Interactive function based shape modelling

Levinski Konstantin

2007

Levinski, K. (2007). Interactive function based shape modelling. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/35730>

<https://doi.org/10.32657/10356/35730>

Interactive Function Based Shape Modelling

Levinski Konstantin

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in fulfilment of the requirement for the degree of
Doctor of Philosophy

2007

Acknowledgements

First and foremost, I am greatly indebted to my supervisor, Prof. Alexei Sourin, for his invaluable guidance and support during my work on the project. Without him, this work would have never become a reality, and his vision, involvement and attention was a great source of inspiration. Thank you for providing the indispensable drive behind all the aspects of my work.

I would also like to thank the School of Computer Engineering, Nanyang Technological University, Singapore for the generous financial support and excellent resources provided during my study here.

Konstantin D. Levinski

TABLE OF CONTENTS

Abstract..... 1

Chapter 1. INTRODUCTION 3

Chapter 2. USING FUNCTIONS IN SHAPE MODELLING 5

2.1 Recent Developments and Current Problems of Interactive Shape Modelling..... 5

2.2 Functions in shape modelling..... 9

2.2.1 Analytical functions..... 11

2.2.2 Approximating function 11

2.2.3 Skeletal modelling 12

2.2.4 Operation trees..... 14

2.2.5 FRep..... 15

2.2.6 Summary..... 16

2.3 Function acceleration methods for shape modelling..... 16

2.4 Interactive visualisation of function-defined models 20

2.4.1 Polygonal approximation..... 20

2.4.2 Point rendering 20

2.4.3 Visualisation methods evaluation..... 20

2.5 Polygonisation methods 21

2.6 Haptic rendering for function-based models..... 23

2.7 Summary 24

Chapter 3. INTERACTIVE FUNCTION-BASED MODELLING FRAMEWORK..... 25

3.1 Overview of the modelling process..... 25

3.2 Operators for shape modelling 26

3.3 Operator acceleration 29

3.4 Summary31

Chapter 4. GENERAL DESIGN AND IMPLEMENTATION ISSUES33

4.1 Design and implementation workflow33

4.2 Principles of shape functions definition35

4.3 Essential issues of operator definition35

4.4 General acceleration algorithms and data structures38

4.5 Visualization considerations40

4.6 Summary41

Chapter 5. INTERACTIVE RENDERING42

5.1 Polygonisation42

5.1.1 General characteristic of the method42

5.1.2 Polygonisation pipeline44

5.1.3 Cells used for propagation45

5.1.4 Cells detection50

5.1.5 Generation of polygons53

5.1.6 Re-meshing and fine polygonisation handling56

5.1.7 Optimal mesh data layout57

5.1.8 Performance measurement57

5.1.9 Image-based scene update59

5.2 Extension for parallel rendering62

5.3 Haptic rendering64

5.3.1 Forces for function-based haptic feedback64

5.3.2 Acceleration for haptic rendering65

5.4 Summary66

Chapter 6. INTERACTIVE MODELING ARCHITECTURE68

6.1.1 Modelling framework68

6.1.2 Function module70

6.1.3 Geometry extraction module	71
6.1.4 Geometric model	72
6.1.5 User interface module.....	73
6.1.6 Summary.....	75
6.2 Experimental results and discussion	76
6.3 Summary of the application design.....	78
Chapter 7. FUNCTION-BASED WEB VISUALISATION	80
7.1 Remote shape viewing	80
7.2 Function-based terrain.....	82
7.3 Summary	83
Chapter 8. DISTRIBUTED RAYTRACING	84
8.1 Raytracing	84
8.2 Operator clusters for ray tracing.....	85
8.3 Distributed ray tracing.....	86
8.4 Results	86
Chapter 9. CONCLUSION.....	91
9.1 Function-based modelling techniques.....	91
9.2 Visualisation algorithm	92
9.3 Using functions for remote and distributed visualisation.....	93
9.4 Deliverables.....	93
9.5 Future work	95
Bibliography (109 references).....	96

TABLE OF FIGURES

Figure 1. Basic modelling methods.	5
Figure 2. Examples of interactively sculptured models.	6
Figure 3. Phantom haptic device and a Free-form model created with the help of the force feedback.	7
Figure 4. Specialised modelling software.	8
Figure 5. A transformation of a blobby object defined with two-point skeleton.	13
Figure 6. Skeleton on the left defines the model shown on the right.	13
Figure 7. Example of the BlobTree. The shape on top is constructed by recursively applying operations in the nodes of the tree to their respective children.	14
Figure 8. A-patch convex hull and control points.	18
Figure 9. Blob-tree with levels of detail. Parts of the tree which will not contribute to the image are removed from the model to simplify rendering.	19
Figure 10. Function representation pipeline.	26
Figure 11. Examples of different operators.	28
Figure 12. Acceleration by skipping non-contributing operators.	30
Figure 13. Layout of the modelling system.	31
Figure 14. Function structure.	34
Figure 15. Defining the tool's coordinate system.	36
Figure 16. One-dimensional example of a modification in $[-1,1]$ interval.	37
Figure 17. Different approaches to defining the areas of influence for different modifications.	38
Figure 18. Smooth transition from tool to shape, using the guarding functions.	38
Figure 19. Two different cell types and their lattices.	46
Figure 20. Proposed optimal cell.	47
Figure 21. The process of finding the neighbour cells through mirroring.	48
Figure 22. Locating the tetrahedron containing a given point. (2D analogy).	49
Figure 23. Detection of the correct seeding tetrahedron.	50
Figure 24. Local update procedure.	53
Figure 25. Clustering polygonisation result.	56
Figure 26. The test shapes.	58
Figure 27. Image update stages.	61
Figure 28. Cell blocks.	62
Figure 29. Parallel surface extraction.	63
Figure 30. Normal and tangent components.	64

Figure 31. Force magnitude profile.	65
Figure 32. Using tetrahedra to accelerate function evaluation for haptic rendering.	65
Figure 33. Sculpting session using 6 DOF force feedback device.	66
Figure 34. Application module interaction.	70
Figure 35. Function construction and evaluation module.	71
Figure 36. Two-resolution polygoniser.	72
Figure 37. Storage of geometric model with fast access.	73
Figure 38. User interface layout.	75
Figure 39. Model with colour and program interface.	77
Figure 40. Initial shape and its defining formula.	77
Figure 41. Candlestick model with fine details and colour.	78
Figure 42. “Mouth of Truth” model sculptured out of a disk.	78
Figure 43. Web-based remote visualisation process.	81
Figure 44. Shape transferred over the web.	81
Figure 45. Repolygonisation for terrain generation.	82
Figure 46. Generated terrain, with and without fog.	83
Figure 47. Ray tracing architecture.	85
Figure 48. Dynamic sparse grid.	85
Figure 49. GRID distributed rendering.	86
Figure 50. Ray-traced examples.	88
Figure 51. Vase of ruby and face of marble.	89
Figure 52. High-quality ray traced image.	90

Abstract

In this work, interactive shape modelling is addressed using a function-based approach, where geometric shapes and their properties are defined with mathematical functions. Interactive modification of a function model with concurrent visualisation of the respective polygonal mesh provided both the interactivity and any required level of detail resulting in a photo-realistic appearance of the shapes. Mathematically the same functions are used for defining the shape's geometry as well as colours, which can be interactively applied to the shape. Any of the operators involved in creating the shape can be edited, modified or removed at any time thus allowing for a great flexibility of the modelling pipeline and opening prospects for efficient re-using and improvement of the models previously created.

The goal of the developed modelling system is to place as little restrictions on the interactive operations as possible. Most interactive modifications change the shape only in limited area, which allows optimizing the model by avoiding modifications irrelevant to a given point.

A robust visualisation method intended for function based shapes and interactive operations on them was proposed. The method uses two levels of detail to create a high quality renderer and an interactive visualisation tool suitable for complex objects. The image based algorithms made it possible to combine the properties to achieve interactive rendering of photographic quality images.

Lower resolution approximation is used for interactive manipulations and alterations, while local modifications are handled in a very effective way, allowing the user to modify the meshes consisting of hundreds of thousands of triangles, and providing a good interactive experience. An image-based approach allows for combining fine details with the interactive response, extending the possible applications of the algorithm even further. The mesh used for visualisation was reused to augment the rendering process with haptic feedback.

The interactive shape modelling system was designed and implemented, where the shape is interpolated by polygonisation algorithm and rendered on a common personal computer. The models can be visualised using the proposed rendering algorithm, patched GRID-enabled version of POV-Ray raytracing program or through a VRML plugin.

In addition to graphical rendering, efficient haptic rendering and interaction with function-based models using a 6-DOF Phantom device were developed, where the properties of force feedback can be consistently defined using functions themselves.

Chapter 1. INTRODUCTION

Many three dimensional crafts are performed as gradual modifications of initial shapes. For example, *modelling* generally refers to working with malleable or plastic materials, such as clay, using hand or simple tools such as spatulas or modelling tools. *Sculpting*, *carving* and *engraving* imply carving of solid material with sharp tools such as chisels, drills, burrs, or axes. *Embossing* is based on raising or depressing relatively small areas of a metal foil with punches and chisels. There is almost no limit to the choice of materials for any of these processes, and a great variety of tools can be used for them. However, all of these tools, or more exactly, single operations of applying these tools can be classified into:

- Local removal of material,
- Local deposition of material, and
- Local deformation of the work-piece.

Thousands of little individual operations constitute the real shape modelling process, and it has always been a challenge to simulate this process with a computer, using a high level of immersion and photo-realism. A common approach to interactive shape modelling is to define some data structure for storing geometric information, and based on input from the user, modify it so that the changes reflect the user's action. The operation itself is usually not included in the model, and therefore cannot be amended or discarded subsequently. Since the information about the user's intentions is lost, the artefacts of the underlying representation—aliasing, polygons, etc.—can often be noticed at a closer look at the model.

The goal is to develop methods and software which would allow us to interactively create and modify complex function-defined shapes. The function representation lets us define shapes which are difficult or impossible to make using other models, incorporate different models in one function definition, exchange the shape models over the Internet, and convert the function models into commonly used data formats. The software to be

developed is planned for use in engineering, scientific and medical visualisation application projects.

This thesis describes a function-based approach to interactive shape, where operations are fundamental building elements. Ultimately, the sequence of operations yields the resulting model, which can have any level of detail with no special storage required. If necessary, each stage of the modelling process can easily be derived from the proposed representation. The focus of this work is on the function representation of shapes using consequent modifications, achieving interactive rates of these modifications, as well as fast photorealistic rendering of the resulting objects.

The thesis is organised as follows:

Chapter 2 introduces current trends and work in related areas for a clearer view of the problem. Chapter 3 discusses methods used for creation and modification of the function-defined shapes. Chapter 4 describes issues to consider when designing a function-based modelling system, Chapter 5 deals with interactive rendering of function-based models and Chapter 6 presents an example of a modelling architecture based on the described principles. Remote visualisation using the VRML protocol is described in Chapter 7 and distributed ray-tracing of the function-based model is presented in Chapter 8. The summary and plans for further development are given in Chapter 9.

Chapter 2. USING FUNCTIONS IN SHAPE MODELLING

This Chapter reviews current trends in interactive shape modelling, followed by discussion of function-based methods for shape modelling, their advantages and drawbacks.

2.1 Recent Developments and Current Problems of Interactive Shape Modelling

The field of interactive shape modelling is continuously expanding. Let us review the major trends and demonstrate why it might be advantageous to approach the current problems with function-based methods.

Figure 1. Basic modelling methods.

(a) **Parameter-based model definition.** User gives explicit numeric parameters of the shape and gets it in one step. (b) **Interactively controlled creation of a model.**

Modifications are continuously guided by the user.

Many modelling systems represent shapes in a way optimised for computers and people with technical mindset. However, a prominent trend in the computing industry today is to adapt to different ways of thinking, i.e. embracing for people whose primary skills lie far from the computing field. In a shape modelling application, there can be different approaches to make shape modelling applications more user-friendly. One way

is extensive - by refining existing representations and evolving CAD-CAM modelling systems. This is clearly seen in examples such as 3D Studio MAX[*3DSMaxSoftware*], Maya [*MayaSoftware*] and TrueSpace [*TrueSpaceSoftware*] modelling systems, which are gradually adapting their toolsets, interfaces and representations to different ways of thinking. Methods for defining shapes and their modifications are evolving from adjusting numerical parameters (Figure 1(a)) to more intuitive and advanced techniques, like moving NURBS control points, lofting, subdivision surfaces, etc. (see examples in Figure 1 b). A great variety of plugins are available for these applications, to enable people to extend these packages for their needs.

Figure 2. Examples of interactively sculptured models.

(a) ZBrush software. This complex image was created with different tools using “painting in 3D” metaphor. (b) Freeform modelling system. Sculpting is a powerful way to express one’s ideas about shapes.

Currently, free-form interfaces are rapidly developing, along with data structures needed to support them. For example, [*ZBrushSoftware*] allows users to interactively sculpt the model, starting from an initially coarse shape. Sculpting metaphor, as well as other features taking advantages of drawing skills, like combination of 3D (sculpting) and 2D (painting) dimensions, make creating shapes, like those shown in Figure 2(a), easy for people familiar with these techniques. While efficient interactive methods are being utilised, the model itself is represented with polygons-points combination and quite often results in inconsistent meshes.

Another prominent example of successfully introduced new interaction concept is Sensable Inc. [SFFWeb]. The company has developed a haptic feedback device (Figure 3(a)) and a voxel-based modelling system, which employs the artist's clay modelling skills. The learning curve thus is greatly reduced, and real-world skills and natural human ability of working with geometric shapes are utilised. Voxel representation, however, requires high-end hardware and the resulting models (Figure 3(b)) are quite huge.

Figure 3. Phantom haptic device and a Free-form model created with the help of the force feedback.

(a) Phantom haptic device. (b) Resulting model. Sense of touch is very important for sculpting.

