. An intersection of an object

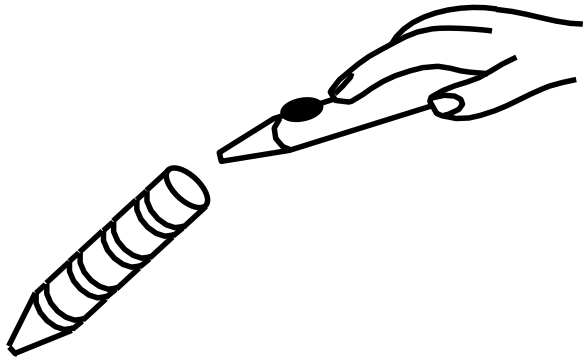

Figure 6.5: The design of the ToolFinger, shown with the hand and the input device. Thick segments correspond to selectable tools. They are separated by thin segments.

with one of those sections of the pointer corresponds to a tool selection. With each section of the ToolFinger, a tool is associated; see Figure 6.5.

Related work

Most of the summarized techniques for system control and selection of tools (see section 6.4.1) are based on moving the arm or the hand in order to put down or pick up an item or a prop from some location in the environment. Note that the user needs to change the focus of attention each time he reaches for a new tool.

Apart from those contributions, the main methods of application control and tool selection in VEs are based on 2D widgets that are used in a similar way as within desktop environments. In VEs, system control techniques that are similar to the ToolFinger approach are not known. In 2D environments, however, related techniques have been proposed.

The Toolglass [BSP+93] is a widget that can appear, as a sheet of glass, between an application area and a cursor. The Toolglass area is subdivided into regions to which a set of tools, or a set of alternative selections for an attribute, e.g. colors or contour shapes, is assigned. By positioning the widget relatively to the application so that the selection region overlaps an object, that selection can be applied, by moving the cursor inside. The principle of relating regions of interaction widgets with objects is very similar to the ToolFinger approach. However, the ToolFinger widget incorporates both a set of selection regions and the cursor, which are separated in the Toolglass approach. Consequently, the ToolFinger is operated with just one hand, in order to have the other hand available for orienting and positioning the model appropriately.

The FlowMenu [GW00] is a new kind of marking menu, for use with a pen device directly on large display devices such as wall-mounted displays. The FlowMenu consists of eight octants placed around a central rest area. Starting from that area, the user enters the fields of the menu with a pen, eventually activating sub-menus, without leaving the display area. Sequences of interactions can be arbitrarily long. They smoothly integrate command selection, text entry and direct manipulation, e.g. touching letters for quickwriting, or crossing octant lines clockwise or counterclockwise to scale an object. As in the ToolFinger approach, the FlowMenu aims at integrating consecutive tool selections and direct manipulation tasks fluidly. It is designed for display surfaces in a 2D environment, whereas the ToolFinger widget directly interacts with objects in a 3D virtual environment.

Purpose

Although the interactive complexity of applications running in table-like work environments tends to increase, there is a lack of novel interaction techniques specifically designed for handling complex object manipulation in a VE more efficiently. In these applications, various functions are typically defined for each object class. To mention just a few, most objects can be moved, copied, or deleted. Moreover, the application might support various geometric modeling functions, such as smoothing, deforming, or moving control points. Complex editing tasks are characterized by frequent changes of the current tool. This is an action that can distract the user from his main work if it is not properly supported. The main purpose of the ToolFinger is to provide an efficient method of dealing with that kind of interaction.

Design

The ToolFinger is an interaction widget shaped like a virtual pointer, or "finger", and formed by connecting several colored cylindrical segments. There are thin and thick segments. A thick segment has a length of about 1.3 cm, whereas a thin segment has half the length of a thick segment. The diameter of the segments is 1.3 cm. The tip of the ToolFinger has a shape similar to a cone, although its function is equivalent to that of a thick segment. Choosing that kind of design, the ToolFinger resembles a drawing tool, such as a pencil, which indicates its purpose.

Each thick segment of the finger corresponds to a specific manipulation tool, e.g. copy, move, delete, or moving control points. In order to prevent an abrupt change of the tool when moving the ToolFinger along the object, thin segments that have no tool assigned are placed in between two thick segments.

The ToolFinger is connected to the stylus tracker and therefore follows the hand of the user. Normally, at the Responsive Workbench, the hands are located above and in front of the scene. In order to support interacting with objects positioned that way, the ToolFinger can be tilted slightly downward, relative to the stylus axis, see Figure 6.5. The shown ToolFinger would have a length of about 9.1 cm.

Usage

Principally, the way of using the ToolFinger is very similar to the use of the familiar pick tool in a virtual environment. With the pick tool, usually represented by a virtual pointer following the hand, the user points at an object, presses a button, and moves the object to another location. With the ToolFinger, the user selects an object using that section that corresponds to the desired tool. The user then presses the stylus button and performs the action while continuing to keep the button pressed. This is followed by the release of the button allowing the ToolFinger to be applied in the same way again.

In order to select the right tool, the user is supported with a visual feedback mechanism. When a segment of the ToolFinger intersects an object, a short text describing the functionality of the associated tool appears right above the stylus, see Figure 6.6. It is therefore easily possible to browse through the available functions by moving the ToolFinger across the object, since a tool is only selected when the button is pressed.

Compared to the use of menus or toolboxes, where picking up a tool and applying the tool are separate steps, the ToolFinger approach integrates tool selection with tool application. The tool can be applied immediately after touching the object with the associated finger segment, by pressing the button. The use of the ToolFinger is illustrated in Figure 6.6 and 6.7. Suppose the user wants to move a curve. Using the ToolFinger technique, just one corresponding movement of the arm is sufficient to perform the whole task. The user grabs the curve with that section of the

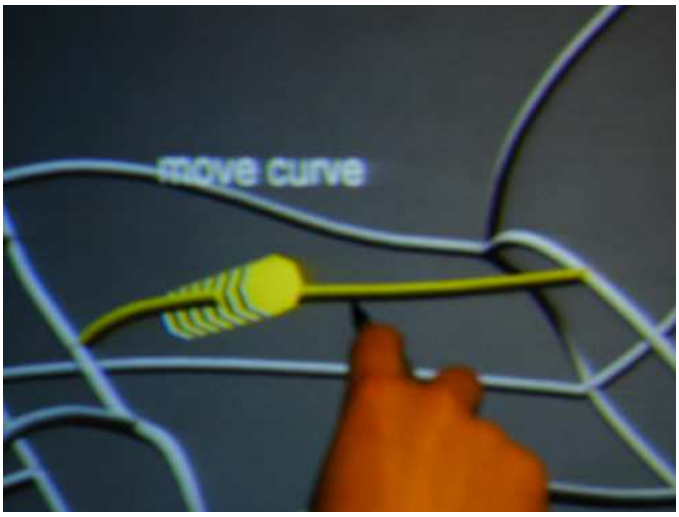

Figure 6.6: Selecting the move tool with the ToolFinger

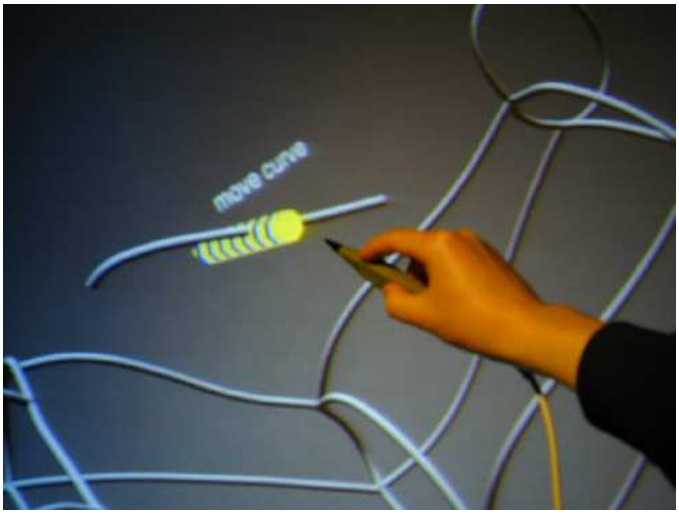

Figure 6.7: Moving the curve immediately after tool selection

ToolFinger that is associated with the move tool (Figure 6.6), presses the button and moves the curve (Figure 6.7). Releasing the button places the curve at the current location.

Advantages

The integration of a system control task into the flow of action of manipulation tasks is the main contribution of the ToolFinger approach. It implements a fluent transition between selecting a tool and applying it. The ToolFinger technique especially benefits complicated editing tasks that require many subsequent transitions between different tools. Three main characteristics distinguish the ToolFinger approach from traditional menu-based tool selection methods:

– The spatial area where the tool is selected and where it is applied is nearly the same. In other words, the hand always acts in vicinity to the object.

- The available manipulation tools are all attached to the hand at the same time.
- The ToolFinger can be completely controlled by the dominant hand alone.

These characteristics benefit performance and user comfort in the following way:

- The user's focus of attention can stay concentrated on the editing task. He can always look at the object that he is manipulating.
- There is no need for the user to interrupt his main task, since he is freed from moving away his arm in order to reach a menu item.
- Two-handed interaction, which is the predominantly used technique at the Responsive Workbench, is much supported by the ToolFinger. Since the non-dominant hand is left out of the tool selection process, it is always free to align the model appropriately.
- There is no need for a separate tool drop technique that would add increased complexity to the kind of editing tasks described, since the ToolFinger is ready for reuse after the button has been released.

Obviously, other tool selection methods have similar advantages, e.g. speech recognition or pinching. However, the use of speech input is often regarded as problematic, e.g. [FJZ98] reports frequently misinterpreted spoken commands and ambient noise affecting the system. Concerning the use of Pinch Gloves, in case the application requires additional hand-held input devices, pinching would be rather impractical.

Requirements

ToolFinger interaction requires accurate low-jitter tracking of the input device. The normally used manipulation space in a table-like environment is given by the reach of the arms, which corresponds to the limited working volume of usual electromagnetic tracking systems.

For a Responsive Workbench environment, it is suggested to mount the transmitter of the Polhemus Fastrak system at the front of the table, so that the stylus device sensor normally is within roughly 1 m of the transmitter. Within this range, usually ToolFinger interaction is not affected by jitter, and the accuracy is sufficient for interacting with detailed geometry.

Limitations

As shown, ToolFinger interaction is always related with a virtual object that is being manipulated. The ToolFinger therefore is not generally applicable for all tasks related with function selection or system control instead it is a special purpose technique. Consider e.g. object creation, and suppose a user wants to choose the class to create an object from, e.g. a curve, a cube, or a sphere. In such situations, it is not possible to choose the desired tool using the ToolFinger. Other system control tasks might benefit from the ToolFinger approach, but its way of use would not be obvious.

Another situation where the ToolFinger might not be ideal is applying the same tool to several objects. Consider e.g. deletion of objects. Using the ToolFinger would require the selection of the delete tool for each object separately. It would be more comfortable to pre-select the delete tool from a menu and just touch each object.

Consequently, there is a need of a combination of different system control techniques, including the ToolFinger, e.g. menus or equivalent approaches providing individual functions together with ToolFingers.

Applying the ToolFinger

The ToolFinger technique supports manipulation of objects with the hand acting in vicinity to the object. The tools assigned to curves and surfaces are shown in the following table:

class	tools
curves	*smooth, sharpen, drag, edit curve points, copy, move, mirror, delete*
surfaces	*drag, smooth, sharpen, delete*

Therefore, two ToolFingers are provided, one for curves, consisting of eight sections, and another one for surfaces, with four sections. The sections (shown in light color) are separated from each other by thin pieces to avoid sudden transitions between tools assigned to neighboring segments.

Suppose a user wants to elaborate a curve or a surface until he is satisfied with its shape, alternately using the tools *smooth*, *sharpen* and *edit curve points*. The user can easily edit the object by using different segments of the same finger. Tool transitions occur implicitly. Note that traditional menu-based tool switching would require interrupting the workflow each time when a new tool would be needed.

Variations and improvements

Further extensions and improvements of the ToolFinger technique are possible.

- With the ToolFinger, the combined selection of the object and the tool needs to be accomplished in 3D, although selecting a tool from a set is conceptually 1D. A mechanism that constraints the movement of the ToolFinger to stay in contact with the object while the user is moving it would ease browsing through the set of tools. Then, the intersection of an additional segment of the ToolFinger with the object would allow reactivating free movement.

- Since several kinds of objects usually appear in a scene, each associated with a different set of functions i.e. with a different ToolFinger, there should be a method of how to select the right ToolFinger. A possible solution is automatic selection of the right tool set when the pointer intersects an object. With the current implementation however, the user switches manually between different ToolFingers, by pressing a button.

- Another possible improvement is related with visual feedback. If the user were able to remember the location of the section associated with a certain tool, an instant tool application would be possible. Currently, the name of the tool only appears when the user intersects an object. Coloring or varying the geometry of individual sections of the ToolFinger could further support the user. In addition to that, sections belonging to tools that are needed very frequently could be made longer or thicker. This would result in a more distinct appearance as well, which would illustrate the possible choices of selection more clearly.

- The ToolFinger concept could be extended to better support the direct application of the same tool, i.e. without switching the tool, to several objects, as discussed above. For this, the chosen tool could be assigned to the tip of the ToolFinger as the default tool. In this manner, the ToolFinger could be

applied just as a usual pointer, representing a tool that was picked up from a toolbox.

– The size of the set of tools that can be handled by the ToolFinger is obviously limited. A size of about 8–10 tools seems to be appropriate in order not to exceed a reasonable size of the widget and to keep its sections long enough. A solution for supporting the selection from a larger set might be switching between several ToolFingers.

– An open question related to that is what would be the "optimal" size of a section of the ToolFinger, since the geometry of a section influences the usability of the technique. The size has to be chosen based on experiments, according to the number of tools, the geometry of the scene, the characteristics of the tracking system, and the fine motor skills of a user.

6.4.4 Summary

Two solutions to the problem of performing system control in complex VE applications have been proposed and implemented for use with the modeler.

The hand menu and the ToolFinger in comparison

The hand menu and the ToolFinger both have their advantages and limitations, which, in comparison, can be summarized as follows.

Hand menu	**ToolFinger**
General-purpose technique; support for creation, manipulation, and system control	Special-purpose technique that integrates system control with direct manipulation
Familiar concept, easy to learn for novice users	Suitable for advanced users, which have experience with 3D interaction
No 3D intersections required	Selecting a tool is a 3D operation. Mode errors are more likely to occur.
A selected tool is ready for repeated application.	Tool application always involves a selection process.
Repeatedly switching tools involves intense menu interaction.	Switching between tools produces no interaction overhead.
A tool is not immediately ready for application after selection. It has to be brought in position first.	The regions for selection and application of a tool are co-located.
Selection and application of a tool are two separate steps.	Selection and application of a tool are one action.
There is a need for an additional tool drop technique	A tool is active as long as the button is depressed.

Choosing interaction metaphors

Concerning interaction metaphors, note that the ToolFinger concept is *not* based on familiar work situations that occur in the physical world. Instead, the ToolFinger

concept makes use of the fact that both the scene and the tools are virtual objects. Note that, in reality, it hardly would be possible to hold more than one tool in hand at the same time, e.g. to hold a screwdriver, a file, and a hammer while working on a piece of material. Metaphors based on the physical world are often used to derive interaction concepts for virtual environments. This can restrict the possibilities when interacting with virtual objects and tools (see also [Poup00] for a discussion on artificial techniques).

As demonstrated by the ToolFinger, system control techniques based on artificial concepts are an alternative to traditional approaches, and could help to deal with the increasing functionality of virtual reality applications.

Non-conventional system control techniques currently are gaining relevance, parallel to the increasing functionality of virtual environment applications. Since the proposed methods cannot be a solution to all system control issues, a combination of different techniques that complement each other, promises to be a possible approach for future development.

Chapter 7

Collection of Sketches

In this chapter, it is shown how simple models are sketched directly at the Responsive Workbench, using the modeler described in previous chapters. Three models, a seat, a teapot, and a boat, are shown. For the seat, all stages of the design are explained in detail. The teapot and the boat were constructed in a similar manner.

7.1 Sketching a seat

The process of sketching a seat is shown in Figures 7.1–7.10. The user starts with drawing the two side lines of the left half of the seat (Figure 7.1). They are supposed to be reused for the right side as well. These curves are drawn as planar curves.

They are then repositioned so that the seat gets wider in the front. After applying the indirect curve smoothing tool (see section 5.2.3) to them, they are mirrored to obtain the right half. The plane of symmetry has been configured to be parallel to the vertical screen of the workbench, and is located in the middle of the horizontal screen.

In order to construct the front curves connecting both sides, the user draws them as symmetric curves freely in space, as described on page 91. Since the main drawing direction is from left to right, the model has been turned toward the front of the table before, using the left hand. Note that after the drawing strokes are complete, the curves snap into the network automatically, as described in section 4.4.1.

The remaining short curves of the front part are sketched as space curves and carefully deformed with the smoother, the sharpener, and the dragger (see Figure 7.2) until they have their final shape. The side curves of the front part are mirrored, and the curves connecting both halves are constructed using the symmetric drawing mode.

The backrest, which has a uniform width, is designed in a similar manner. The curve network is now complete, as shown in Figure 7.3.

The second step is creating the surfaces by pointing into the loops with the stylus (Figure 7.4). Not all loops can be found using this simple method, though (see section 4.5.1). Therefore, the tool for selecting individual pieces of a loop is used instead. For each remaining loop, the user points at the connected curve pieces that form the loop.

The initial surfaces specified that way already seem to have a pleasant shape (Figure 7.5). They adopt a shape outlined by the curve pieces surrounding them and are created according to table 4.2 on page 58.

In the last step, some of the surfaces are deformed. To vary the thickness of the headrest, the user sculpts it by applying the surface smoother (Figure 7.6) and the

sharpener to it (Figure 7.7). The smoother and the sharpener will keep the surface symmetric, opposed to the dragger that allows the user to deform the surface freely. To sculpt the front part, first the user aligns the model appropriately with the left hand, and then he deforms the surface with the right hand, as shown in Figure 7.8. The user inspecting the whole model is shown in Figure 7.9. Since the curve network is just an auxiliary means for the creation of the surface, it can be removed from the model, as shown in the Figure. In Figure 7.10, the final result is shown.

The initial sketch of the seat can now be exported for further elaboration. The entire process of sketching the shown model has been noted to take an experienced user approximately 15 minutes.

7.2 Sketching a teapot

In Figure 7.11 and 7.12, the sketch of a teapot is shown. Constructing the teapot was more difficult than sketching the seat. The teapot is more detailed, resulting in many small curves that needed to be drawn and shaped carefully.