Various shape representations suitable for free-form interaction are a subject of many research projects. For example:

Kizamu system by MERL (Mitsubishi Electric Research Laboratories) offers an efficient way to store shapes. Fine details can be preserved, but interactive modification is not straightforward.

Variational modelling system jointly developed by MERL and Brown University allows for sketching of 3D models simply with 2D strokes. The resulting shapes do not contain sharp features, but nevertheless enable users to communicate spatial concepts effortlessly.

Canis M-P offers flexible adaptive shape representation for sculpting. While it provides intuitive control over the shape, fine details are limited and cannot be sufficiently controlled.

Shape modelling is being applied to a growing number of human tasks – from dental surgery modelling [CEREC3D] (Figure 4(a)) to jewellery design [TrueSpaceSoftware] (Figure 4(b)), shape retrieval [Funkhouser T. et al., 2003] and virtual environment interaction [DigitalArtForms].

As products become increasingly complex and timeframes are constantly shrinking, it seems beneficial to allow people to work on the same object simultaneously in a virtual environment. A lot of research is being conducted in this area, and commercial software is also appearing, for example [Alibre], [ImpactXoft], [Verse]. Web visualisation techniques for virtual environments compliment the collaboration modelling process, and also deliver project descriptions to the customer, so that he/she can evaluate the future product or make a decision about the existing one [Cortona].

Figure 4. Specialised modelling software.

(a) Dental modelling application. (b) Jewellery design system.

Operations with models are becoming more computationally demanding, as a result of application diversity and increased requirements. As there are indications that individual single-core processors have already reached the practical performance limit [Sutter H., 2005] more computing power will have to be acquired from parallel clusters. A lot of work is being done in interactive visualisation (for example [Zhang X. et al, 2002],

[Thiebaut M. 2004]), which use computational GRIDs to handle increasingly complex models and simulations.

Function-based shape modelling can be effectively applied to the current trends and developments, and has certain advantages in interactive shape modelling field:

- Ultimate precision allowing for the finest detail possible

- Guaranteed consistency of the model

- Design for free-form style interactivity

- Compactness of the representation

- Efficient parallelisation

Properties such as ease of parallelisation, small model size and the possibility of concurrent modification make function-based models suitable for visualisation over the Web, rendering on clusters of parallel computers and fast, high-quality visualisation.

The focus of this work is on using the advantages of the function-based modelling approach to create a flexible representation for free-form modelling. As function becomes more complex, efficient interactive visualisation algorithms are required. Complemented with lossless function acceleration methods, function-based representations may be applied to many areas of interactive shape modelling, including collaborative modelling, Web visualisation and distributed rendering on GRIDs.

The following is a review of the existing methods of interactive function based shape modelling and visualisation. The strengths and weaknesses of current approaches will also be demonstrated.

2.2 Functions in shape modelling

The three major types of functions used to define shapes in shape modelling are parametric, implicit and explicit.

Implicit functions define a surface with equation $f(\mathbf{p})=0$, where point $\mathbf{p} \in \mathbf{R}^3$. Explicit functions $g=f(\mathbf{p})$ define a scalar field over the $\mathbf{R}^3$ space. In this field one can either derive iso-surfaces, or consider all the points on the shape, inside, and outside it. Examples of such representations are *soft objects* [Wyvill G. et al., 1986], *blob-tree* [Galin E. et al.,

2000], *FRep* [Pasko A. et al., 1995]. Further in this thesis, the term “function” will be used with reference to implicit and explicit functions of three coordinates.

Function-based shape definitions are usually very compact, but have a serious deficiency. Rendering shapes defined with functions normally requires a lot of computation since the space is to be sampled at many points, and at each point the function is to be evaluated. If the function is complex, the resulting slowdown can make the model unusable. Therefore it is essential to develop acceleration techniques are essential to be developed for the function-based shape modelling.

In general, most of the following methods describe how to create defining function. A comprehensive survey on this and many other issues of function-based modelling can be found in [Bloomenthal J., 1994]. A more recent approach intended for animation application is presented in [Cani M.-P., 1999].

We will first look at basic function-based models based on mathematical expressions without a underlying structure, followed by another way of defining a function by interpolating it on a set of samples. This type of function definition is popular due to its simplicity and controllability. However it has its own shortcomings, which will be discussed further.

A skeletal approach to function definition exists from the very beginning of this type of shape representation. It uses simple geometric shapes like points, lines and polygons to define the function. This method offers great advantages for animation, as well as for interactive modelling, as a complex shape can be controlled by a set of defining primitives.

In skeletal modelling, the function keeps track of parts of the object via the defining geometry. On the contrary, operation tree function construction methods use simple primitives and a set of operations on them to define the final model. This approach is similar to the CSG one, but in the function-based modelling domain it offers far greater possibilities.

2.2.1 Analytical functions

A function can be defined in an analytical form to represent some mathematical objects like spheres, ellipsoids, etc. Analytically defined functions are not generally suitable for interactive modelling. However, interesting results are achieved when the function is modified to suit certain constraints. This allowed Andrew Witkin [*Witkin A. and Heckbert P., 1994*] to control the surface of implicit function with particles located on it. The method was further extended in [*Hart J.C. et al., 2002*], but it still does not provide enough freedom and flexibility.

Variational surfaces are also a form of analytical approach. Coefficients of a complex analytical expression are calculated to fit constraints, such as points on the surface of the shape, normals, and inside-outside constraints. This method can be used for surface reconstruction, conversion from polygonal mesh to implicit surfaces, morphing [*Turk G. and O'Brien J. F., 1999*] and interactive modelling [*Karpenko O. et al., 2002*]. The drawbacks of this type of model are computational complexity both of fitting and evaluating the surface, and lack of sharp edges and small features in the resulting shape.

2.2.2 Approximating function

Another way to define a function is to perform some form of its approximation by constructing it from a set of samples [*Galyean T. and Hughes J., 1991*]. These techniques resemble spatial-enumeration modelling methods in the sense that uniform or octree-like structures are used for indirect classification of space volumes. The drawback of the construction method is also similar – memory requirements, modification complexity, and blobby-like appearance of the resulting shapes.

There are many works that employ this approach for the function modelling [*Desbrun M. and Cani M.-P., 1995*], [*Desbrun M. and Cani M.-P., 1994*]. They present a number of ways to improve the original technique.

In [*Baerentzen, J., 1998*] an octree is used to store only those field values that are near the surface of the shape. Several complicated tools were also proposed for operations on shapes. This representation improves the resolution, but it is still impossible to refine it during the modelling process, in case finer details are needed.

Cani et al. used a similar approach in [Ferley E., Cani M.-P., Gascuel J.-D., 2000], although instead of values they stored the set of cells that are near the surface. The distinguishing feature of this system is that the modelling space is not limited, that is model can be extended indefinitely. It is also possible to create custom tools, extending the set of operations on the shape – any model created with the system can be used as a tool. The multi-resolution extension for this method was presented in [Ferley E., Cani M.-P., Gascuel J.-D., 2002].

Another important example of the adaptive shape representation is the ADF [Friskens S.F. et al., 2000], or Adaptively sampled Distance Fields. They represent the shape with the field of distances to its surface, which is adaptively sampled with more accuracy near zero level. The precision of this representation is rather high, although it is difficult to update the structure when the shape is altered.

[Kazakov M., Adzhiev V., Pasko A. 2001] presents an example of using the approximating representation as an intermediate level between the actual model and the final image. The Hyper-Fun [Adzhiev V. et al., 1999] FRep description language [FREPWeb] was used to define the initial model, then it was sampled and dynamically updated to provide interactive walkthrough.

2.2.3 Skeletal modelling

This approach is one of the major ones in function-based shape modelling. Here, the function is not directly specified by an analytic expression or a set of samples, but rather derived from a set of simpler primitives, called a skeleton. It has many modelling uses, from animating characters to modelling complicated life forms.

Figure 5. A transformation of a blobby object defined with two-point skeleton.

Initially, points were used for the function construction. Such shapes are known as ‘metaballs’, ‘blobbies’, etc., because of their organic look, as shown in Figure 5. Each point defined a potential, either finite or not, after which the functions were blended together through simple addition or more complicated blending functions.

[Bloomenthal J. and Wyvill B., 1990] proposed the use of simple shapes like lines and polygons as skeleton components for the implicitly defined model. This allowed for creating more complex shapes defined with fewer elements. More advanced skeleton elements like subdivision curves were used to interactively manipulate complex implicit surfaces in [Cani M.-P. and Hornus S., 2001], which was further extended in [Angelidis A., Cani M.-P., 2002]. A fast polygonisation method allowed the use of subdivision curves in interactive modelling.

Figure 6. Skeleton on the left defines the model shown on the right.

A detailed review and generalisation of the skeletal approach can be found in work on convolution surfaces [Sherstyuk A., 1997]. The general mathematical approach used in this paper allowed for the creation of realistic organic shapes from rough geometric representation, like in Figure 6. Methods for ray-tracing visualisation are also presented in [McCormack J. and Sherstyuk A., 1998].

The skeletal approach has a number of problems for both the user and the application designer. The automatically provided blending can be inappropriate, or the elements should be blended in a certain user-defined way. For example, if a humanoid-like creature is being modelled, the sections of its arm should blend with each other to form joints, while the two arms should be kept separate, even when they have a point of contact. There are works to provide proper blending, like [Angelidis A., Jepp P., Cani M.-P., 2002], and many others. A proper modelling hierarchy is a solution to the blending problem, but the representation still remains limited in many common operations, like boolean, sweeping and morphing. This limitation can be eased by keeping track not only of the skeleton and basic shapes, but also on the operations on them.

2.2.4 Operation trees

Another way to define the function for a model is to start with a basic set of shapes, and to derive the final function from them with a set of predefined operations. The definition is very flexible, as it defines all the intermediate stages along with the final shape. This approach is an extension of the CSG modelling, which is intensively used in CAD applications.

Figure 7. Example of the BlobTree. The shape on top is constructed by recursively applying operations in the nodes of the tree to their respective children.

An example of such a structure is the Blob-tree (Figure 7) [Wyvill B. et al., 1999]. It extends the CSG tree modelling approach to incorporate operations common for function-defined shapes, such as blending and offsetting. The possible operations were further

expanded in [Galin E. et al., 1999] for morphing and other metamorphosis. This representation gives the user great opportunities, making it possible to combine operations typical for conventional modelling systems with the advantages of function-based representation. An application and a toolkit utilising the Blob-tree approach was developed, and it is available at [BlobWeb].

Unlike the Blob-tree which defines operations on the shape itself, Constructive Hypervolume Modelling or CHV, described in [Pasko A. et al., 2002], operates on attributes of the shape or on point attributes. The operations are defined for multidimensional spaces, where additional dimensions can represent either the time scale or other parameters, like colour or opacity.

2.2.5 FRep

The function representation (or FRep) defines a geometric object by a single explicit real continuous function of several variables: $g=F(X) \geq 0$. This is a generalised way to define solids with real valued functions [Pasko A. et al., 1995].

In FRep, a complex object is defined by a single continuous function of several variables (point coordinates in multidimensional space). This representation can be constructed by applying different operations to primitive objects. A primitive is considered a "black box" with the defining function given by the function evaluation procedure. There is a rich set of operations closed on this representation, i.e., resulting in a continuous real function: set-theoretic operations and Cartesian product defined with R-functions, blending, offsetting, sweeping, projection, hypertexturing, metamorphosis, and extended space mapping, which combines space and functional transformations (mappings) [Savchenko V. and Pasko A., 1998]. Therefore, FRep is a more generalised model with respect to traditional skeletal implicits, convolution surfaces, distance-based models, CSG, sweeps, and voxel models, and therefore can unify them.

In addition to incorporating existing representations, FRep mathematical apparatus can express various important and difficult to represent shape modifications like implicit sweeps of arbitrary shapes [Sourin A. and Pasko A., 1996], to have an advanced control

over the operations on shapes, like controlled blending [Pasko G. et al., 2002]. Extended operations, such as morphing, offsetting, were also introduced in the FRep modelling.

Let us describe how operations for FRep models are defined. The result of each operation is also a valid FRep object. Let object G_1 have the definition $f_1(X)$. For unary operations, the object G_2 is said to be derived from G_1 as $G_2 = \Phi(G_1)$, and it is defined by $f_2 = \Psi(f_1(X)) \geq 0$, where Ψ is a continuous real function of one variable. Examples of unary operations are bijective mapping, affine mapping, projection, offsetting. For binary operations, the object G_3 is said to be derived from G_1 and G_2 as $G_3 = \Phi(G_1, G_2)$, and it is defined by $f_3 = \Psi(f_1(X), f_2(X))$, where Ψ is a continuous real function of two variables. Examples of binary operations are set-theoretic operations, blending operations, Cartesian product.

Several modelling systems use FRep for storing their geometry. Among these are applications for carving [Pasko A. et al, 2001], virtual embossing [Sourin A., 2001], and general purpose FRep model description language HyperFun. [Adzhiev V. et al., 1999]

2.2.6 Summary

Every modelling approach is suitable for its own area of application. Some are appropriate for use in animation, others in some physically-based simulations, etc. For free-form shape modelling and precise imitation of real-life crafts, an FRep-based modelling approach was selected for its natural capability to represent the widest variety of operations.

2.3 Function acceleration methods for shape modelling

Function definitions are usually very compact, but have a serious deficiency. When rendering shapes defined with implicit and explicit functions, a lot of computation is normally required since the space is to be sampled at many points, and at each point the function is to be evaluated. If the function itself is difficult to compute, the representation can become too slow to be useful. Therefore, acceleration techniques are essential for function-based shape modelling.

The acceleration methods can be classified by how they treat the internal structure of function. *Approximation* methods work without using any information about the way how functions are computed, while *constructive* methods aim to speed up functions by optimising the way they are constructed.

Approximation methods augment functions with special data structures and pre-calculated function parameters, such as sampled values, surface position, and control points to calculate the actual function value at any arbitrary point.