7.3 Sketching a boat

On the other hand compared to the teapot, sketching the boat was a rather simple task, see Figure 7.13 and 7.14. The keel was drawn first as a planar curve. The whole fuselage of the boat was then assembled using four copies of the keel. Two of them, forming the starboard side, have been positioned by hand. Mirroring those two curves created the port side. The tail consists of two symmetric curves.

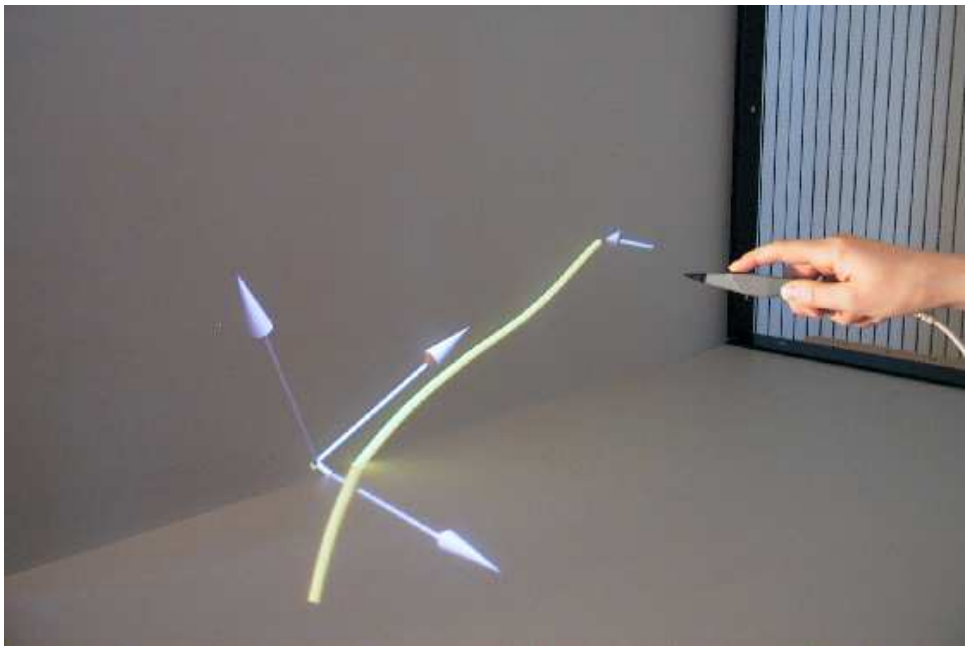

Figure 7.1: Drawing the first curve of the seat

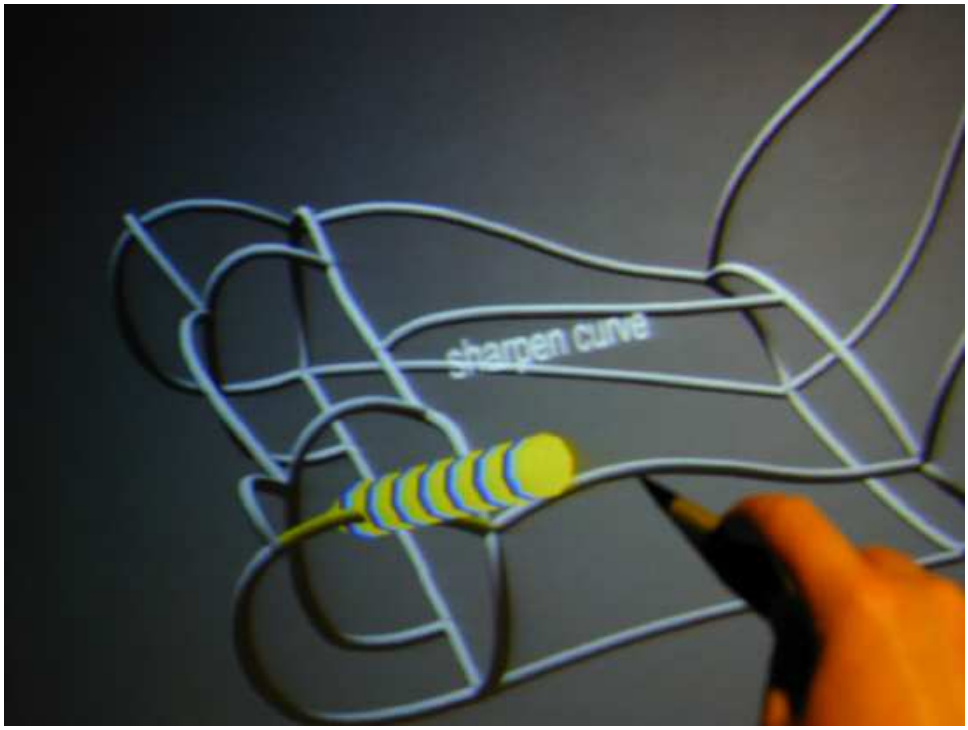

Figure 7.2: Deforming a curve, using the ToolFinger

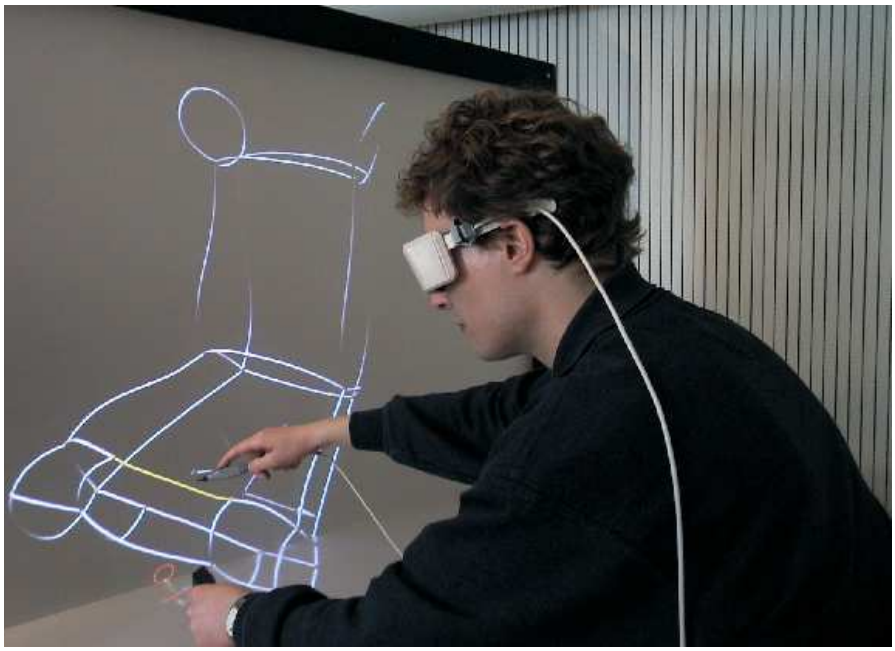

Figure 7.3: The curve network

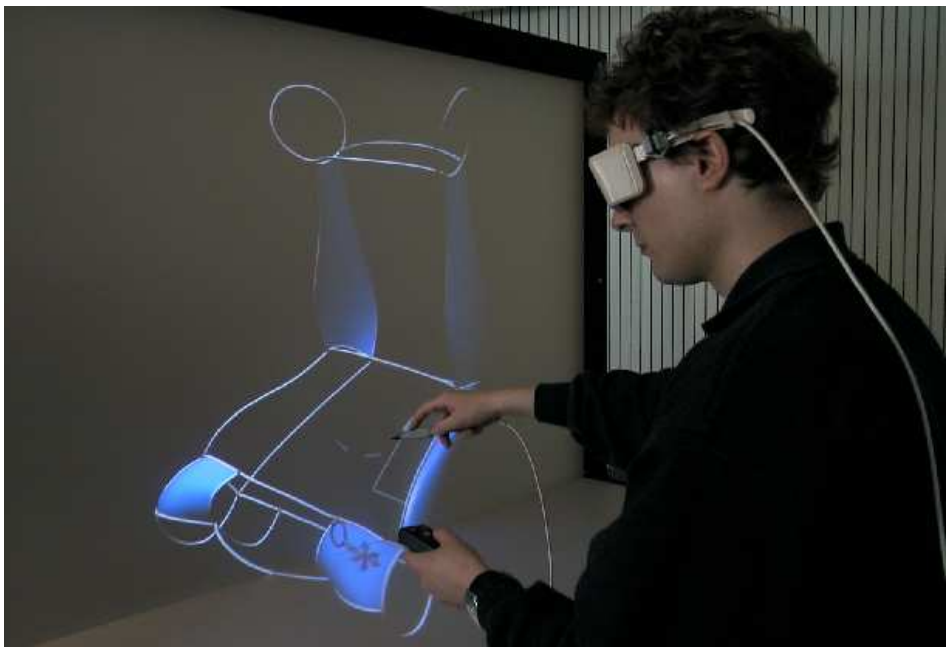

Figure 7.4: Creating surface parts

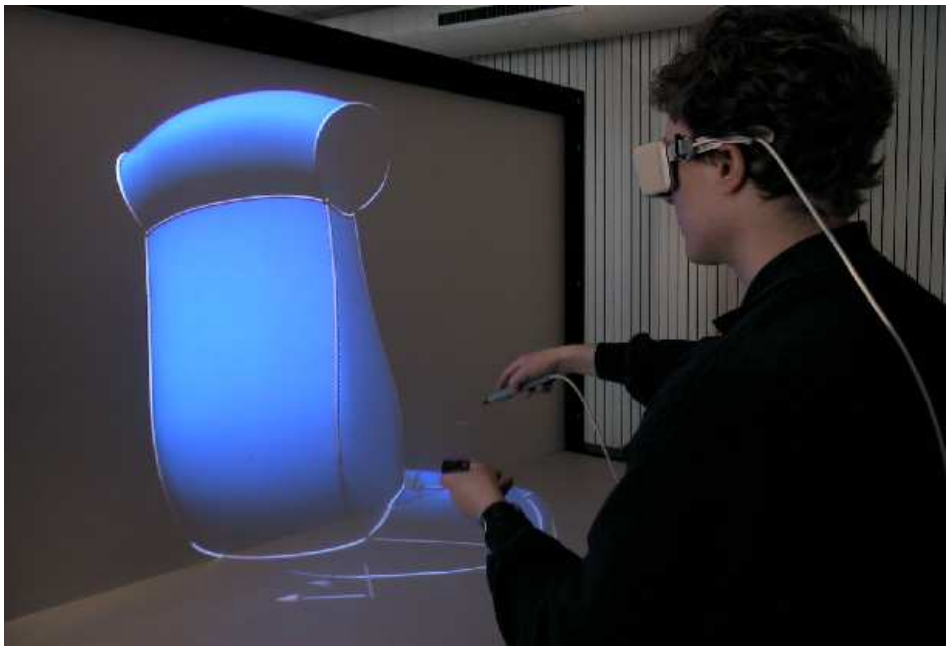

Figure 7.5: The curve network with the surface

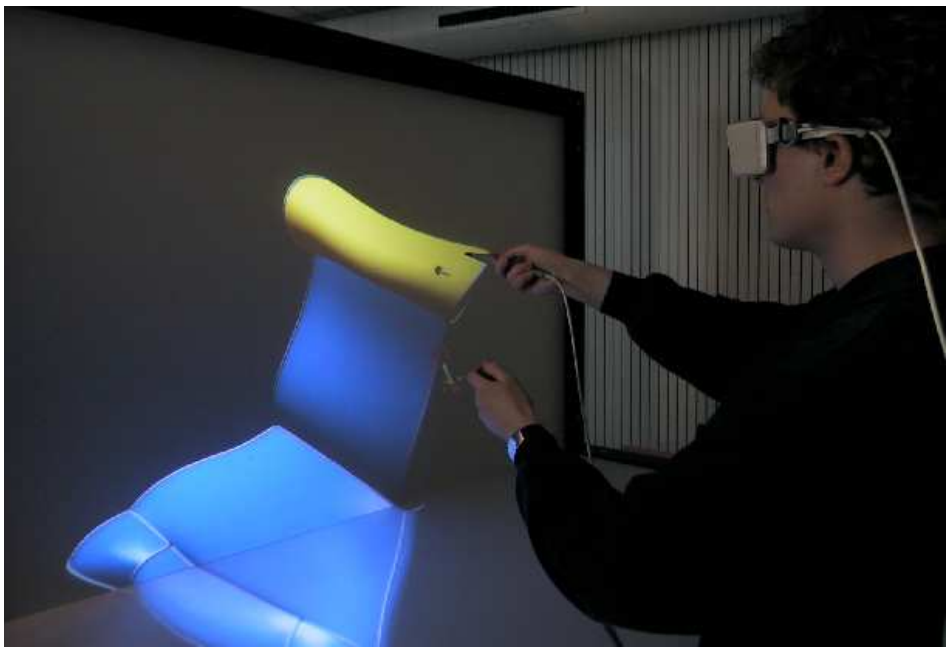

Figure 7.6: Sculpting the head-rest (1)

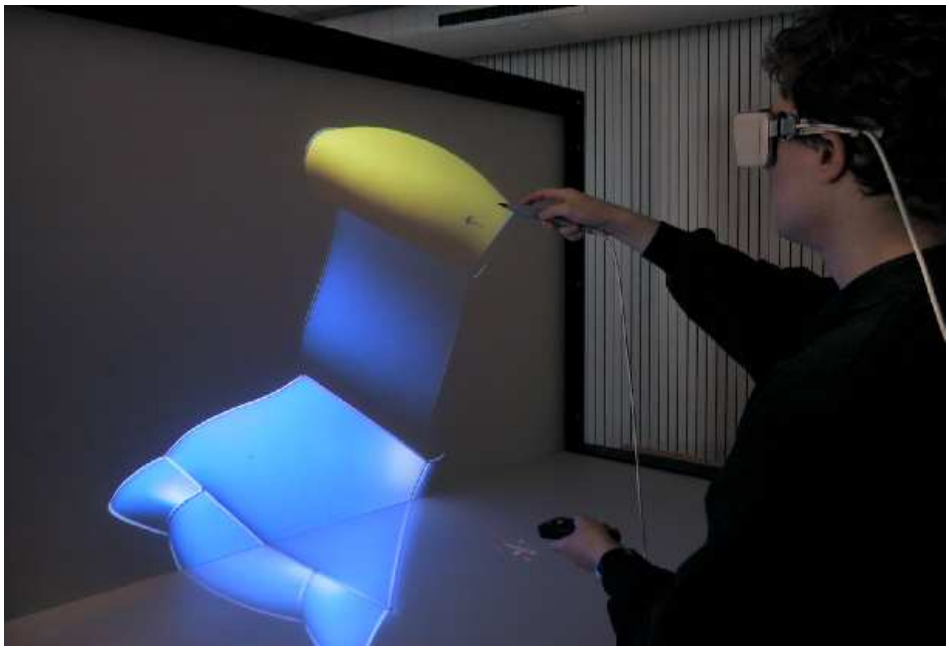

Figure 7.7: Sculpting the head-rest (2)