One of the most characteristic examples of the approximation method is the uniform grid described in [Galyean T. and Hughes J., 1991], where a defining function is initially sampled at regular intervals with consequent interpolation between the grid points. In general case, it takes a lot of space even if each cell contains small amount of data. To reduce the number of cells, it was suggested by [Ferley E., Cani M.-P., Gascuel J.-D., 2000] to keep track of the points near the surface only.

Regular grid sampling cannot represent shape details smaller than the sampling interval. Several adaptive approaches were proposed to solve this problem for both surface [Ferley E., Cani M.-P., Gascuel J.-D., 2002], and solid representations [Friskén S.F. et al., 2000]. The acceleration in these works is achieved by using octree-like structures to locate the required points as fast as possible.

Besides the space requirements, another drawback of the grid-based methods is their inflexibility. They are mostly useful to store models, rather than to modify them, especially when plastic modifications are required.

Trivariate polynomials over convex hulls (Figure 8) were used to construct implicit surfaces with desired properties by [Bajaj C. et al., 1995]. While it is impossible to devise an analytic representation for any shape, one can enclose the shape surface into a set of convex hulls, such as tetrahedra, and perform interpolation inside each of them separately, enforcing the desired boundary conditions. The polynomials have a number of useful properties, such as C^1 continuity, evaluation efficiency and they can also be easily modified using control vertices. This method was applied to wide range of applications, starting from surface reconstruction [Bajaj C. et al., 1997] to free-form design [Xu G. and Bajaj C., 2001].

The representation is appropriate for smooth interpolation of point sets, triangular meshes, as well as for interactive modelling.

Figure 8. A-patch convex hull and control points.

A-patches are designed to derive approximations with required properties out of some existing representations, and can be controlled by modifying either the properties or the initial representation.

Other approximation acceleration methods use reconstruction of function through its properties, such as surface points, normals, as well as inside and outside points. Variational surfaces are an example of this technique. They are very suitable for interactive modelling described by [Turk G. and O'Brien J. F., 1999]. However, fine details cannot be defined with such methods, and they appear to be suitable for sketching, as proposed by [Karpenko O. et al., 2002] and [Igarashi T. and Hughes J., 2003].

Constructive methods explicitly use the function's internal structure, therefore the discussion of the acceleration techniques will be augmented with reviews of functions which can be used.

Some acceleration methods are developed for gradual modification of defining functions during the modelling process. Knowing which part of the function is being updated, it is possible to avoid re-processing the entire shape, and update only the modified part of it as proposed by [Galin E. and Akkouche S., 2000].

Other acceleration algorithms work by simplifying or even bypassing elements of functions that do not contribute to the resulting shape at a given point.

Blob-tree structure described in [Wyvill B. et al., 1999] consists of a set of components, which affect only a certain part of the final function. Each component is itself a function.

By forcing these functions to conform to Lipschitz condition—such as $f(r_1) - f(r_0) < \alpha |r_1 - r_0|$, which holds for any r_1, r_0 — it becomes possible to determine the contributing components efficiently. This condition allows for guaranteeing that for any point r , such as $|r_0 - r| < R$, the function will not reach some threshold value, e.g., that a bounding box does not contain the surface. This property was combined with an octree containing contributing elements in its nodes. Thus, bounding boxes hierarchy was added to the tree.

Another acceleration technique by [Cani M.-P. and Hornus S., 2001] proposes to use a level-of-detail when evaluating the function. Different kinds of skeletal representations, e.g. subdivision curves, were used to adjust to the required level-of-detail. The subdivision skeleton was able to change its complexity to match the current detail settings. Meanwhile, [Barbier, A. and Galin, E., 2004] proposed a different approach. Every blob-tree node was augmented with the level-of-detail information. The tree was sorted according to the importance of the nodes, as shown in Figure 9.

Figure 9. Blob-tree with levels of detail. Parts of the tree which will not contribute to the image are removed from the model to simplify rendering.

Then, parts of the blob-tree, which are not important for the current viewer's location, were dynamically removed. All the operations were performed during run time, so no additional overhead was introduced.

The drawback of the described methods is the requirement that all the functions are to pre-exist in the modelling system. This requirement was removed in this work.

2.4 Interactive visualisation of function-defined models

2.4.1 Polygonal approximation

This common method of surface visualisation suits us because of its wide support in hardware, readily available approximation, and optimisation possibilities. The problem is to extract the polygonal approximation from the functionally defined shape. The classification and the description of the polygonisation methods will be presented in 2.5.

2.4.2 Point rendering

The ultimate primitive for every drawing operation are points. Lines, triangles, and many other primitives are converted into points to be displayed. The point rendering approach avoids these conversions by representing the surface with a dense set of shaded 3-dimensional points [Levoy M. and Whitted T., 1985]. Point size can be altered during rendering to avoid gaps. The points can also be organised in a hierarchical manner to handle real-time drawing of extremely large point sets. In the Qsplat system [Rusinkiewicz S. and Levoy M., 2000], points were used to progressively display and interactively manipulate meshes consisting of several millions of triangles. Hierarchical arrangement allowed for broadcasting the meshes over network [Rusinkiewicz S. and Levoy M., 2001].

A point-based modelling system PointShop3D described in [Pauly M. et al., 2003] makes use of point representation for interactive modelling, however it cannot handle multi-resolution meshes well, and also geared towards surfaces, not solid objects.

2.4.3 Visualisation methods evaluation

Our goal is to give quality photorealistic experience to the user, while allowing him/her interaction with reasonable response times. Even though ray tracing yields excellent images of the model, interactions with the model would generally fail, except for methods designed in [Sourin A., 2001].

Particle-based methods are very good at giving an idea of the shape during the modelling process, but the fine detail representation is poor and photorealism is not achieved.

Point-based and triangulation methods are similar in quality and complexity. Point-based visualisation is good at representing shapes with fine detail, while requiring sampling the number of points comparable to the ray-tracing methods. In many cases it may be preferable to take less samples and interpolate them with polygons. This is especially true if the shape has smooth or flat areas.

2.5 Polygonisation methods

There are many methods for representing the surface of function-defined shapes with polygons. In this section the classification scheme proposed in [Bloomenthal J., 1994] will be used. All the algorithms fall into two major classes:

The methods intended for operating on continuous functions.

The methods devised for processing discretely sampled data.

Algorithms designed for continuous data include *particle – based*, *predictor-corrector* and *subdivision* techniques.

Particle-based techniques were initially designed for interactive modelling. The methods for effective interaction between particles and the implicit surface were developed in [Stander B.T. and Hart J.C., 1995]. Unfortunately, the maximum number of particles that can be operated in real time is only about a few thousands. Therefore, these techniques fail to satisfy the requirement of the fine detail reconstruction.

When a high quality mesh is required, *Predictor-corrector* methods are to be sought. These methods are based on prediction of the adjacent triangles positions, with their consequent placement on the actual surface. The triangle positions can be chosen to achieve near-equilateral shape of the triangles. The recent works in the area are [Karkanis T. and Stewart A.J., 2002] and [Härtmann E., 1998]. The *predictor-corrector* method requires initial seeding triangle to be put on the surface—the condition, which could easily be met when the original shape is known. Shape modelling with local mesh updating based on the *predictor-corrector* algorithm is presented in [Akkouche S. and Galin E., 2001]. In this work only blobby objects were constructed, since the method was not suitable for models with sharp edges. Using of adaptive polygonisation leads to undersampling, i.e. omitting of fine details. Moreover, global overlap detection can be

expensive for fine meshes. Therefore, although the shape of polygons is acceptable, this method fails to reveal fine details of the model.

Subdivision mesh extraction methods are based on recursion. A cell containing the model is subdivided into eight smaller cells, each of which is examined whether it contains the surface or not. The process repeats, until the finest cell level is reached. All the leaf cells containing the surface are polygonised. Unlike the predictor-corrector methods, this algorithm does not need an initial point to begin the polygonisation. It detects the surface automatically. The mesh generated by this algorithm is similar to one generated by the Marching Cubes algorithm, but, like most of the adaptive methods, may suffer from occasional undersampling. The working area is generally limited by the starting cell size. An interactive system utilising this method was described in [Galin E. and Akkouche S., 2000]. The initial cell size restriction was successfully handled by octree growing. For reliable detection of the function-defined surface in a cell, *Lipscitz* conditions were used. This resolved the undersampling problem, but greatly restricted a possible set of implicit functions. Another point preventing the method from being utilised in our application is a poor mesh quality, similar to that of the marching cubes algorithm [Lorensen W.E. and Cline H.E., 1987].

Since the nature of function-defined models is continuous, only considering continuous methods should be enough, however the algorithms initially designed for discrete data can be successfully applied to polygonisation of continuous functions as well. Among the most common methods are *spatial enumeration* and *continuation* techniques.

Spatial enumeration is probably the oldest algorithm for polygonising volume data [Lorensen W.E. and Cline H.E., 1987]. It is perfectly suitable for range data from CT or MRI scans, but for interactive visualisation of a large function-defined model, the uniform sampling requirement may become unfeasible due to the high complexity of each function evaluation. Because of the assumption on the model topological connectivity, the search through all the volume becomes redundant.

The second approach named *continuation* and presented in [Wyvill G. et al., 1986], exploits spatial coherence for detecting all the cells intersecting the surface. Initially, a

seeding set of cells is introduced. Then, at each step the neighbouring intersecting cells are examined. To avoid checking the same cell twice, a sort of hashing can be introduced for the cells. The hashing can be based, for example, on integer coordinates of the cell corners. The demerit of the continuation method in comparison to spatial enumeration is in its lack of generality since it requires to place seeding points on each separate surface defined by the implicit function. Fortunately, in an interactive application it can be easily implemented. The continuation algorithm detects the surface effectively without unnecessary function evaluations. Also, due to its discrete nature, it is very fast and can be extended for quick local re-polygonisation. It must be noted, though, that the mesh quality is highly dependent on the way every cell is polygonised, configuration of the cell and optimisation algorithms used. These issues will be addressed in the section 4.3.

2.6 Haptic rendering for function-based models

When a function-based model is visualised, one has to perform function queries uniformly, to reveal the surface or internals of the shape. Unlike the visualisation process, haptic rendering is essentially a local operation, which requires fast evaluations in or near the point of contact. Apart from complex situations involving the probe of finite size [Otaduy A. and Lin C. 2003], haptic feedback problem is reduced to calculating the reaction force in each point r .

The locality can be efficiently exploited for optimisation purposes, as it was shown by Avila in [Avila, R. S. and Sobierajski, 1996] and [Avila, R. S. 1998]. The function-based models allow for easy point classification, normal computation and are inherently well-formed in terms of topology.

A common method for the haptic rendering of function-based models is to use force proportional to the value of the function, imitating elastic response. For smooth performance, force computations are to be performed at least 1000 times per second, which can be a problem for complex functions. Different methods were used to accelerate function evaluations, for example replacing geometry with a local plane, or create localised polygonal representation.

In [Avila, R. S. and Sobierajski, 1996] haptic rendering and modelling was performed by transforming a voxel model to implicit representation and performing touch feedback using this model. [Huang, C. et al., 1998] used direct interaction with function to calculate direction and magnitude of feedback.

2.7 Summary

Interactive shape modelling is a very active and rapidly developing area of computer graphics. Advances in technology enabled the algorithms and methods once considered impractical, to be used in commercial products. Function based modelling is a promising technology that found limited use in commercial applications, but received a lot of attention from researchers.

Function-based modelling provides benefits both for the end-user of the system and for the designer of the modelling software. The advantages include high precision, compact representation well suited for transfer over network, simplicity of operations and easy topology handling. Therefore, applications to distributed visualisation and modelling should be considered. Users of such systems will find them more intuitive,

On the other hand, there are challenges to face: robust visualisation and construction of the function with desired properties. These issues have been addressed extensively, but often through sacrificing precision or interactivity. Visualisation and surface extraction algorithms intended for interactive operations like particle-based algorithm do not provide a sufficient resolution for the model. Function construction methods often limit themselves to creation of blobby shapes, or making an approximation, which results in the lack of sharp edges and loss of fine details. Other methods are generally not intended for interactive use.

Free-form modelling with functions allows the introduction of haptic interfaces, and inherent space classification property of the functions becomes very useful for this task. Combined with methods for distributed processing for in order to handle more complex objects and remote visualisation techniques, function-based models have incredible potential for research and applications.

Chapter 3. INTERACTIVE FUNCTION-BASED MODELLING FRAMEWORK

We consider shape modelling as a process of creating the final object by gradual local shape modifications. By “local” we mean that the area of each individual shape modification is significantly smaller than the overall size of the shape. We use functions to represent shapes and their properties. Therefore each modification can be represented with an operator transforming a function defining the current shape into a function defining the modified shape. Section 3.1 presents a high-level overview of the modelling pipeline. In Section 3.2 we explain what kind of functions we use to define shapes and their properties and introduce operators implementing shape modifications. Section 3.3 describes an acceleration technique based on operator clustering.

3.1 Overview of the modelling process.

Modelling process consists of modifications applied to an initial shape. The initial shape is defined by its function. Modifications change one shape into another, that is, one function into another, which can be expressed as operator. Defining an operator means defining a way to calculate the modified function at every point, which means defining a special function, called operator kernel.

Each modification is made using some *tool* applied along a *tool path*. This tool and path are used to define the operator and its kernel for the modification.

The final shape is obtained by applying all the modifications. Similarly, the function representing the final shape is obtained by applying all the operators, i.e. calculating their kernels. One shape could require thousands of modelling operations to build. If no measures are taken, at each point of the final function one has to calculate thousands of operator kernels, which is not feasible in most cases.

Most modifications change shape only in some limited area. The area is defined by the tool size and its path. The part of 3D space where modification takes place is called *area*

of influence. Any specific point is covered by a limited number of areas of influences, while other modifications do not contribute and can be safely omitted (for this particular point). Therefore, a big number of operator kernels can be significantly reduced.