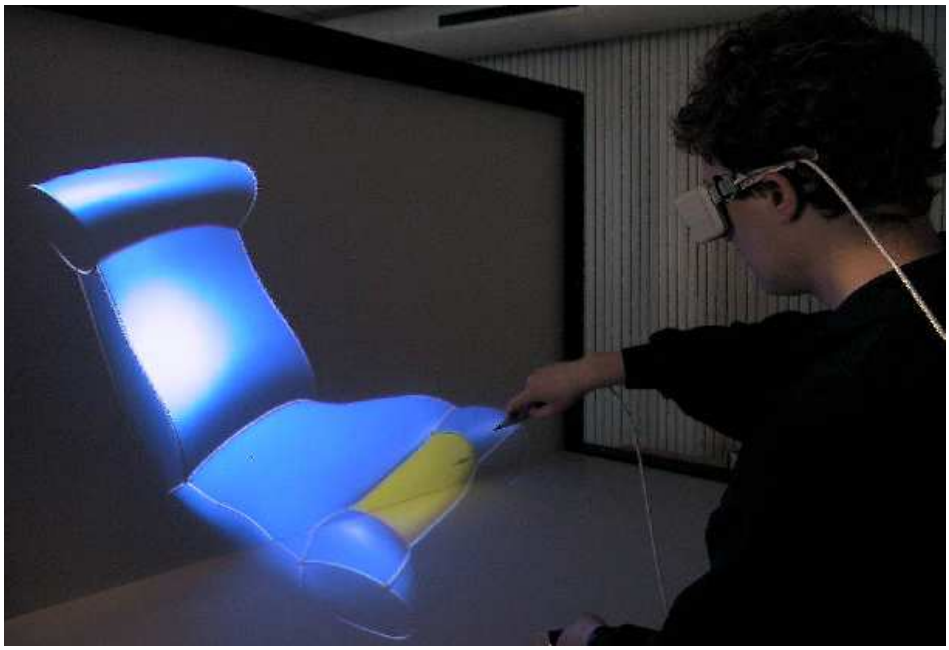

Figure 7.8: Sculpting the front part

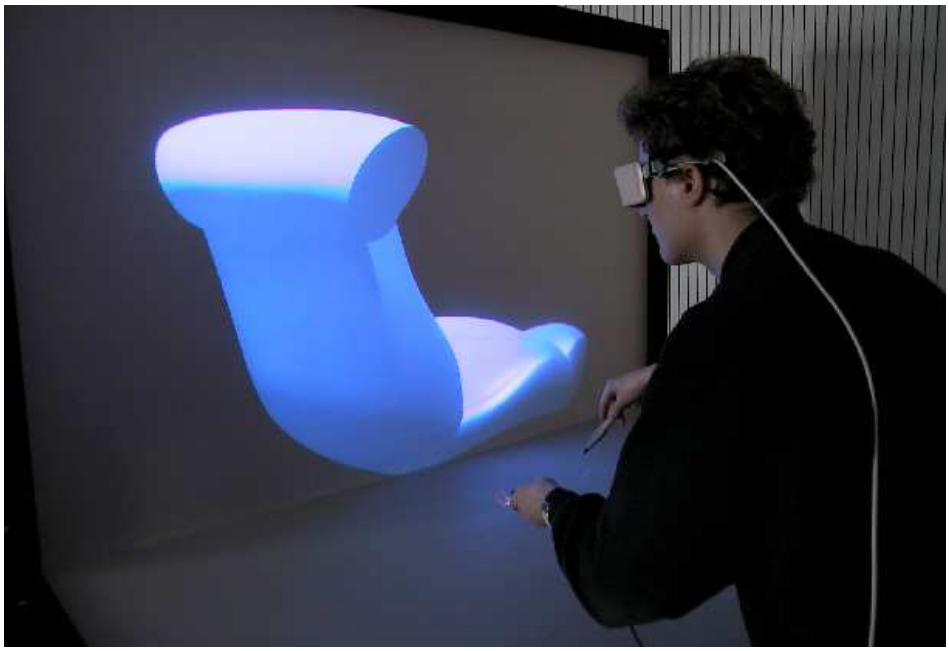

Figure 7.9: Inspecting the surface

Figure 7.10: The final sketch

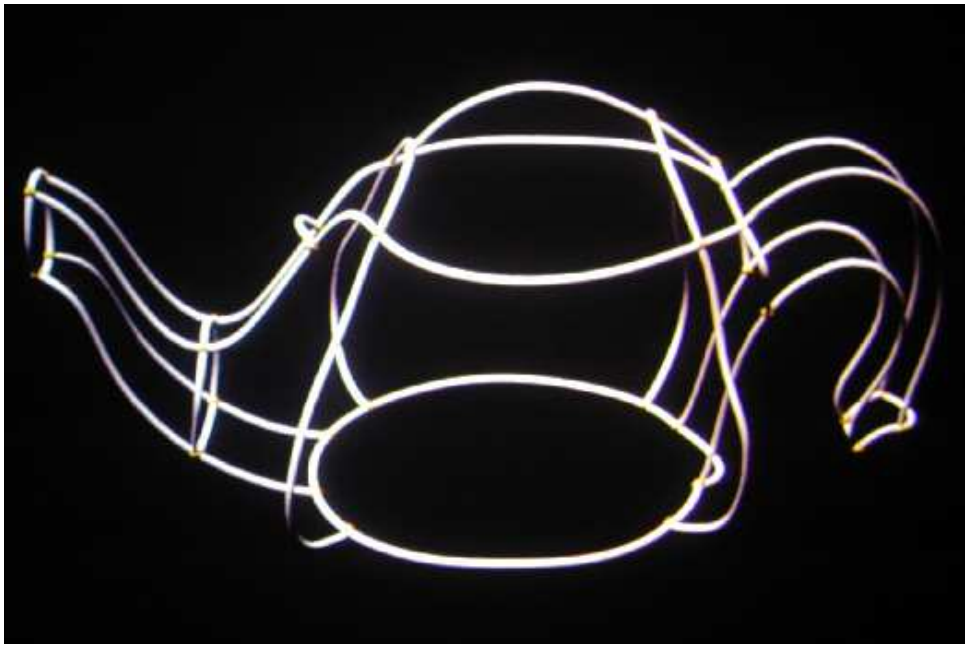

Figure 7.11: The curve network of a teapot

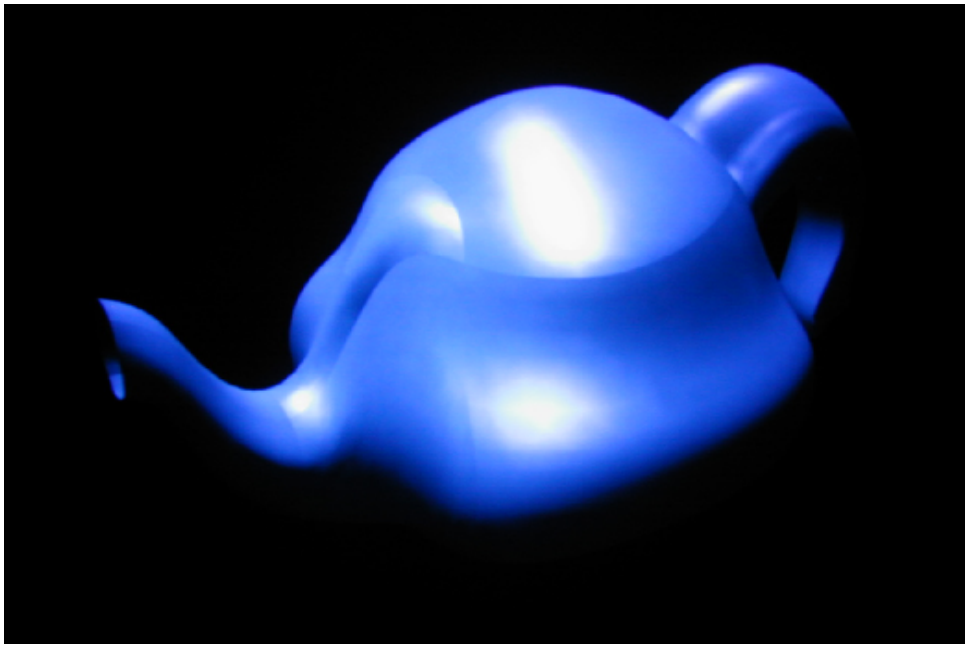

Figure 7.12: The sketch of a teapot

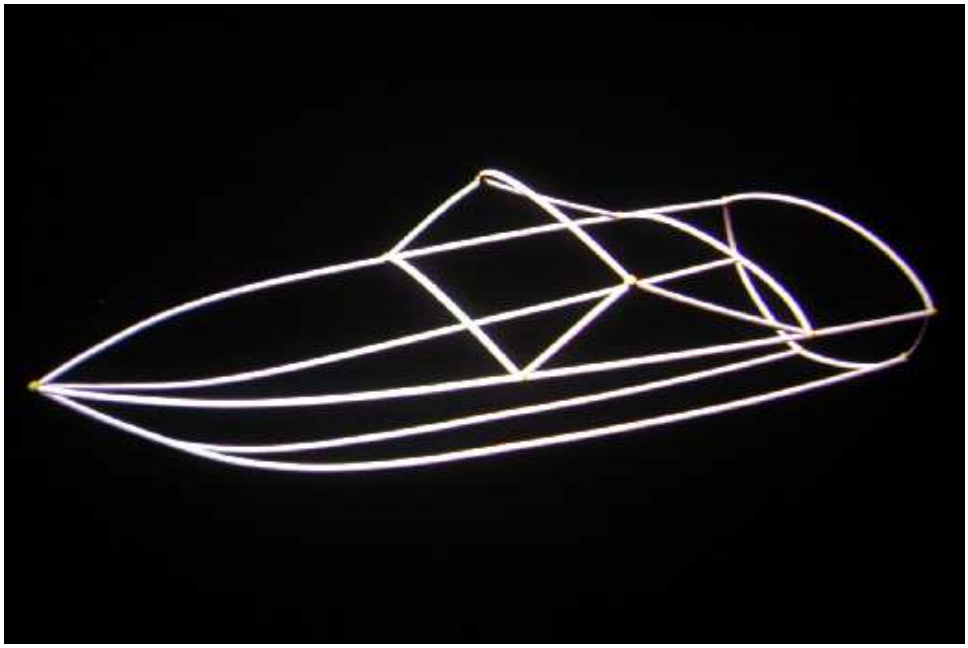

Figure 7.13: The curve network of a boat

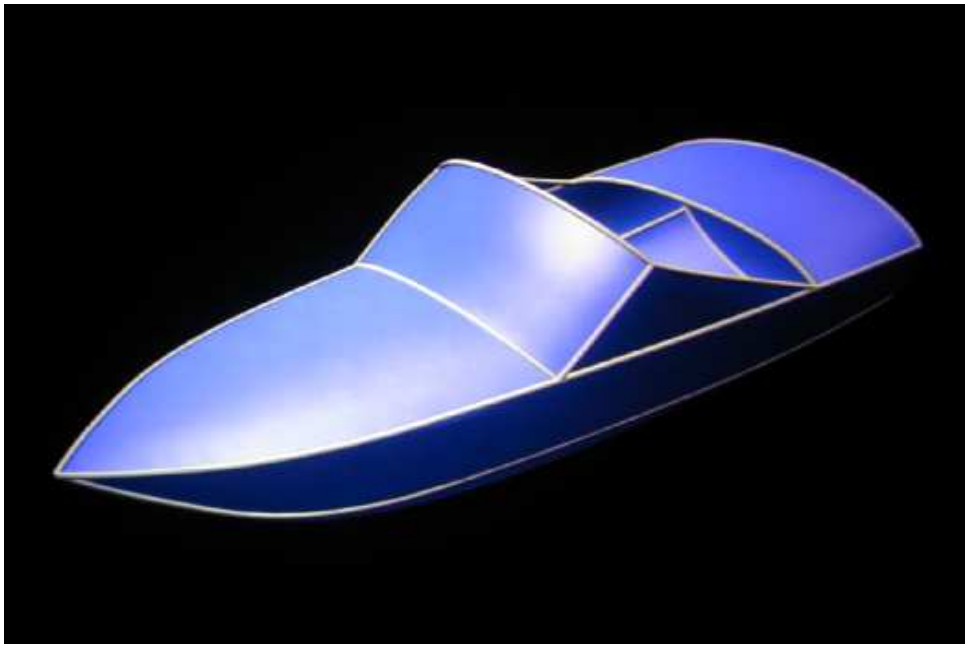

Figure 7.14: The sketch of a boat

Chapter 8

Conclusion

8.1 Conclusion

This work has shown that a different view on virtual environments makes possible an innovative use of such systems.

The input and output components of virtual environments are adapted to the natural interplay of using the visual sense and the hands. It is thereby important to note that using the hands in virtual environments should not be restricted to imitating interaction with physical objects. Principally, the *general ability* of humans to use their hands is supported. Therefore, artificial concepts for methods of work could be developed that are only possible in a virtual environment.