Areas of influence are derived from the tool and tool path data. They are difficult to index directly, and even though we can skip most of the operator kernel evaluations, we still have to check many areas of influence. To solve this problem, we group the areas of influence together into easily indexable groups called operator clusters.

Figure 10 shows the data flow of the function-based modelling process.

Figure 10. Function representation pipeline.

In the following, we are going to give a more thorough and precise description of the proposed function-based shape modelling method.

3.2 Operators for shape modelling

Every point of the modelling space has some properties associated with it. These could be material density, colour, transparency, shininess, temperature, etc. If the geometric location of a point in the space is defined by three coordinates from $\mathbf{R}^3$ and the set of all possible properties is defined as $\mathbf{P}^n$ then any possible shape in the space can be defined by function $S:\mathbf{R}^3\rightarrow\mathbf{P}^n$. The set of properties $\mathbf{P}^n$ depends on the modelling system being designed. For example, we may decide that the functions values $g\geq 0$, $g\in\mathbf{R}^1$ correspond to the points belonging to the shape. To associate colours with the points, we can choose $\mathbf{P}^n:\mathbf{R}^1\times\mathbf{C}$, where $\mathbf{C}=[r\times g\times b]$ is a colour space.

There are many ways to visualize the defining function. Generally, visualization involves repeated evaluation of the function near the surface of the shape and deriving graphics primitives, such as triangles, voxels or pixels from this data.

Let's assume that the original shape is defined by function S_0 while the modified shape is defined by function S_1 . Then, a transformation from function S_0 to S_1 can be expressed as $F: S_0 \rightarrow S_1$, where F is an operator on functions. The final shape defined by function S_n is obtained from the original shape through n consecutive interactive modifications:

$$F_n \cdot F_{n-1} \cdot F_{n-2} \cdot \dots \cdot F_2 \cdot F_1 S_0 \rightarrow S_n \quad (1)$$

where F_i are operators on defining functions $S_i: \mathbf{R}^3 \rightarrow \mathbf{P}^n$, $i=[1, n]$.

S_0 is the initial function which defines the original shape. It is defined in the beginning of the modelling process.

We will define operators F by their *kernel function* K and the modification parameters P :

$$F_{i+1}: S_{i+1} = K_{i+1}(P_{i+1}, S_i) \quad (2)$$

Parameters P_{i+1} define a particular shape modification and may include parameters of the geometry representing the tool, colour which is to be deposited to the shape, path of the tool and force of its application. Function K_{i+1} defines how exactly to modify function S_i with these parameters. For example the same tool can either add or remove material from the original shape.

To obtain the final function, kernel functions have to be calculated sequentially, as shown in (1)

Let us give some examples for operator definition. For the sake of example, assume geometric shapes are defined by functions $g=S(p) \geq 0$. Shape modifications will be defined by parameters $P=\{R, \mathbf{r}(t)\}$, which are size R of the tool and its path $\mathbf{r}(t)$. In our examples, the path is represented with a set of line segments, hence the discontinuities of the strokes seen in Figure 11.

The first operator implements a tool which deposit material by sweeping a sphere of radius R by the surface of the shape along the path $\mathbf{r}(t)$ (Figure 11(a)):

$$F_1: S_1(\mathbf{p}) = \max(R - |\mathbf{p}_0 - \mathbf{p}|, S_0(\mathbf{p})) \quad (3)$$

where $\mathbf{p}_0$ is a point on $\mathbf{r}(t)$ closest to $\mathbf{p}$.

Next, we will change the kernel function K to make a new operator performing a plastic deformation of the shape within width R along path $\mathbf{r}(t)$ (Figure 11(b)):

$$F_2: S_1(\mathbf{p}) = R e^{-(R - |\mathbf{p}_0 - \mathbf{p}| - R)^2} + S_0(\mathbf{p}) \quad (4)$$

where $\mathbf{p}_0$ is a point on $\mathbf{r}(t)$ closest to $\mathbf{p}$.

Figure 11. Examples of different operators.

(a) Sweeping a sphere by the surface (b) Surface deformation (c) Carving.

In operators (3), (4) the shape function S_0 is evaluated only at point $\mathbf{p}$, however more complex operators may require evaluations at several points for each given $\mathbf{p}$. For example, carving a shape with a cutter of width $2R$ and the incision depth of R (Figure 11(c)) can be defined as:

$$F: S_1(\mathbf{p}) = \min(S_0(\mathbf{p}), -S_s(\mathbf{p})) \quad (5)$$

where $S_s = (R (1 - e^{-S_p^2}) - |\mathbf{p} - \mathbf{p}'| - |\mathbf{p}' - \mathbf{p}'_0|)$, S_p is the distance along the path either from $\mathbf{p}_0$ to $\mathbf{r}(0)$ or from $\mathbf{p}_0$ to $\mathbf{r}(t_{\max})$, whichever is smaller, $\mathbf{p}_0$ is a point on $\mathbf{r}(t)$ closest to $\mathbf{p}$, and $\mathbf{p}'$ and $\mathbf{p}'_0$ are points on the surface of the shape closest to $\mathbf{p}$ and $\mathbf{p}_0$, respectively. To calculate $\mathbf{p}'$ and $\mathbf{p}'_0$, we have to perform additional evaluations of S_0 for any given $\mathbf{p}$, as seen from (5).

3.3 Operator acceleration

Since the number of operators in (1) can be very large, the complexity of the representation becomes $O(n)$ or worse, where n is the number of modifications.

A common way to solve this problem is to replace all the operators with just one which interpolates the shape based on interpolation from sampled points $S: \mathbf{R}^3 \rightarrow \mathbf{P}^n$. However, this method normally creates artefacts and significantly increases the size of the model.

An efficient acceleration structure can be proposed since we mostly consider local modifications and only a little percentage of all the operators in (1) will contribute to any given point of the shape. Hence, we can confine every operator F to the respective area of influence ω as follows:

$$\text{For all } S_0, S_1, F: S_0 \rightarrow S_1: \text{ if } \mathbf{p} \notin \omega, \text{ then } S_0(\mathbf{p}) = S_1(\mathbf{p}) \quad (6)$$

According to (6), when $\mathbf{p}$ is outside the area of influence ω the whole operator F can be skipped.

To illustrate it, let's consider modifications with spherical areas of influence. In Figure 12, areas of influences of operators are shown as bounding boxes. If no acceleration is performed, then all several hundreds of operator kernels have to be taken into account, as shown in Figure 12(a), which would considerably slow down the modelling process. Figure 12(b) presents an example of accelerated evaluation, where only relevant operator kernels are computed.

Figure 12. Acceleration by skipping non-contributing operators.

Function evaluation at the point specified by the arrows. Operators that were taken into consideration are marked with bounding boxes. (a) Non-accelerated evaluation. (b) Accelerated evaluation.

A straightforward approach to implement the acceleration would be to perform a spatial classification of the areas of influence ω_n , and for every $\mathbf{p}$ to check only $F_n: \mathbf{p} \in \omega_n$. To use areas of influence efficiently, we introduce the concept of *operator cluster*. Operator cluster is a set of operators contributing to a certain area Ω . To create a cluster, we select all F_n for which ω_n intersects Ω . The respective operators F_n are associated with the cluster. The operator clusters serve the purpose of spatial identification of the evaluation areas, as they allow for efficient indexing. They are used to determine what ω_n are to be checked according to the specific evaluation needs.

Figure 13 shows the overall structure of the modelling system described above.

Figure 13. Layout of the modelling system.

3.4 Summary

In this chapter a practical interactive modelling framework was proposed. The framework allows for implementing a very broad set of operations. The provided approach offers a flexible way for representing a function-based model during the course of interactive modification. However, to make the framework usable for real applications, special measures are to be taken.

Initially, no restrictions were imposed on the interactive operations. In this case no acceleration strategies could be applied, as well as the operations themselves could be expensive to perform. To workaround this a localisation property was proposed for each operation, and the dependencies were explicitly specified. In particular, this allowed defining the operations on functions simply as functions themselves, which is beneficial for the application.

The localisation property introduced to the operations has made it possible to employ different acceleration algorithms. The proposed algorithms are based on the basic idea of excluding non-contributing operations from consideration.

The overall result from implementing the proposed methods and models is substantial. Many interactively modelled shapes may include thousands of separate interactive

operations. Without any acceleration all of them would have to be applied to get a function value. In most cases, it is possible to reduce the number drastically, thus allowing for further interactive extensions of the model.

Chapter 4.

GENERAL DESIGN AND IMPLEMENTATION ISSUES

In this chapter, we discuss what steps are to be taken to create a function-based shape modelling system. First, we discuss what considerations are to be taken into account, and then we describe each stage of the design.

4.1 Design and implementation workflow

To create an interactive shape modelling application based on the algorithms and methods introduced in Chapter 3, the following steps should be followed:

- Define functions to represent shapes
- Define operators to represent shape modifications
- Devise acceleration algorithms for the operators
- Develop visualization algorithms suitable for interactive applications

Let us discuss what should be considered for a successful implementation of these steps.

We have proposed a set of essential components for a function-based modelling system, including the following:

Path is defined directly by the user and defines most interactive aspects of the system, what kind of operations it accepts, and how they are represented.

Path provides a local coordinate system for *Kernel* function to define *Operator*. Kernel function defines how to modify the original function to obtain the resulting one.

Area of influence is derived from path to guarantee the limits of the function.

Cluster is method to handle areas of influence. When the areas of influence are derived from a path, there is no way to index them for fast access, as the areas are defined by functions. Cluster module provides generation of clusters suitable for indexing algorithms

out of areas of influence. Each cluster is a simple and straightforward structure such as a sphere or an axis-aligned box, and it contains one or more areas of influence.

Indexing Structure uses clusters to perform fast searches to find out what areas of influence are to be checked in the current point or in certain vicinity. Underlying algorithms depend on implementation and use – when one anticipates uniformly distributed modifications, it is sensible to use a lattice for lookup, if the distribution is not specified then algorithms like R-tree are more suitable.

- *Function* uses operators and indexing structures to choose which operators to use, and applies them one after another, providing data for shape rendering.

Figure 14. Function structure

Function is the main interface for the feedback loop of the modelling system. It is used to perform queries on the model, which in turn provide visualization and haptic feedback to the user.

Component interconnection is shown in Figure 18. On top user input is shown, which provides a modification path and a modification tool, expressed by a kernel function.

Visualization uses function to derive graphics primitives, as well as it may optionally use areas of influence to render shape updates.

4.2 Principles of shape functions definition

First, one should choose the domain of the functions representing the shape. This is the same domain where all future shape modifications will take place (e.g., either the whole $\mathbf{R}^3$ or some limited region of it).

Function defines the properties of every point of domain. For example, if we are interested in physical simulation, we might want to know material density, temperature, elasticity. For optical properties we are interested in opacity, refractive index, colour, etc. For each point, the shape-defining function yields a vector of values which we are interested in. Formula (7) shows how a function can define a shape model:

$$S(\mathbf{r})=\mathbf{v} \quad (7)$$

Where $\mathbf{r}$ is in domain, and $\mathbf{v}$ – vector of the properties we are going to model.

The appropriate visualisation methods should be chosen for the selected properties. For an interactive application, visualisation methods should be capable of producing real-time display of the properties. Additional visualization methods for off-line high-quality rendering should also be considered.

User perceives the function-based model via a generated image or a haptic feedback. Haptic feedback or visualization can require different parameters to generate, which can be encoded as a result of the function. For example, if Phong illumination model is used for visualization, then function can generate its parameters directly, allowing to change colour, opacity, or shininess. Similar possibilities exist for haptic feedback – the function can provide elasticity of material for every point of the model.

4.3 Essential issues of operator definition

To define an operator it is necessary to define how we are going to calculate it.

During interactive modelling, changes to a shape are introduced using tools, which are usually operated with hand. As hands usually move along a path or just touch the object at different points, it is natural to use *path* notion to define modifications. The operator is defined with operator kernel function. For each point in space, it is necessary to provide a tool coordinate system for the nearest tool position, and the amount of path that it travelled to reach this position.

Additional information might be needed in certain cases, such as whether the point is inside the loop or outside, which would allow special processing for areas inside the path; however, this additional information can be replaced with a user interface feature, generating several modifications inside the path.

For several sampling points A , B and C the tool coordinate system for a modification is shown in Figure 14.

Figure 15. Defining the tool's coordinate system

Operator kernel uses tool coordinate system and position along the path, along with static parameters, to calculate new function value vector from old one.

Operator kernel is designed to make a correct modification, without regarding whether it is confined to a specified area or not. A function which faithfully represents the desired modification within an area of influence can produce some artefacts further away. That is why every operator definition has to be accompanied by a *guarding function*. For each operator, there is ω , which contains the area of influence ω . $\omega \in \omega$. We will define a guarding function f_{guard} with the following properties:

- $f_{guard} - C^1$ – continuous.
- $f_{guard}(\mathbf{p})=0$ for $\mathbf{p} \notin \omega$
- $f_{guard}(\mathbf{p})=1$ for $\mathbf{p} \in \omega$

The following formula guarantees that the operator does not have any effect on the shape outside of its area of influence: $S_{n+1}(\mathbf{p})=S_n(\mathbf{p})$ if $\mathbf{p} \notin \omega$:

$$S_{n+1}(\mathbf{p})=(1-f_{guard}(\mathbf{p})) S_n(\mathbf{p}) + f_{guard}(\mathbf{p}) S'_{n+1}(\mathbf{p}) \quad (8)$$

Even if S'_{n+1} differs from S_n outside of the intended area ω , formula (8) allows us to guarantee that $S_{n+1}(\mathbf{p})=S_n(\mathbf{p})$ for any value of $\mathbf{p} \notin \omega$. The formula is illustrated in Figure 15.

Figure 16. One-dimensional example of a modification in $[-1,1]$ interval.