Based on this view, it was examined how conceptual shape design could benefit from the use of virtual environments. This process is insufficiently supported by current computer-aided design systems, and there is a strong need by designers for systems that better meet their needs.

A geometric modeling application was presented that supports the drafting of free-form models from scratch directly in a virtual environment. This involves several interaction tasks, such as shape creation, shape manipulation, and system control. Model creation is based on constructing curve networks, which on the one hand does not strongly support spontaneous sketching, but on the other hand eases the specification of surfaces considerably, since only the principal curves need to be drawn. Curve and surface shaping tools are based on variational modeling. The curve deformation tools are especially novel. They are directly usable in a virtual environment without force feedback.

It was demonstrated how complex systems can be controlled directly within the environment, using techniques such as hand-oriented menus, and new approaches, e.g. the ToolFinger. These techniques exploit the fact that hands directly act in space.

The proposed spatial interaction techniques for the tasks creation and manipulation are applicable to other geometric modeling systems as well. Furthermore the system control methods are applicable to many complex virtual environment based applications.

Concerning the usability, an earlier version of the system described in this work was ranked by designers with 2.4 [DBW+00], in a ranking scale of 1-3, where 1 is unsatisfactory and 3 is excellent. Designers and even some persons without a background in design, to which the current system was presented, were able to sketch simple models with it. Automotive design departments, and also auto industry suppliers, are very interested in using virtual environments and related display technologies for shape design. Currently, projects in this area are being planned, with

a strong industrial involvement on a European level. These projects are specifically dedicated to the conceptual shape design process. It is therefore foreseeable that conceptual shape design will change toward innovative use of computer support.

However, it is difficult to predict how conceptual styling systems of the future will look like. Some promising approaches have been proposed. A future system could take advantage of a combination of several techniques. What may occur in the near future is a change of currently non-computerized steps in the shape design process. For example, consider the use of digital tape drawing (see section 2.2.3 on page 18), together with the use of curve deformation and system control tools, such as those presented in this work.

8.2 Schlussfolgerung

Diese Arbeit hat virtuelle Umgebungen aus einem anderen Blickwinkel betrachtet und gezeigt, dass dies eine neuartige Nutzung dieser Systeme ermöglicht.

Die Eingabe- und Ausgabekomponenten virtueller Umgebungen sind an das natürliche Zusammenspiel zwischen der visuellen Wahrnehmung und der Benutzung der Hände angepasst. Hierbei ist es wichtig, dass die Art der Benutzung der Hände in virtuellen Umgebungen nicht darauf beschränkt sein muss, die Interaktion mit realen Objekten nachzuahmen. Prinzipiell wird die *allgemeine Fähigkeit* von Menschen, ihre Hände zu gebrauchen, unterstützt. Daher können auch künstliche Ansätze für Interaktionsmethoden entwickelt werden, die nur in virtuellen Umgebungen ausführbar sind, wie etwa beim ToolFinger.

Hierauf basierend wurde die Frage diskutiert, inwieweit die konzeptionelle Formgestaltung von der Benutzung virtueller Umgebungen profitieren könnte. Dieser Prozess wird nur unzureichend von aktuellen Computer-Aided-Design-Systemen unterstützt. Es gibt einen großen Bedarf seitens der Designer nach Systemen, die ihren Anforderungen besser entgegenkommen.

Eine Anwendung für geometrische Modellierung wurde beschrieben, die das Entwerfen von Freiform-Modellen von Grund auf, direkt innerhalb einer virtuellen Umgebung, unterstützt. Diese Anwendung beinhaltet mehrere Interaktionsaufgaben, wie Erzeugen einer Form, Verformung, und Anwendungskontrolle. Das Erzeugen von Modellen basiert auf der Konstruktion von Kurvennetzen, was einerseits ein spontanes Entwerfen nicht direkt unterstützt, aber andererseits die Formung von Flächen erheblich erleichtert, da nur Form beschreibende Randkurven gezeichnet werden müssen. Die Werkzeuge zum Formen von Kurven und Flächen basieren auf Methoden des Variational Modeling. Besonders die Werkzeuge für Kurven sind neuartig; sie sind direkt in einer virtuellen Umgebung sogar ohne Kraftrückkopplung anwendbar.

Es wurde gezeigt, wie komplexe Anwendungen direkt innerhalb der virtuellen Umgebung durch Techniken wie das Hand-Menü, oder durch neue Ansätze, wie dem ToolFinger, gesteuert werden können. Diese Ansätze nutzen die Tatsache aus, dass die Hände frei im Raum agieren können.

Die vorgeschlagenen räumlichen Interaktionstechniken zur Formerzeugung und Objekt-Manipulation sind auch für andere Modellierungssysteme anwendbar. Die Techniken zur Systemkontrolle sind für viele komplexe Anwendungen in virtuellen Umgebungen geeignet.

Was die Benutzbarkeit betrifft, wurde eine frühere Version des in dieser Arbeit beschriebenen Systems mit 2,4 bewertet, in einer Bewertungsskala von 1-3, wobei 1 unbefriedigend und 3 ausgezeichnet bedeutet [DBW+00]. Designer und auch einige Personen ohne Erfahrung auf diesem Gebiet, denen der aktuelle Modellierer präsentiert wurde, waren in der Lage, damit einfache Modelle zu entwerfen. Designabteilungen der Automobilindustrie und auch der Zulieferindustrie sind sehr

interessiert daran, virtuelle Umgebungen und ähnliche Displaytechnologien für das Design einzusetzen. Zurzeit werden auf diesem Gebiet Projekte auf europäischer Ebene mit starker Industriebeteiligung geplant. Diese Projekte widmen sich speziell dem konzeptionellen Designprozess. Es ist daher vorhersehbar, dass konzeptioneller Formentwurf die Nutzung von Computern in innovativer Weise einbeziehen wird.

Wie computerunterstützter Entwurf in der Zukunft aussehen wird, ist allerdings schwer abzuschätzen. Mehrere viel versprechende Ansätze wurden bereits präsentiert. Ein zukünftiges System könnte sich die Vorteile mehrerer Techniken zu Nutze machen, indem es sie kombiniert. Was sich in naher Zukunft abzeichnen könnte, ist ein Wandel zurzeit noch nicht computer-unterstützter Methoden im Designprozess. Als Beispiel hierfür ist eine Kombination aus Techniken des Digital Tape Drawing (siehe Abschnitt 2.2.3 auf Seite 18), zusammen mit Werkzeugen zur Kurvenverformung und zur Systemkontrolle, etwa wie sie hier präsentiert wurden, denkbar.

Acknowledgements

Many people supported this work directly or indirectly. First of all, many thanks to Hans-Peter Seidel, who supervised this dissertation. He gave many useful advices and fruitful comments on this work.

I'd also like to thank Heinrich Müller, who co-supervised this thesis. I gratefully acknowledge discussions with him concerning the topics spatial interaction, and geometric modeling.

This work would not have been possible without the funding of the Deutsche Forschungsgemeinschaft (DFG). The project "Real-time interaction with free-form surfaces in virtual environments", which this dissertation is based on, was funded under the grant numbers GO 856/2-1 and GO 856/2-2.

In this context, I would like to thank Martin Göbel and Martin Reiser who have supported this project. Marc Droske has implemented many useful modules the modeler is using, including solving linear equations, rendering curves and surfaces, and parts of the code for curve deformation. Igor Nikitin has implemented one of the methods to select a loop within a network of curves.

I also would like to thank the Fraunhofer Institute for Media Communication IMK in Sankt Augustin, an institute of the former GMD – German National Research Center for Information Technology. This is a very creative environment that fosters us to explore novel directions in the field of Virtual Environments.

I also have to mention Wolfgang Krüger†, whose idea of the Responsive Workbench led to further ideas, e.g. using it for such complex things as free-form surface modeling.

Without the virtual reality framework "Avango", on top of which our modeler is implemented, it would have been much more difficult to make everything work. Many thanks to the development group, including Jan Springer, Jürgen Wind, and Bernd Fröhlich, who implemented classes for interaction at the Responsive Workbench. These were used to derive classes for the interaction and system control methods of our modeler.

Many thanks also to Joachim Deisinger at Fraunhofer IAO; and to my colleagues Ernst Kruijff, Andreas Simon, Gernot Goebbels, and Florian Dombois. We had many fruitful discussions on spatial modeling and interaction. Thanks as well to all other people in the Virtual Environments group at IMK. Klaus-Günter Rautenberg assisted the – sometimes quite complicated – video sessions that were necessary to document this work.

The review service "English Text Doctor", available at [ETD] read through the text and corrected several errors.

Finally, thanks to my parents for their support; and especially, many thanks to my lovely wife Givanete for her patience with me, and for keeping me motivated.

Bibliography

[BBB97] H.J. Bullinger, W. Bauer, and M. Braun. *Handbook of Human Factors and Ergonomics*, chapter 52, pages 1725–1759. John Wiley & Sons, New York, 2^{nd} edition, 1997.

[BCMS03] M. Bailey, R. Crawford, S. McMains, and C.H. Séquin. 3D Hardcopy: Converting Virtual Reality to Physical Models. In *SIGGRAPH '03 Course Notes*, 2003.

[Bert99] J. Berta. Integrating VR and CAD. *IEEE Computer Graphics & Applications*, 19(5):14–19, September/October 1999.