(a) The graph shows how the function being modified is multiplied by zero within the interval. (b) Final function result is multiplied by (b), which smoothly becomes zero outside of the interval to avoid any modifications outside the area of influence.

Areas ω and ω' are specified in terms of the distance from a point to the path of the tool. If, for point $\mathbf{p}$, the distance $l < d$, then $\mathbf{p} \in \omega$, and if $l < d'$, then $\mathbf{p} \in \omega'$. This definition of ω and ω' allows us to define f_{guard} using f_{def} (9) in the following way:

$$f_{guard}(\mathbf{p}) = \begin{cases} f_{def}\left(\frac{l-d'}{d-d'}\right) & l > d' \\ f_{def}(0) & otherwise \end{cases} \quad (9)$$

Figure 16(a) illustrates how a complex modification can be guarded with this function. The simpler modifications in Figure 16(b) allow us to avoid evaluating the complete function. If we define f_{guard} for each section separately (Figure 16(c)), we can evaluate only the required part of the function. In certain cases, we can achieve n -fold increase in the evaluation speed, where n is the average length of the continuous modifications. The

acceleration method is very general and makes little assumptions on the internal modification structure.

Figure 17. Different approaches to defining the areas of influence for different modifications.

Shaded area denotes ω dashed line - ω (a) Guarded modification. (b) Common guard. (c) Separate guards.

Besides guarding, the guarding function can also help to create operators. When a tool is applied to soft material, the trace exactly replicates the shape of the tool. Outside the area of contact, the shape should remain intact. To ensure this, we just combine a function representing the tool S_t and a guarding function for it, thus efficiently obtaining the desired result without any need for CSG or similar modifications. Figure 17 demonstrates the cross-section of such a modification.

Figure 18. Smooth transition from tool to shape, using the guarding functions.

(a) Original Shape. (b) Cutter. (c) Resulting profile.

4.4 General acceleration algorithms and data structures

Many shape modifications can be modelled by evaluating shape functions at only one point. In this case, a simple form of acceleration can be used, such as direct lookup of the areas of influence.

If evaluation is required at several points then more complex algorithms are needed, as even single query can generate many sub-queries and it is impossible to re-check areas of influence again and again. Instead, spatial indexing structures such as R-trees determine what clusters are required for a given point.

While areas of influence can be conveniently defined with functions, it will be quite difficult to intersect them with each other, and this operation is important in determining an optimal sequence of the operator application. To solve this problem, a bounding volume should be chosen, for example a sphere or an axis-aligned box. As shown in Figure 16(c), it is often optimal to use several bounding volumes for one operation.

Bounding volumes are an efficient means to detect what operators should be applied at a point. There exist well-studied area of research on space indexing, which deals with organizing bounding volumes for efficient processing of different queries, such as “what volumes are intersected with this one”. R-tree is an efficient indexing algorithm for spatial data. It generates clusters and provides lookup method, to locate the areas of influence.

As most of the evaluations are performed in a space-coherent manner where the evaluations take place close to each other, we are exploiting the coherence with the help of the operator clusters.

The clusters stored in the algorithm-dependent data structures enable the algorithm to select the relevant operators at each point. The relevant operators are efficiently derived from the indexing structure described above.

A single function evaluation at point $\mathbf{p}_0$ may yield an operator cluster for that point by collecting the $\{F_i\}$ that should be applied in this area. Therefore, any consequent evaluations near point $\mathbf{p}_0$ will bypass calculations of the functions not associated with the cluster. This is useful, for example, when binary subdivision for two points— $\mathbf{p}_0$ outside the shape, and $\mathbf{p}_i$ inside the shape—is performed to find the point on the surface. First, we create cluster Ω in such a way that $\mathbf{p}_i, \mathbf{p}_0 \in \Omega$. Then, for all further evaluations we can use a subset of $\{F_1 \dots F_n\} \in \Omega$, and achieve an acceleration without any overhead.

Most of the function evaluations take place inside the current clusters, because the cluster radius is larger than the propagation step. Therefore, on average, the

computational complexity reduces to the cost of applying the operators inside the cluster. For many typical shape modelling scenarios this is a significant improvement. In the following the component model of the modelling system is presented.

4.5 Visualization considerations

Along with many advantages of the function-based shape modelling, there is a problem that often makes the use of functions in shape construction infeasible. Whatever the modelling results may be, they are to be displayed interactively during the modelling process and with the highest possible quality afterwards.

To perform interactive modifications, i.e., at most 2-3 seconds per modification, one has to come up with a way to perform fast updates to the generated image. If the overall generation could be made interactive, then this problem would be solved, as one would just redraw the whole image. In the most cases, image generation is too slow, so one has to perform a local update.

Most modern graphics cards use the following rendering primitives:

- *Pixels* are 2D points on the screen. They can be obtained from a function-based model directly, using ray tracing algorithm.
- *Polygons* can be drawn by any 3D card, but cannot be directly derived from a function-based model.
- *Points* in 3D are more efficient than polygons for drawing objects with fine details, where the size of polygon approaches that of point. There is no need for rasterization, and transfer overhead is also smaller.

If global updates are not required, then ray tracing is a good method for providing interactive response. One can just re-trace the part of the picture that has changed.

Most of the times, however, users are interested in polygonal or point-based models, as they can be easily rotated and relighted without any changes to the model. Generating polygons or 3D points from a function-based model takes a lot of function evaluations and can be very time-consuming.

The following requirements for an algorithm generating 3D primitives should be met:

- Function evaluation should be performed as little as possible.

- There should be a way to update some part of the model without changing the rest.

In the following Chapter we are going to describe how these requirements can be met.

4.6 Summary

We have proposed the methods for efficient construction of the operations for the function-based models. They are aimed at two different but related goals: first, one has to faithfully represent shape deformations, taking into account properties of material, type of the tool and similar parameters. Then, we proposed a way to ensure compact support of the proposed operators without placing explicit restrictions on the operator itself.

Taking into consideration the proposed function-based models, we have defined the requirements for rendering. A set of proposed rendering methods meeting the requirements will be presented in the next Chapter.

Chapter 5. INTERACTIVE RENDERING

Along with many advantages of the function-based shape modelling, there is a problem that can make the use of functions in shape modelling infeasible. Whatever the modelling results may be, they are to be displayed interactively during the modelling process and with the highest possible quality. In this Chapter a set of methods and techniques for interactive rendering of the function-defined shapes will be presented. Robust response to modifications will be augmented with a consequent fine quality rendering. The algorithms will rapidly extract a polygonal surface approximation from the function-based model for rendering on the modern fast polygon-drawing hardware.

Most common method for rendering function-based shapes is polygonisation. Our polygonisation method is suitable for efficient surface extraction. The extraction method minimizes the number of function evaluations, so it is useful for computationally intensive models.

As models grow bigger, one processor becomes insufficient for interactive updates. We have proposed a scheme to perform interactive rendering of function-based model using a cluster of computers. This involved adapting propagation model.

Finally, we describe a method for haptic rendering for our model.

5.1 Polygonisation

Most common method for interactively rendering shape models is to convert them to polygons and render them through traditional graphics hardware. For polygonisation, we take the most efficient propagation approach and adapt it to work with significantly more efficient tetrahedral cell instead of the standard cubic one.

5.1.1 General characteristic of the method

There is no ideal method for converting the surface of a function-defined shape to a polygonal representation. Each method suits best for certain cases, and behaves poorly otherwise.

To make optimizations possible, we impose a set of natural restrictions on the model. *A surface of the shape is presumed to be continuous*, which means that the function is to be continuous too. Discontinuities in the derivatives, on the other hand, are possible and should be properly handled by the algorithm. Then, the information on the topologically disconnected parts of the shape should be provided, i.e. *at least one point on the surface of each topologically independent part of the shape* should be known. This requirement is rather difficult to meet in general, but the described methods are primarily designed to work with interactively built shapes, therefore the condition can be met always. No other restrictions are imposed on the topology or on the surface of the shape.

Many surface extraction algorithms such as [Karkanis T. and Stewart A.J., 2002] and [Hartmann E., 1998] use the adaptive approaches to minimise sampling, aiming to pay more attention to the higher detailed areas. The adaptive approaches usually generate better meshes, because flat areas are represented with fewer and better shaped polygons. On the other hand, such an approach has its shortcomings for our case. As no assumptions on the surface are made, we cannot reliably detect the areas of the shape which do not have fine details. In terms of sampling efficiency, adaptive algorithms are no better than non-adaptive, as we still have to sample the function at predefined resolution uniformly.

Another problem with adaptive approaches is that it is very difficult to update the resulting mesh locally without complete regeneration. There are works, however, which have attempted to design such an update method, but it is generally slow, and fails if the polygon is larger than the area to be modified. Besides, the method only works with certain type of functions.

Uniform sampling without mesh simplification will be used in the described method because of the issues mentioned above. The most general uniform method is the Marching Cubes algorithm [Lorensen W.E. and Cline H.E., 1987]. It uses a cubic grid to search through the volume, and every cell that is found to intersect the surface generates one or several polygons. As all surface parts are known in advance, there is no need to search through the entire volume to detect them. This enables us to adopt the continuation

strategy for the surface extraction algorithm, so the entire surface will be traced starting from the known points somewhere on the surface.

We propose to use cell-based continuation algorithm. The space is subdivided into cells, and the algorithm has to locate all the cells intersecting the surface, which cells can be further used to approximate the surface with a triangular mesh or a point set. The surface search begins from a seed cell that is known to intersect the surface. As the function is continuous, its values in the corners of the cells allow us to tell which of the adjacent cells also intersect the surface. The process is repeated until all the intersecting cells are discovered.

5.1.2 Polygonisation pipeline

Common graphics hardware can interactively handle millions of polygons at interactive rate. For uniform mesh, this amount of polygons is sufficient to give a general idea of the shape and to reveal details up to several pixels in size. To detect finer details, higher resolution has to be applied, which can be neither extracted nor displayed in real time. The problem can be resolved by simultaneously extracting meshes for two resolution levels – *fine* and *coarse*. Since the size of the polygonal mesh obtained for the *fine resolution* will be substantial, this mesh can be stored and maintained in a secondary storage, such as a hard drive, while the polygons for the *coarse resolution* and the respective control data will be stored and maintained in the RAM.

The *polygonisation pipeline* works as follows. First, for the initial function-defined shape, the coarse resolution mesh is created and rendered on the screen. Concurrently, the fine resolution mesh is created in another thread. The fine mesh is stored on the hard drive, and rendered on the screen, gradually replacing the coarse resolution image. If the shape position or lighting changes, the coarse resolution mesh is used to render the moving shape while the fine resolution mesh is rendered as soon as the shape comes to rest. When any interactive modification is to be done, the defining function is modified first, followed by fine and coarse re-polygonisation of the respective area. Only the modified polygons from the fine mesh are used for updating the image on the screen while the coarse resolution polygons are stored for future use. During rendering, only the

modified part of the image is updated, while the rest of the image remains displayed on the screen. This creates an illusion of interaction with the whole shape using a high resolution. Both meshes are updated and available for further rendering thus providing the sustained interactive rate.

Fine resolution polygons can be replaced with points, but it is equivalent to pixel-sized triangles.

Below, the implementation details of the proposed rendering method are described.

5.1.3 Cells used for propagation

The choice of a cell for continuation algorithm is very important. The following issues are to be considered:

The cell should provide sampling as uniform as possible to use the function samples in the most effective way. The sampling can take up to 90% of the running time, so this criterion is of a very high importance.

The important criterion is how easy the surface can be extracted from the cell. For the continuation method it is important to find the adjacent cells effectively, as it is a very common operation.

Finally, one more requirement is to be mentioned: for an interactive application it should be possible to detect a cell from a predefined lattice containing an arbitrary point. It is essential as the polygonisation process starts to operate from a certain point on the surface, where the cell is not known.

Usually, a cubic cell is used in continuation algorithms. Although extracting of the surface from a cubic cell is more complicated than from a tetrahedral cell, cubic cells are easy to handle, sampling seems to be rather uniform, and it is very easy to locate the neighbours (See the theoretical study in [Allgower E. and Schmidt P., 1989] and [Dobkin, D. A. et al., 1990]). To simplify the surface extraction, a cubic cell is often decomposed into 5 or 6 tetrahedra. We decided to explore viable alternatives, to use tetrahedral cells directly, without dealing with cubes, and select a cell that suits our criteria best.

The search was limited to tetrahedra that tile the space by mirrored copies of itself. The goal was to avoid dealing with tetrahedra of various types during the propagation

process, as it would definitely decrease the uniformity of sampling and the mesh generated. Moreover, it was decided to limit the search even further to tetrahedra that fill the space by mirroring over their faces. This quality would enable us to make searches for the neighbouring tetrahedra as efficient as for cubic cells. All the possible mirror – filling cells for all possible dimensions were explored and classified in the mathematical work [Coxeter H. S. M., 1934]. In the classification, 3-D counterpart of a square is a cube, and for equilateral triangle it is the tetrahedron displayed in Figure 19. (Equilateral pyramid does not fill space by itself). This tetrahedron is generated by a body-centred lattice in a way shown in Figure 20.

Although from common sense it might seem that the cubic cell is the most uniform one in terms of sampling, this is not really so. The body centred lattice has been recently used to extract iso-surfaces from volume data. As it was shown in [Chan S. L. and Purisima E. O. 1998], the lattice possesses the most uniform sampling of all regular lattices. There, the cell was used to extract surfaces from volume data. The method was improved with mesh simplification in [Treece G. M. et al., 1999]. There are also some comparisons, clearly showing superiority of the body-centred lattice for dealing with iso-surfaces in terms of sampling efficiency, for example [Skala V., 2000] and [Theul T. et al., 2001].

Figure 19. Two different cell types and their lattices.

(a) Cubic lattice. (b) Proposed tetrahedral lattice

For the cubic lattice, the ratio between the most distant and the closest vertices in a cell is 1.73, while for the body-centric lattice it is only 1.15. For an absolutely uniform lattice, this parameter would be equal to 1, but this is not possible, as equilateral tetrahedra do not pack 3-D space. It was noted that the lattice is more sensitive to small details, and the resulting mesh reflects the fine features of the original shape better than others. Several recent works have used this lattice for surface extraction from volume data, although, to our knowledge, no one has used it for continuation algorithms.

Figure 20. Proposed optimal cell.

The task of finding neighbouring cells for our tetrahedron is even simpler than that for other tetrahedra filling space by reflection. In general, for this type of space packing cells, the search for a neighbour is reduced to simple mirroring of a vertex through the opposite face. In the case of the proposed cell configuration, this operation simplifies to a 2-D mirroring over an edge, like shown in Figure 21(a). The new vertex is found through a simple formula $A+B-D$ (Figure 21(a)). This requires even less computation and makes the operation identical to cubic cell in terms of complexity. For clarity, the 2-D analogy to the cell mirroring process will be used, as shown in Figure 21(b). The search algorithms presented below can be understood more easily if described in the lower dimension. The cell shown in Figure 21(b) also fills the space by reflections, and its neighbours are easily found through flipping.

The ability to quickly locate a cell in the lattice containing a specified point is essential for the continuation algorithm, especially if it is intended for interactive use. Initial conditions are most frequently specified as a point on the surface, from where the polygonisation should start. Interactive algorithms may face several initial points on the same surface, and the propagation process should form a seamless common surface. This

is natural for cubic cells, but the selected cell tiles the space with 12 orientations of itself. So some reliable method to locate the cell is to be derived.

Figure 21. The process of finding the neighbour cells through mirroring.

(a) Cell mirroring. (b) 2-D analogy/

There are several methods to locate the cell, starting from the recursive subdivision of larger cell to the examination of all 12 possible variants for the given point. A more efficient method was adopted. To locate a cell, one should locate its four vertices. The first vertex can be located immediately – it is the centre of the original cube, to which the seeding point belongs. Figure 22 shows a 2D analogy, where initial point is denoted as S , and initial cell is $A_1A_3A_4A_2$, yielding initial vertex B_3 . For the next point, there are 6 candidates – centres of the neighbouring cubes. The closest one is selected, using the distance data already obtained in the search for the original cube. For the 2D analogy, closest edge is A_3A_4 , so the second point is B_4 , the centre of $A_3A_5A_6A_4$. If some of the neighbouring cubes happen to be equally distant, the point O is on the edge or on the surface of the cell. In this case, the point is slightly offset to avoid ambiguities.

Figure 22. Locating the tetrahedron containing a given point. (2D analogy)

Now, when the 2 points are known, we need to find out the remaining 2 points. We use the symmetry of the cell and repeat the procedure, replacing the original grid with another grid with vertices in the centres previous grid's cells. In turn, vertices of the original grid become centres of the derived one. Thus another two points of the same cell are obtained, and the cell is found. For our 2D analogy, this is point A_4 , the centre of $B_3B_5B_6B_4$ in Figure 22. The only issue left to is that if a displacement was applied in the second step, the first one has to be repeated as well, because the displacement could change the previously obtained two points. The method is four times more computationally expensive than that for cubic cell, but this difference is amortised by other operations.

Figure 23. Detection of the correct seeding tetrahedron.

Continuation process places additional requirements on the cell localisation algorithm. If the cell is a seeding cell on the surface to be polygonised, there should be at least one vertex inside the shape, and at least one outside. This is the required condition to be able to find neighbouring cells intersecting the surface. There can be cases, like shown in Figure 23, when the initial point supplied to the polygoniser is located in an unsatisfactory cell. In this case, an initial step should be taken to find the valid cell for the polygonisation to start. The search is performed by propagating the cell in every direction until the satisfying tetrahedron is found. In Figure 23, the initial neighbours of the original tetrahedron ABC did not yield the valid cell. The propagation process continues to the tetrahedron $A'B'C'$, which appears to be valid. Such a process guarantees the surface detection even if the initial seeding were not precise.

5.1.4 Cells detection

The polygonisation method consists of two stages. First, all the cells intersecting the surface are detected. The criterion for this stage is simple – all the cells that have some of their vertices inside, and some outside of the shape, are to be detected by propagation.

Next stage is required to derive the actual triangulated surface from the detected cells.

Data structures and basic propagation

The main principle of the search for the cell can be described as ‘advancing front’. The front consists of newly detected cells. Initially, the front consists of only one, seeding cell. This cell is processed to detect its neighbours, which contain the surface as well. If

the defining function has both positive and negative values in the vertices of a face, then the adjacent cell contains the surface as well. After the neighbours have been detected, the original cell is stored for further processing, while the newly detected cells are marked as active. Then the next active cell is selected and processed, until the process is finished. To avoid marking just processed cell as active again, each active cell keeps track of its 'parent' tetrahedron and checks if the new cell was not already marked active by another part of the propagation process.

Discrete nature of the cells is very useful when detecting intersections. The cell vertices can be assigned integer coordinates, and the cell itself can be uniquely distinguished by the average of its corner coordinates, e.g. the centre of the cell. This property allows us to store the cells in a very efficient way — a hash table based on integer coordinates. Hash table has constant access time for most operations, from inserting to searching and deletion.

Hash table of the lattice vertices helps to keep track of the points, so it is not needed to calculate function more than once for the same point. Since the computation of a function can be very expensive, this layout can provide up to four-fold speedup. Moreover, this structure ensures that a single vertex represents all the collocated vertices of the neighbouring tetrahedra. This condition is useful not only for the memory conservation, but it is also very important for the polygonisation process.

Local updates

When a change to the function-based model is introduced, one should be able to update the geometric model to show the changes to the user. To do this it at interactive rates, global re-polygonisation is to be avoided, as it will take too long to be considered anywhere close to interactivity. That is why one of the major points of the proposed visualisation method is the possibility of local updates. Below, we describe a local update method for meshes generated with the described algorithm. Same update method is used for updating both coarse and fine meshes.

For the local update to have advantage over complete repolygonisation, the following conditions should be met:

The modified area should be substantially smaller than the overall area of the shape. If the modification area is too large, it will not be possible to reduce the processing time in comparison to the global repolygonisation. Our methods were essentially designed for use in applications where the shape is changing gradually from one modification to another at a reasonable rate.

There should be sufficient information to determine which part of the shape was modified. In an interactive application this information can be derived from user actions.

The last condition is less natural than the other two, but it is not fatally restrictive, as it could seem at first glance. The modification area should be connected, for the update algorithms to be easily applied. In case if the area is not connected, two or more successive iterations can be used to deal with each disconnected part separately.

The update area is defined in the most flexible way – by a function. It is convenient to define an update area of a function with another function. The update area is defined as follows: at the points where it is greater than zero, the shape may have been altered; otherwise it is guaranteed to remain intact.

After the function confining the modification is given, the cells which are affected by modification might not intersect the new surface of the shape anymore. Hence, such cells need recalculating and are called outdated. One should detect and remove the outdated cells and replace them with newly propagated ones, as shown schematically in Figure 24.

The next step is to remove the outdated tetrahedra. This is done by propagating from the border cells inside the modification area marking the tetrahedra found as ‘dead’. This step does not involve any function sampling at all, all the propagation is performed based on the hash table. As we have reliably detected every border cell of the modification area, there is no danger that the ‘killing’ process could expand beyond it. (see Figure 24(b)). After the propagation is completed, the ‘dead’ cells are removed from the hash and memory, and the associated storage is released. This process is shown in the Figure 24(c).

Finally, the border cells are changed into the active ones, and the standard coarse and fine propagation is applied in a standard way described above. (Figure 24(d)).

In general the method requires $O(n)$ time where n is the number of tetrahedra generated by the update process, and the complexity does not depend on the total size of the model.

Figure 24. Local update procedure.

(a) Detecting the surface of modification area. (b) Detecting the 'border'. (c) Marking the cells from the border inside the area as outdated. (d) Reconstructing the cells over the changed region.

There are, however, some cases where this method fails to update the mesh correctly. If the tool causes the function to yield disjoint parts, for example chops off some part of it, the method may fail to detect that separate part. To workaround this problem the model should be connected.

5.1.5 Generation of polygons

After a set of tetrahedral cells containing the surface was obtained, it is necessary to derive the surface approximation from these cells. In the classic approach, this operation

is straightforward – the function is linearly approximated inside each cell and the surface is represented with one or two polygons for every tetrahedron. These polygons combine into a seamless approximation of the shape surface. There can be no inconsistencies or holes in the mesh – approximated function is always a manifold, though it can be different from the original one. For fine resolution, the difference is very small as the tetrahedra are small, and for the coarse polygonisation the speed and robustness are the most important factors as long as the general idea of the shape is conveyed.

Linear approximation of the surface inside a cell is convenient, but there are major deficiencies. First of all, the mesh quality is poor because the tetrahedron is intersected by the surface in an arbitrary way, and many thin and long triangles appear. These triangles are using the storage inefficiently, and cause artefacts in the smooth shading. Additional efforts should be made to make the co-located triangle points to be really the same points. This will save a lot of space, but requires additional computation. There are many algorithms that improve poor mesh, bring aspect ratios of the triangles close to unity, and even simplify it. Unfortunately, most of these algorithms will require breaking the connection between the cell and the corresponding tetrahedron, which is undesirable, as it will make repolygonisation of certain areas difficult. Besides, any decent mesh simplification algorithm will take additional time to complete, and this is also undesirable.

Therefore, one has to come up with a polygonisation method that will possess the following qualities:

Robustness and performance should remain the same as for the basic algorithm. It should take any cell configuration and produce reasonable approximation of the surface.

Consequently, the method should not require any additional function evaluations.

The generated triangles should be near-equilateral.

Each triangle should be attached to its respective cell, for easy handling. In particular, the algorithm should not produce seams during partial repolygonisation.

Every co-located triangle vertex should occupy the same memory, otherwise every vertex will be repeated at least four times, as every edge is shared by four tetrahedra.

The polygons are generated based on the *clustering principle*. This principle was recently used in different marching surface extraction schemes, but in the continuation methods it has never been applied, to our knowledge, especially with a repolygonisation option. Moreover, certain improvements were introduced to its performance.

Straightforward polygonisation by trilinear approximation inside each cell gives us all the properties except the triangle aspect ratio. It is easy to note that the surface of the linearly approximated function is always a plane; otherwise, for some line segment for the function would be zero at its ends, and non-zero at some point in-between, which is impossible for linear approximation. A plane gives only two cases of polygonisation – one triangle or one quadrilateral, which in turn can be represented as two triangles. The mesh contains many poorly shaped triangles, but still it is consistent. Every corner of a triangle belongs to a cell's edge. For each vertex there are three to five other vertices located at exactly the same point.

To improve the shapes of the polygons, i.e. avoid long and thin polygons, for every cell, the appropriate vertices on its edges are relocated to the nearest of the two of the respective edge's ends. These points, the vertices of the tetrahedra, are called clusters, as several points finally join there. Small triangles are reduced to lines, and any triangle that is not reduced becomes equal to the cell's face. The surface remains closed, as the operation is essentially just movement of the vertices. While there are no holes in the surface, and it consists only of the ideal triangles, as displayed in Figure 25(a), it looks somewhat rugged, because the triangle's vertices are significantly displaced off the surface. To overcome this deficiency, the cluster coordinates are split into two groups: – the first coordinates are integer and serve the logic purpose to denote the cluster, and the other ones are real, and computed as an average of all the points included. The real coordinates are close to the surface if the surface is smooth and linear approximation works well, but very sharp corners will fade as a result of the operation. This effect will be of the scale of a single cell, which is negligible in our case.

As it can be seen in Figure 25(b), the result is pleasing and suits us well, as it fulfils the initial requirements: the mesh quality is good enough, every cell knows its part of the surface, the mesh is consistent, and the vertices are joined together. These properties can

be achieved without several complex stages but in a single run during propagation process.

Figure 25. Clustering polygonisation result.

(a) Mesh vertices clustered in the corners of the lattice. (b) Clusters brought to the surface of the shape by averaging original vertex positions.

5.1.6 Re-meshing and fine polygonisation handling

Let us consider how local updates can be performed with clustering polygonisation. Each cluster keeps track both of cells referring to it and polygon vertices clustered. If the counters are decreased to zero, for example when outdated mesh is being removed, the cluster is removed from memory and hash tables. The only question is how to treat the border cells during the update process. To achieve the optimal merge, the cells themselves are not removed, but the triangles are eliminated. Next, the triangle reference counters of the corresponding clusters are also decreased. After these preparations have been completed, the normal polygonisation process begins with the border cells. Although triangles at the border can slightly change their form, they remain well formed and consistent, as the propagation process is the same as used for initial propagation.

A different approach is needed for the fine cell polygonisation. The problem is that their clusters are not persistent, and they are removed after processing. Without special handling, this will cause seams during updates, because the clusters located at the edges will be drawn inwards during averaging as there is no compensation from outside. To resolve the situation, the border clusters are handled in a special way – they do not accept contributions from the edges that are not located on the border. The fine clusters that

appear to be located on the edges accept only contributions from this edge. As a result of the restrictions, the patch aligns with existing faces nicely and joins seamlessly with neighbouring ones.

The additional measures for the fine polygonisation have negligible impact on the polygonisation speed, but if the cells are fine enough, the hassle can be altogether avoided by using point representation. Each of the fine tetrahedra can be assigned a point and a calculated normal. Each representation has its own pros and cons: the polygonal one is more difficult to obtain but it provides better quality. On the other hand, points are more efficient for the computers without the graphics support in hardware. The final decision is left to the user.

5.1.7 Optimal mesh data layout

Effective rendering is the vital part of the overall performance. The coarse mesh should be redrawn at least four times per second during interactive manipulations, so the internal layout of the mesh storage has to be heavily optimised for rendering performance. This is especially true for coarse mesh, which is responsible for the interactive behaviour.