[BFBK00] W. Buxton, G. Fitzmaurice, R. Balkrishnan, and G. Kurtenbach. Large Displays in Automotive Design. *IEEE Computer Graphics and Applications*, pages 68–75, July/August 2000.

[BFKB99] R. Balakrishnan, G. Fitzmaurice, G. Kurtenbach, and W. Buxton. Digital Tape Drawing. In *ACM UIST '99 Proceedings*, pages 161–169, 1999.

[BKJ+00] D. Bowman, E. Kruijff, J. LaViola Jr., M. Mine, and I. Poupyrev. 3D User Interface Design: Fundamental Techniques, Theory, and Practice. In *SIGGRAPH '00 Course Notes*, July 2000.

[Broo99] F.P. Brooks. What's Real About Virtual Reality? *IEEE Computer Graphics & Applications*, 19(6):16–27, November/December 1999.

[BSP+93] E.A. Bier, M.C. Stone, K. Pier, W. Buxton, and T.D. DeRose. Toolglass and Magic Lenses: The See-Through Interface. In *SIGGRAPH '93 Proceedings*, pages 73–80, 1993.

[BT81] B.A. Barsky and S.W. Thomas. TRANSSPLINE – A system for representing curves using transformations among four spline formulations. *The Computer Journal*, 24(3):271–277, 1981.

[BW01] D. Bowman and C. Wingrave. Design and Evaluation of Menu Systems for Immersive Virtual Environments. In *Proceedings of IEEE Virtual Reality '01*, pages 149–156, Yokohama, Japan, March 2001.

[CC78] E. Catmull and J. Clark. Recursively Generated B-Spline Surfaces on Arbitrary Topological Meshes. *Computer Aided Design*, 10(6):350–355, 1978.

[CFH97] L. Cutler, B. Fröhlich, and P. Hanrahan. Two-handed Direct Manipulation on the Responsive Workbench. In *Proc. Symposium on Interactive 3D Graphics*, 1997.

[Clar76] J. H. Clark. Designing Surfaces in 3-D. *Communications of the ACM*, 19(8):454–460, August 1976.

[CW99] S. Coquillart and G. Wesche. The Virtual Palette and the Virtual Remote Control Panel: A Device and an Interaction Paradigm for the Responsive Workbench. In *IEEE VR '99 Proceedings*, pages 213–216, Houston, Texas, USA, March 1999.

[DBW+00] J. Deisinger, R. Blach, G. Wesche, R. Breining, and A. Simon. Towards Immersive Modeling - Challenges and Recommendations: A Workshop Analysing the Needs of Designers. In *Eurographics Workshop on Virtual Environments*, pages 145–156, June 2000.

[DG97] T. H. Dani and R. Gadh. Creation of concept shape designs via a virtual reality interface. *Computer-Aided Design*, 29(8):555–563, 1997.

[DS78] D. Doo and M. Sabin. Analysis of the Behaviour of Recursive Division Rules near Extraordinary Points. *Computer Aided Design*, 10(6):356–360, 1978.

[Elli95] S. Ellis. Human Engineering in Virtual Environments. In *Virtual Reality World '95*, pages 295–301, Stuttgart, February 1995.

[ETD] http://www.englishtextdoctor.com/.

[Fari97] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, 4^{th} edition, 1997.

[FJZ98] A.S. Forsberg, J.J. LaViola Jr., and R.C. Zeleznik. ErgoDesk: A Framework for Two and Three Dimensional Interaction at the ActiveDesk. In *Proceedings of the Second International Immersive Projection Technology Workshop*, pages 11–12, Ames, Iowa, USA, May 1998.

[FPW+00] B. Fröhlich, J. Plate, J. Wind, G. Wesche, and M. Göbel. Cubic-Mouse-Based Interaction in Virtual Environments. *IEEE Computer Graphics & Applications*, 20(4):12–15, July/August 2000.

[FS89] J. Foley and J. Silbert. User-computer interface design. In *Lecture Notes, CHI '89 conference*, Austin, Texas, 1989.

[FSCN93] T.A. De Fanti, D.J. Saudi, and C. Cruz-Neira. A Room with a View. *IEEE Spectrum*, 30(10):30–33, October 1993.

[GBK+01] T. Grossman, R. Balakrishnan, G. Kurtenbach, G. Fitzmaurice, A. Khan, and B. Buxton. Interaction techniques for 3D modeling on large displays. In *ACM Symposium on Interactive 3D Graphics*, pages 17–23, 2001.

[GBK+02] T. Grossman, R. Balakrishnan, G. Kurtenbach, G. Fitzmaurice, A. Khan, and B. Buxton. Creating Principal 3D Curves with Digital Tape Drawing. In *CHI '02 Proceedings*, pages 121–128, 2002.

[GC01] J. Grosjean and S. Coquillart. Command & Control Cube: a Shortcut Paradigm for Virtual Environments. In *Immersive Projection Technology and Virtual Environments 2001 Proceedings*, pages 1–12, May 2001.

[GLS95] M. Green, J. Liang, and C. Shaw. Interactive 3D Geometrical Modelers for Virtual Reality and Design. In *Proc. International Conference on Virtual Systems and Multimedia '95*, pages 29–36, Gifu, Japan, September 1995.

[GLW96] G. Greiner, J. Loos, and W. Wesselink. Surface Modeling with Data Dependent Energy Functionals. In *Eurographics '96*, volume 15, pages 97–110, 1996.

[Grib99] M. Gribnau. *Two-handed interaction in computer supported 3D conceptual modeling.* Dissertation, Technical University of Delft, The Netherlands, October 1999.

[GS97] G. Greiner and H.-P. Seidel. Automatic Modeling of Smooth Spline Surfaces. In N. Magnenat-Thalmann and V. Skala, editors, *Proc. WSCG '97*, pages 665–675, 1997.

[GTB+04] G. Goebbels, K. Troche, M. Braun, A. Ivanovic, A. Grab, K. v.Lübtow, R. Sader, F. Zeilhofer, K. Albrecht, and K. Praxmarer. ARSyS-Tricorder – Entwicklung eines Augmented Reality Systems für die intraoperative Navigation in der MKG Chirurgie. In *2. Jahrestagung der Deutschen Gesellschaft für Computer- und Roboterassistierte Chirurgie e.V.*, Nürnberg, 2004.

[Guia87] Y. Guiard. Asymmetric Division of Labor in Human Skilled Bimanual Action: The Kinematic Chain as a Model. *Journal of Motor Behavior*, 19:486–517, 1987.

[GW00] F. Guimbretiére and T. Winograd. FlowMenu: Combining Command, Text, and Data Entry. In *UIST '00 Proceedings*, pages 213–216, 2000.

[GWWF00] M. Göbel, G. Wesche, J. Wind, and B. Fröhlich. Interactive Engineering in Virtual Environments. Unpublished project sheet, available at Fraunhofer IMK, Sankt Augustin, Germany, January 2000.

[Hanr82] P. Hanrahan. Creating Volume Models from Edge-Vertex Graphs. *Computer Graphics*, 16(3):77–84, July 1982.

[Harr] G. Harrod. 3D Printing Gets Into Top Gear. http://www.cadinfo.net/editorial/z402.htm.

[HDSG97] H. Haase, F. Dai, J. Strassner, and M. Göbel. *Immersive investigation of scientific data.* IEEE Computer Society Press, 1997.

[HKD93] M. Halstead, M. Kass, and T. DeRose. Efficient, Fair Interpolation using Catmull-Clark Surfaces. In *SIGGRAPH '93 Proceedings*, pages 35–44, 1993.

[HPGK94] K. Hinkley, R. Pausch, J. Goble, and N. Kassell. Passive Real-World Interface Props for Neurosurgical Visualization. In *Proceedings of ACM CHI '94*, pages 452–458, 1994.

[HPO+97] C. Hummels, A. Paalder, C. Overbeeke, P.J. Stappers, and G. Smets. Two-handed gesture-based car styling in a virtual environment. In *30th ISATA Proceedings, Mechatronics*, 1997.

[HSO97] C. Hummels, G. Smets, and C. Overbeeke. An Intuitive Two-handed Gestural Interface for Computer Supported Product Design. In *Proceedings of the Bielefeld Gesture Workshop*, September 1997.

[IMT99] T. Igarashi, S. Matsuoka, and H. Tanaka. Teddy: A Sketching Interface for 3D Freeform Design. In *SIGGRAPH '99 Proceedings*, pages 409–416, 1999.

[Joy] K. Joy. Bicubic uniform B-spline surface refinement. http://graphics.cs.ucdavis.edu/CAGDNotes/CAGD-Notes.html.

[KBF+95] W. Krüger, C. A. Bohn, B. Fröhlich, H. Schüth, W. Strauss, and G. Wesche. The Responsive Workbench: A virtual work environment. *IEEE Computer*, 28(7):42–48, 1995.

[KBS94] P. Kabbash, W. Buxton, and A. Sellen. Two-handed input in a compound task. In *Proceedings of ACM CHI 94*, pages 417–423, 1994.

[KL96] F. L. Krause and J. Lüddemann. Virtual Clay Modeling. In *Proceedings IFIP WG 5.2, Geometric Modeling for CAD*, 1996.

[Kobb00] L. Kobbelt. Discrete Fairing and Variational Subdivision for Freeform Surface Design. *The Visual Computer Journal*, 2000.

[Krue91] M. Krueger. *Artificial Reality.* Addison-Wesley, Reading, 1991.

[Krui00] E. Kruijff. System Control. In D. Bowman, E. Kruijff, J. LaViola Jr., M. Mine, and I. Poupyrev, editors, *3D User Interface Design: Fundamental Techniques, Theory, and Practice. SIGGRAPH '00 Course Notes*, pages 147–165, July 2000.

[Kuri94] S. Kuriyama. Surface modelling with an irregular network of curves via sweeping and blending. *Computer-Aided Design*, 26(8):597–606, 1994.

[LaVi00] J. LaViola. Multimodal Interfaces in Virtual Reality. In D. Bowman, E. Kruijff, J. LaViola Jr. M. Mine, and I. Poupyrev, editors, *3D User Interface Design: Fundamental Techniques, Theory, and Practice. SIGGRAPH '00 Course Notes*, pages 217–221, July 2000.

[Levi99a] A. Levin. Combined Subdivision Schemes for the design of surfaces satisfying boundary conditions. *Computer Aided Geometric Design*, 16(5):345–354, 1999.

[Levi99b] A. Levin. Interpolating Nets of Curves By Smooth Subdivision Surfaces. In *SIGGRAPH '99 Proceedings*, pages 57–64, 1999.

[LG94] J. Liang and M. Green. JDCAD: A Highly Interactive 3D Modeling System. *Computers and Graphics*, 18(4):499–506, 1994.

[Luen84] D.G. Luenberger. *Linear and Nonlinear Programming.* Addison-Wesley, Reading, 2^{nd} edition, 1984.

[MBS97] M. Mine, F. Brooks, and C. Sequin. Moving objects in space: exploiting proprioception in virtual-environment interaction. In *SIGGRAPH '97 Proceedings*, pages 19–26, 1997.

[MCCH99] L. Markosian, J. M. Cohen, T. Crulli, and J. Hughes. Skin: A Constructive Approach to Modeling Free-form Shapes. In *SIGGRAPH '99 Proceedings*, pages 393–400, 1999.

[MWB+96] J. Menon, B. Wyvill, C. Bajaj, J. Bloomenthal, B. Guo, J. Hart, and G. Wyvill. Implicit Surfaces for Geometric Modeling and Computer Graphics. In *SIGGRAPH '96 Course Notes*, 1996.

[NUK98] H. Nishino, K. Utsumiya, and K. Korida. 3D Object Modeling Using Spatial and Pictographic Gestures. In *VRST '98 Proceedings*, pages 51–58, Taipeh, Taiwan, November 1998.

[NW99] I. Nikitin and G. Wesche. Discussion, 1999. The algorithm for topology extraction has been implemented by I. Nikitin.

[OKHS97] C. Overbeeke, T. Kehler, C. Hummels, and P.J. Stappers. Exploiting the expressive: Rapid entry of car designers' conceptual sketches into a CAD environment. In *30th ISATA Proceedings, Mechatronics*, 1997.

[PASS95] A. Pasko, V. Adzhiev, A. Sourin, and V. Savchenko. Function Representation in Geometric Modeling: Concepts, Implementation and Applications. *The Visual Computer*, 11(8):429–446, 1995.

[PERF] Iris Performer reference manual. SGI Inc., Mountain View, CA 94043, USA.

[Poup00] I. Poupyrev. The Art of Designing 3D Interfaces. In D. Bowman, E. Kruijff, J. LaViola Jr. M. Mine, and I. Poupyrev, editors, *3D User Interface Design: Fundamental Techniques, Theory, and Practice. SIGGRAPH '00 Course Notes*, pages 177–194, July 2000.

[PR97] J. Peters and U. Reif. The Simplest Subdivision Scheme for Smoothing Polyhedra. *ACM Transactions on Graphics*, 16(4), October 1997.

[PT97] L. Piegl and W. Tiller. *The NURBS Book*. Springer-Verlag, Berlin, 2^{nd} edition, 1997.

[PTVF92] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 2^{nd} edition, 1992.

[QMV98] H. Qin, C. Mandal, and B. C. Vermuri. Dynamic Catmull-Clark Subdivision Surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 4(3):215–229, July-September 1998.

[SdA00] A. Stork and R. de Amicis. ARCADE/VT - a Virtual Table-centric modeling system. In *Proceedings of 4th International Immersive Projection Technology Workshop (IPT '00)*, Ames, Iowa, June 2000.

[SES99] D. Schmalstieg, L.M. Encarnaçao, and Z. Szalavari. Using Transparent Props for Interaction with the Virtual Table. In *Proceedings of the 1999 ACM Symposium on Interactive 3D Graphics*, pages 147–154, 1999.

[SG94] C. Shaw and M. Green. Two-Handed Polygonal Surface Design. In *Proceedings of ACM UIST '94*, pages 205–212, 1994.

[SG97] C. Shaw and M. Green. THRED: A Two-Handed Design System. *Multimedia Systems Journal*, 5(2), March 1997.

[SPS01] S. Schkolne, M. Pruett, and P. Schröder. Surface drawing: Creating Organic 3D Shapes with the Hand and Tangible Tools. In *Proceedings SIGCHI '01*, 2001.

[SRS91] E. Sachs, A. Roberts, and D. Stoops. 3-Draw: A Tool for Designing 3D Shapes. *IEEE Computer Graphics & Applications*, pages 18–26, November 1991.

[SS99] S. Schkolne and P. Schröder. Surface Drawing. Technical Report CS-TR-99-03, Caltech Department of Computer Science, 1999.

[Stam98] J. Stam. Exact Evaluation of Catmull-Clark Subdivision Surfaces at Arbitrary Parameter Values. In *SIGGRAPH '98 Proceedings*, pages 395–404, 1998.

[Stam99a] J. Stam. Evaluation of Loop Subdivision Surfaces. In *Subdivision for Modeling and Animation, SIGGRAPH '99 Course Notes*, 1999.

[Stam99b] J. Stam. Exact Evaluation of Catmull-Clark Subdivision Surfaces at Arbitrary Parameter Values. In *Subdivision for Modeling and Animation, SIGGRAPH '99 Course Notes*, 1999.

[Tonn98] D. Tonnesen. *Dynamically Coupled Particle Systems for Geometric Modeling, Reconstruction, and Animation.* PhD thesis, University of Toronto, Canada, November 1998.

[Tove92] M. Tovey. Intuitive and objective processes in automotive design. *Design Studies*, 13(1):23–41, 1992.

[TQ94] D. Terzopoulos and H. Qin. Dynamic NURBS with Geometric Constraints for Interactive Sculpting. *ACM Transactions on Graphics*, 13(2):103–136, April 1994.

[Tram99] H. Tramberend. Avocado: A Distributed Virtual Reality Framework. In *IEEE VR '99 Proceedings*, pages 14–21, 1999.

[USV96] M. Usoh, M. Slater, and T. I. Vassilev. Collaborative Geometrical Modeling in Immersive Virtual Environments. In *3rd Eurographics Workshop on Virtual Environments*, Monte Carlo, 1996.

[vD93] C.G.C. van Dijk. Conceptual surface modeling for industrial design. In *Graphics, Design, and Visualization*, pages 271–278. Elsevier, North-Holland, 1993.

[vD94] C.G.C van Dijk. *Interactive modeling of transfinite surfaces with sketched design curves.* Thesis, Delft, 1994.

[vEvdH95] A. van Elsas, A.J. van den Hout, J.S.M. Vergeest, and W.F. Bronsvoort. Automatic Topology Extraction from an Irregular Network of Sketched 3D Curves. In *28th ISATA Proceedings, Mechatronics*, pages 135–142, 1995.

[VW95] R. C. Veltkamp and W. Wesselink. Modeling 3D Curves of Minimal Energy. In *Eurographics '95*, volume 14(3), pages 97–110, 1995.

[WD00] G. Wesche and M. Droske. Conceptual Free-Form Styling on the Responsive Workbench. In *VRST '00 Proceedings*, pages 83–91, Seoul, Korea, October 2000.

[Wesc03] G. Wesche. The ToolFinger: Supporting Complex Direct Manipulation in Virtual Environments. In *Proceedings of 7. Immersive Projection Technology Workshop and 9. Eurographics Workshop on Virtual Environments*, pages 39–45, Zurich, Switzerland, May 2003.

[Wess96] W. Wesselink. *Variational Modeling of Curves and Surfaces*. PhD thesis, Technical University Eindhoven, The Netherlands, 1996.

[WS01] G. Wesche and H.-P. Seidel. FreeDrawer–A Free-Form Sketching System on the Responsive Workbench. In *VRST '01 Proceedings*, pages 167–174, Banff, Alberta, Canada, November 2001.

[WV95] W. Wesselink and R. C. Veltkamp. Interactive design of constrained variational curves. *Computer-Aided Geometric Design*, 12(5):533–546, August 1995.

[WW92] W. Welch and A. Witkin. Variational Surface Modeling. In *SIGGRAPH '92 Proceedings*, pages 157–166, 1992.

[ZHH96] R. C. Zeleznik, K. P. Herndon, and J. F. Hughes. Sketch: An interface for sketching 3d scenes. In *SIGGRAPH '96 Proceedings*, pages 163–170, 1996.

[Zori99] D. Zorin. Subdivision Zoo. In D. Zorin and P. Schröder, editors, *Subdivision for Modeling and Animation. SIGGRAPH '99 Course Notes*, pages 65–87, 1999.