The extensive benchmarking of OpenGL revealed that in most cases the preferred way of storing data is indexed array. It can be readily rendered with a single *DrawIndexedArray()* call. The clusters store indices of the respective Vector and Normal entries, while the cell itself, if it had triangles, stores the pointer to the array of indices. A custom-made allocation procedure has been created to allocate and de-allocate storage in these arrays in the fastest possible way. We are not going to describe further details, as memory allocation problems fall beyond the scope of this thesis.

The fine resolution rendering is performed in the background, and the requirements are less restrictive here. Every fine patch consists of a list of coordinates and normals, and for every patch a separate OpenGL command is issued. Points are drawn in the same way, and require three times less storage.

5.1.8 Performance measurement

We have checked the performance of the polygonisation algorithm by running a set of tests. Each test consisted of the complete visualisation of the shapes in Figure 26. The

shapes differ both in visual and computational complexity, which was proportional to the number of functions the shape contained. The function defining model in Figure 26(a) is made with a single operator, Figure 26(b) contains 187 operators, and the most complex model in our test Figure 26(c) contains 7545 operators. The number of operators gives only approximate idea of the function complexity, as different operators take different time to evaluate, and their areas are also different.

The polygonisation process was run in two modes for each model. In the initial run, only the coarse processing was performed, and the coarse mesh was refined in the second set of tests. For every model, the data consists of two parts: the coarse resolution data includes timings and information about the coarse polygon generation, while the fine resolution describes the time and tetrahedra counts necessary to perform the complete refinement, not including the initial coarse polygonisation.

Figure 26. The test shapes.

(a) Sphere : 1 operator (b) Vase: 187 operators. (c) Dragon: 7547 operators.

The following information was considered vital for the performance evaluation: the number of cells the algorithm looked through to detect the surface, the number of triangles resulting from the processing of these cells, and the amount of function evaluations required for each particular test.

The processing time was measured for the two different hardware configurations L and H: L is a notebook (PIII 800MHz/256 Mb/Win2000), and H is the INTERGRAPH XE2 graphics workstation (2xPIII 800 MHz/1024 Mb/Win NT).

The data from the tests is presented in the table below:

	Sphere, 1 operator		Vase, 187 operators		Dragon, 7545 operators	
	Coarse resolution	Fine resolution	Coarse resolution	Fine resolution	Coarse resolution	Fine resolution
Tetrahedra processed	175,000	2,807,000	117,000	1,918,000	149,000	2,405,000
Triangles generated	61,000	979,000	43,000	701,000	60,000	960,000
Evaluations needed	60,000	978,000	41,000	697,000	57,000	951,000
Time for L	0.650 s	25.237 s	3.044 s	99.543 s	113.193 s	2400.786 s
Time for H	0.610 s	18.597 s	2.734 s	49.578 s	60.125 s	1190.786 s

The data reveals several important qualities of the algorithm. First, note the efficient use of the information derived from the function evaluations. On average, a triangle is constructed using a single function computation, and the triangle shape is not degenerate, as the clustering algorithm was employed. This is clearly seen from the 3-fold difference between the number of the cells examined, and the number of triangles generated. Without clustering, each cell would generate a triangle or two without any gain in quality. The speed of the surface extraction is rather high. For a simple function it gives up to 100000 well-shaped triangles per second even with the lower hardware configuration.

Note that this data is given for the coarse and fine resolution rendering of *the whole shape* after it had been interactively created. For each individual interactive operation the system responds in *real time*, i.e. fast enough to be used in an interactive application (less than 1-2 seconds). Note also that only about one function evaluation was required to perform per each generated triangle including the normal calculations.

5.1.9 Image-based scene update

At the end of the previous subsection rendering-related features were introduced. There the focus was mainly on the optimal data organisation. In this section it will be shown

how two different resolutions can be combined, and how the image can be locally changed without complete redrawing.

When the coarse mesh is being redrawn four times a second, the rendering is performed simply by modifying the observer position and/or lighting condition, clearing buffers and calling the appropriate function.

As for the fine mesh, the situation is different. Redrawing the scene can take several seconds, so it is desirable to avoid complete redrawing after every modification of the shape. Instead, the picture with its already drawn parts should be considered as an important resource, which should be carefully updated instead of discarded. To achieve this goal, we used an OpenGL capability to save screen buffers with depth information. To perform an update, the saved buffers are to be restored, then the update will take place, and after that the buffers will be saved back again. Any other objects in the scene, such as visual tool representations, 3-D controls and others were drawn afterwards to keep the original picture containing the shape clean. Basically, the 3-D tool representations can interact with the shape correctly, even without any mesh actually being in the memory. The fact that only a part of the image is updated creates an illusion of interactive work with the shape rendered entirely in fine resolution.

The updates include two basic types. First is the simplest. It is drawing the fine patches after the corresponding coarse shape has already been shown. It can be performed by drawing the patch at some distance of the coarse mesh in order to overwrite it and avoid visibility problems at the same time. The second type of update is more complicated. It should be performed when the new coarse mesh has been obtained and the old image is to be updated to reflect it without complete refresh. While the problem can appear simple, there is no general solution. If the new patch is simply drawn above the old mesh taking depth information into consideration, everything will look fine until a 'cavity' is introduced to the model. The existing depth information from previous polygons will prevent the new polygons from being displayed, as they are located further from the observer than outdated ones. On the other case, the depth information cannot be ignored either, as it will make invisible parts of the new patch to appear on the screen. To overcome the problem, we developed a three-stage update procedure that works fine in

most situations. The method cannot work for all situations, but this is not a serious problem as in (rare) case of incorrect update, a 2-second complete redraw can be used.

The main idea of the method is to remove the depth information from the image where it is necessary, and leave it in place where the image should not be changed. To accomplish this, the outdated mesh of the updated area is required. It can be easily obtained as a by-product of the model update procedure described earlier. After the new and the old versions of the same area are available, the update starts.

First step. Figure 27(a). The old patch is drawn with a slight offset towards the user. Only a stencil buffer is affected at this step, marking the area where the old mesh is visible. The black dashed line shows the patch to be replaced, and the red line outlines its visible part.

Second step. Figure 27(b). The old patch is drawn once again using the background colour. It destroys depth information masked by the stencil buffer from the previous step. This step effectively resets the z-buffer to the infinity in the places where the old patch is visible to the user. It will be replaced by the new patch there in any case.

Third step. Figure 27(c). The new coarse mesh is drawn. The fine mesh can follow in the usual way after it is ready.

Figure 27. Image update stages.

(a) Detecting where to reset z-values in the image. (b) Drawing old mesh to clean the z-value. (c) New mesh appears only where z-values was cleared.

There are some configurations that cannot be theoretically handled by the described image updating methods. For example, if the old mesh has overwritten some details, and they should have become visible after the update, they will not. Another flaw of the

method is in possible disregarding of the small details on the edge of area, as the edges can be slightly different for old and updated mesh because of clustering. Nevertheless, if the user is not satisfied with the performance of the method in a certain operation, he can request the scene to be redrawn from the scratch. The redrawing delay will not affect general interactive experience as such problematic cases are not common.

5.2 Extension for parallel rendering

Even with efficient polygonisation algorithms described in the previous chapters, one computer might not be enough.

In the following we are going to overview the proposed algorithms for distributed shape rendering.

For every new cell we have to check if it has already been processed. This requirement is difficult to follow when there can be several processors working on the same shape, as one would have to perform at least one remote call for every cell, to make sure no other node has extracted it already. To solve this problem we group tiny cells into cubical *cell blocks*. Each block can be processed independently, using only 2 calls (one for retrieving the task, and another one for returning the extracted surface and border cells). The structure is shown in Figure 28.

Figure 28. Cell blocks.

Cells inside each block can be processed independently. Border cells are sufficient to make sure that there are no cells processed twice. The blocks overlap so that border can be used as seeds for neighbouring cell blocks.

Figure 29 shows how the distributed surface-extracting process works:

- 1) The initial cell block involved in polygonisation is processed using a computing node. It produces a surface patch and a border.
- 2) The border is used to determine which neighbouring cell blocks are to be processed. The cell blocks are distributed among the nodes for computation.
- 3) If no new cell blocks can be found, the process is considered to be completed pending modification requests. Otherwise we repeat from step 1.

Figure 29. Parallel surface extraction.

Each processing node is assigned a cell block. When the block is processed, the next block is taken from the queue for processing

A coordination node receives all the processed patches from the computing nodes and supplies them with the new cell blocks to process.

Cell blocks distribution is coordinated by a scheduler – it is a simple queue where nodes can take blocks to process and return surface patches and border cells. As a single block can contain disjoint parts of the surface, it can appear in the queue several times. To avoid cycles, adjacent cell blocks are not processed at the same time.

If the shape was modified by the user, the scheduler detects which cell blocks are affected and therefore have to be recalculated. To perform consistent modifications, each cell block is given *spatial ID* and a *timestamp*. Spatial ID determines where the block is located, so that when another patch comes in with later timestamp, it could be easily found and replaced.

Speedup of the parallel propagation method is linear on the number of processors used, as there is no interprocessor communication is needed during the extraction.

5.3 Haptic rendering

The polygonisation methods described above provide interactive visual feedback. Now, we are going to describe methods that are needed to perform haptic rendering, providing the illusion of touch.

To do so, we need a method to calculate a force acting on the probe about 1000 times per second. Let us discuss the way of calculating the feedback forces first, and then propose the methods to make these calculations as fast as possible.

5.3.1 Forces for function-based haptic feedback

Forces acting on the user are composed of two components. The first one, denoted N in Figure 30, conveys the shape of the object, by preventing the probe to go inside the shape, and therefore allowing the user to feel it. Component T represents friction and the surface texture of the material, therefore it is controlled independently and can depend on values of the function, allowing the user not only to change the geometry of the object, but to interactively modify surface properties. This can be used for operations like polishing, brushing, etc.

Figure 30. Normal and tangent components.

The normal is directed along the gradient, and the force is expressed as follows:

$$F_N = \frac{\nabla f}{|\nabla f|^2} k_{profile}(f)$$

Where f is the function and $k_{profile}$ determines the force of response for each point.

To avoid high-frequency noise in the value of force feedback, we average it exponentially:

$$F_N=kF_{Ni}+(1-k)F_{N-I}$$

where $k < 1$.

To prevent force overload, and also to allow the probe to stay inside the shape in certain special cases, for example after a modification has added material to the shape, we limit the normal force using the profile shown in Figure 31.

As with friction, T is directed against the probe velocity and $|T| = f_c$, that is T is governed by a function value that can be changed interactively.

Figure 31. Force magnitude profile.

5.3.2 Acceleration for haptic rendering

Haptic rendering is usually performed when we already have the coarse mesh available, so we are going to take advantage of it. Moreover, we have developed function clusters, which will be helpful when the coarse mesh is not available for some reason.

Figure 32. Using tetrahedra to accelerate function evaluation for haptic rendering.

Coarse tetrahedra are located near the surface, exactly where the fastest refresh rate is required. The tetrahedral structure can be detected by querying a hash table in constant

time. Then, each tetrahedron has acceleration cluster associated with it, which can be used right away, without calculating the whole function, as shown in Figure 32.

Figure 33. Sculpting session using 6 DOF force feedback device.

When the help of the surface tetrahedra is not available, we employ the coherence property of the sampling points. When the first point is sampled, we create an operator cluster, so that subsequent evaluations can use it for acceleration. When the sampling is close to the border of the cluster, a new one is created with the centre at the new point. As only a fraction of the whole function is calculated, we get substantial acceleration with smooth performance even for complex objects like those shown in Figure 33.

5.4 Summary

Function-based shapes are very attractive for use in the interactive design applications. They offer a unique set of operations, they are easy to handle, very precise and concise. Unfortunately, functions are rarely used in modelling applications because they are very difficult to render in real time, especially with fine details. We have proposed a visualisation method intended for the function based shapes and interactive operations on them. Two-level resolution allows the method to be used both as a high quality renderer and as an interactive visualisation tool. The image based algorithms allow us to perform interactive rendering of modifications for a very high-polygon models.

The method can handle very complex functions due to its effective sampling algorithm. Every bit of the information obtained from the sampling is utilised for creation of high quality polygonal mesh with near-one aspect ratio. Two-level resolution gradually

reveals every fine detail of the shape up to practically arbitrary level of precision. The resolution is not limited by the available memory, as the fine mesh can be located in the secondary storage. At the same time, lower resolution approximation is available for interactive manipulations and alterations. Local modifications are handled in a very effective way allowing the user to modify the meshes consisting of hundreds of thousands of triangles, and providing a good interactive experience. A proper rendering and image-based algorithms allow us to combine fine details with interactive response, extending the possible applications of the algorithm even further.

The proposed visualisation method is based on regular tetrahedral mesh. We are using it to augment the rendering process with haptic feedback. When a tetrahedral mesh is not available, separate operator clusters are used to provide sufficient refresh rate for the haptic device.

We have developed a method to distribute the propagation algorithm between several processors, in case the model is too computationally expensive. This method successfully deals with interactive updates and suitable for real-time visualization.

Finally, a haptic rendering method was proposed to use with the model. The method makes use of structures created during propagation.

Chapter 6. INTERACTIVE MODELING ARCHITECTURE

In the previous chapters, we have developed a largely independent set of algorithms that would be useful for function-based modelling. To use them, we need to develop an application that will naturally accommodate all the proposed improvements. Below we will introduce the proposed design of such an application.

We have implemented a pilot version of the software, and it has been used to demonstrate the advantages of the methods, as well for creating several really nice pieces of art.

We describe what modules constitute the system, how they are interconnected, and what information flow occurs. Then, each individual module operation is described in greater details. Special attention is paid to how the algorithms interoperate.

6.1.1 Modelling framework

When building a function-based interactive application one has to distinguish two modules accommodating two areas of function-based design – function module and surface extraction one. In our application, we decided to introduce a dedicated component for geometric model handling, so called geometric module. The decision was made because the multi-level mesh and visualisation should be handled independently of other processes. Finally, a user interface module has to be designed for an interactive application.

Let us sum up the module functions:

The *Function* module is responsible for accommodation of a function model, its modification, and proper use of evaluation acceleration techniques.

The *Polygoniser* module is used for effective two-level resolution shape surface extraction for further handling in a geometry module. It encapsulates the propagation and refinement algorithms.

Mesh handling module has no direct relation to the previously described improvements, although it is essential for storage of the polygonal mesh of the surface. It is also responsible for off-core mesh storage.

User Interface (UI) module is intrinsic part of any interactive applications. It handles both input and output devices, and incorporates complicated algorithms, for example those for haptic device handling. Besides, in our case, it takes part in governing interactions between other modules.

Figure 34 shows basic interactions between the modules. In general, the system contains two major parts: a UI and a modelling system. The UI converts user's input into the commands for function modelling, therefore causing the function-based representation of the shape to change accordingly. Then, the interface module generates a request for the update of the geometric representation in the modified area. After the repolygonisation succeeds, the new mesh is acquired from the geometric model to be displayed on the output device.

Figure 34

Figure 34

Figure 34. Application module interaction.

6.1.2 Function module

The function module is a vital part of the system, its behaviour defines modelling quality, and its evaluation speed has determinative effect on the overall performance. In general, function module should contain means to change function, as well as to accelerate its evaluation. Various pre-known conditions and simplifications should be taken into account to speed up the function evaluation even further. In general, this could be accomplished in the way shown in the Figure 35

The central part of the module is *function compiler*. The compiler assembles the final function from the interactive operations provided by the user. Several versions of the function are available at the same time. A direct evaluation component stores the function that does not incorporate any acceleration techniques. It is used for queries that do not require immediate response, as well as for verification purposes.

Accelerated versions of the function include various evaluation acceleration techniques. In general, these techniques may require additional information, such as the area of application, which is provided by the users of the module. The approximation

acceleration component of the function trades precision for speed, it is useful for time-critical operations, for example, for a haptic device interface.

Figure 35. Function construction and evaluation module.

6.1.3 Geometry extraction module

A function-based shape model cannot be visualised directly, so a geometry extraction module is introduced to convert function-based shape to a geometric representation. It incorporates all the improvements to the shape extraction algorithms that were developed earlier.

The polygoniser can receive update requests, and output coarse and fine resolution meshes. These two resolutions serve different purposes: coarse resolution is designed to provide interactivity and to give a general idea of the shape, while the fine provides a complete and highly detailed image.

These two algorithms are not independent. The coarse polygonisation is primary, and has a higher priority as it reflects the changes in the function model interactively. On the other hand, the fine polygonisation has more relaxed timing requirements, and can be performed in the background. In our case a separate low-priority process is devoted to it.

Figure 36. Two-resolution polygoniser.

When the coarse polygonisation process generates a cell, it puts the cell in the queue for fine polygonisation in no particular order. From time to time it can be required to refine a certain area before others. This can be accomplished easily with this design. To have certain cells polygonised earlier, we simply put them in front of the others in the queue. Nevertheless, care should be taken not to put the same cell in the queue for the second time.

After processing, the resulting mesh, both tetrahedral and triangular, is conveyed to the next stage: the mesh handling module for storage.

The overall structure of the polygoniser is clarified with the diagram in Figure 36.

6.1.4 Geometric model

As mentioned, we use two interconnected geometric models – one to represent coarse approximation of the shape being modelled, and the other one to display fine details. Both of these meshes are triangular ones, although if hardware and resolution allows, the fine mesh can be directly replaced by a point representation, although this will not lead to any substantial improvement.

Let us discuss how the models are stored in the geometric module. The storage is based on the coarse mesh, which is always maintained in a complete and up-to-date state by the polygoniser. An additional integer-based dimensional search structure associated with the mesh allows for locating triangles in any specified area, either for replacement or

for visualisation. The search structure is actually derived from the polygonisation process, so it does not require any additional processing time to be created. Every triangle of the coarse mesh is associated with several triangles of the fine one. Therefore, after locating the normal triangles in a required area, the fine representation of that area is automatically located by the geometric module.

Figure 37. Storage of geometric model with fast access.

There are three reasons for such double-layer architecture. First, we do not have to maintain a separate integer-based search structure for the fine mesh. This saves a lot of memory and enables us to create larger meshes. Second, with such a design we can save rarely accessed fine mesh to disk, reducing memory limitations. Finally, saved mesh can be reused in the next modelling sessions. The layout is schematically shown in Figure 37.

6.1.5 User interface module

The interface should enable the user to carry out the modelling process with the available input and output devices in the most effective way. We adopt the sculpting metaphor for the modelling process. The desired shape is created by successive application of the appropriate tools. With the proper tools, the sculpting approach does not limit us to the sculpting itself, but also enables creating of a wide variety of models through the mathematical description of real-world operations.

Overall interaction is generally split up into two functions – *input* and *feedback*.

Input is carried out through either 2-D input device, like the mouse or a Phantom 6DOF force feedback manipulator. The force feedback manipulator provides 3 spatial

coordinates and roll/pitch/yaw. The 2-D and 3-D modes of input are very different: while the 3-D mode is fairly intuitive, the two-dimensional devices act in a straightforward manner only when flat surfaces are concerned. The two coordinates of a 2D input device, for example a mouse or a tablet, can be mapped to three dimensions with the guide of the shape itself by projecting the tool application point to its surface. With such an approach, the spatial modelling is less straightforward, as there is no direct way to define force applied to the tool or the orientation of the tool. Therefore, two-dimensional controls are only appropriate when the details to be created are largely located on a plane, like, for example, in wood plate cutting. For essentially spatial operations, three-dimensional control is desired. After all the processing, the input from the user is directly translated into the requests for functional model alteration.

The most important task in a *feedback* generation for the interactive systems is to provide the timely and sufficient information for the user to evaluate his operations. In our system the feedback is organised into two layers. The first layer is intended to give the user an idea on where his tool is located and how it is oriented. The three-dimensional model representing the tool is generated by the UI module itself (Figure 38), and it is oriented in accordance with the manipulator position to provide the control over the operation. While the user chooses the right position for the tool, the response is instant, because the picture is updated without redrawing the model, just using saved depth information from the most recent rendering. After the tool has touched the shape and a modification has begun, the second feedback layer comes into action. Changes that have been made are to be visualised in real time. The request to polygonise the appropriate area is issued, and the resulting mesh is used to update the image. As for every single action on the shape, the polygonisation is usually confined to a limited area. The polygoniser will promptly extract the updated geometry of the coarse resolution. Small details will be added later, with the fine polygonisation process. During the operations the image is updated only locally, rather than redrawn from scratch, which adds much to the interactivity, especially on the low-end graphics cards.

Apart from the visual feedback, Phantom 6DOF device allows us to use force feedback, so that the user can actually 'touch' the function-defined model and control the

modifications with the natural ‘pressure’ against the shape surface. The control with pressure is also employed when a pressure-sensitive tablet is used. In other cases, the pressure amount for each operation is discretely selected through a keyboard.

Figure 38. User interface layout.

6.1.6 Summary

We have described the basic components of the modelling system. All of the methods developed earlier found a proper use in their respective areas. The methods for the surface extraction were used both in the polygoniser module and in the geometry storage module. Function construction and acceleration techniques found their application in the function module.

A separate user interface module was developed to carry out the communications between the user and the internal modelling core of the system.

The structure of our modelling application shows how the various methods and algorithms devised for working with function based shapes can be brought together to obtain an interactive modelling system. This is by no means the only way, but the one that was adopted by us and that resulted in a decent application.

6.2 Experimental results and discussion

An application of the proposed algorithms to an interactive shape modelling application demonstrated that they can be used in practice to create quite complex models. What is more important, the models were created using techniques common in the real world, for example carving and sculpting, but with complete control possible only in virtual environment, like the ability to cancel an action or modify the result of some past editing.

Efficient visualisation algorithms allowed the modelling system to render raytracing quality image directly during the modelling process. You can see an example of the image and interface in Figure 39, which illustrates the process of interactive embossing and painting.

The modelling system defines the following starting shapes:

Basic primitives like sphere, cube, etc.

Explicitly given formula for function representation

Previously created shape

The shape shown in Figure 41 was created starting from the initial shape defined analytically (Figure 40). First, the relief portions were interactively modelled, and then colours were interactively applied. Similarly, the shape displayed in Figure 42 was carved out of a disk primitive.

Figure 39

Figure 40

Figure 39. Model with colour and program interface.

$$\begin{aligned} & (((1-x^4-(1.8z)^4-1.8y^2)&(0.4-z)&(z+0.4)) \\ & |((0.15^2-(x+0.4)^2-z^2)&(1.1-y)) \\ & |((0.3^2-(x+0.4)^2-z^2)&(y-1.1)&(1.25-y)) \\ & |((0.15^2-(x-0.4)^2-z^2)&(0.95-y)) \\ & |((0.3^2-(x-0.4)^2-z^2)&(y-0.95)&(1.1-y)))&y \end{aligned}$$

Figure 40. Initial shape and its defining formula.

Figure 41. Candlestick model with fine details and colour.

While the representation was designed specifically for enabling real-life free-form modelling with computer, it also has a potential in different areas such as collaborative shape modification due to small model size, and even terrain modification, because of scale independence. However additional work is required for such applications.

Figure 42. “Mouth of Truth” model sculptured out of a disk.

6.3 Summary of the application design

We combined the algorithms for the function-based model visualisation and construction to obtain some practical results from the proposed methods. Two major areas were

considered – the interactive application as a whole, and the construction of the functional model and operations with desired properties.

Despite its advantages, function representation is rarely used in modelling systems. A different layout is needed to deal with the processing pipeline of a function-based application. We have proposed a modular design for such an application, which allows for interactive manipulation, as well as for interactive visualisation of the shape being modelled. Some of these modules can even be readily used in different function based applications.

Acceleration algorithm proposed in Chapter 3 allowed to achieve an order of magnitude speed-up. For example, for dragon model, out of 7 thousands overall operations, each point is affected by only 10 modifications (on average). Taking into account lookup overhead, we have obtained around 10 times faster function evaluation.

The application framework addresses a number of issues, such as gradual refinement, effective storage retrieval of the model mesh, and integration of different acceleration algorithms. Updates of static images of the model are also handled, to improve interactivity.

In the area of model construction, two major problems were dealt with. First, it was necessary to find optimal acceleration structures for different operation types. An approximate classification of the types was performed, and some general speed-up techniques were proposed. Further, the proposed techniques contributed to the construction of an operation set emulating real life tools for embossing and carving.

The practical outcome of this work was a viable modelling application, which enabled a real-world artist to use his virtual instruments on a computer model in a way he usually does it in his work. Several pieces of art were created with the program.

Chapter 7. FUNCTION-BASED WEB VISUALISATION

Among all geometric representations, function-based ones possess the unique property – simplicity of the interface. All the complexity is hidden behind the simple mapping from $\mathbb{R}^3 \rightarrow \mathbb{R}^1$. The property provides us with the possibility to combine all kinds of visualisation methods. An example of the flexibility is presented in Chapter Chapter 5 when the same function was used first for surface extraction, and then for force feedback.

By taking advantage of the function's compactness a method for sharing the function-based objects created with the modelling system and visualising them over the network can be provided. The goal is to facilitate distribution and visualisation of the function-defined models from the place they originate to the users. Most of the end-users of the shapes don't need to modify them. Instead, they need to obtain just the model itself with as little hassle and software to install as possible.

7.1 Remote shape viewing

In [Liu Q., Sourin A., 2004] an extension framework for the VRML ([VRMLweb], [FVRMLweb]) is presented. By adding procedures for visualising and evaluating our function-based models (shown in Figure 43), one can take advantage of the function representation and deliver only the formula instead of thousands of polygons, which can take considerable bandwidth even in an optimized form [Chen, B. Y. and Nishita, T. 2002]. Then, using the described visualisation process, the shape can be visualised on the remote machine, using standard browser with the plugin. (Figure 44).

Figure 43. Web-based remote visualisation process.

Internal binary representation of the shape is processed with a Base64 encoder, and included into the a Virtual Reality Modelling Language (VRML) file for viewing over the Internet.

Figure 44. Shape transferred over the web.

At this point, remote modifications are not considered, so the 2-resolution web visualisation can be implemented using JavaScript. In this case, coarse modification is needed to get a preliminary view of the downloaded model, while the fine view is displayed whenever the controls are idle and the user wants to get a clearer view. Therefore, coarse/fine-resolution approach taken in Chapter 4 is implemented on the client side with the VRML language facilities. This technique allows combining more complex function-based objects in one scene without additional optimisations.

7.2 Function-based terrain

Function-based models do not have any fixed intrinsic resolution. One and the same model can have details on several orders of magnitude different scale. This multi-scale property can be used to represent editable terrains.

Function-based models can be used in interactive terrain visualisation as follows: Plugin described in the section 7.1 allows to stop polygonisation at certain distance from the starting point. For terrain, one does not have to perform fine surface extraction, so the result shown in Figure 46 can be achieved nearly instantly. If the user's viewpoint is elevated, he might see uneven edge where the polygonisation of the current patch has stopped. To hide this artefact, we used the Fog feature, which introduces a gradually increasing fog in the scene, obscuring objects further than certain distance.

Figure 45. Repolygonisation for terrain generation.

As soon as the user approaches his horizon, i.e. travels from the point O to the point O', a new polygonisation process is started, to obtain new surface to walk/fly on. The uniformity of the plugin has considerable advantages. One can use arbitrary scale in the same VRML scene, and explore it from different points of view.

Figure 46. Generated terrain, with and without fog.

(a) A patch of terrain surface is generated from the function in real time. Once the user approaches the edge the generated patch, a new patch centered on the user is produced. **(b)** To improve illusion of walking over continuous surface the edges of the patch can be hidden with fog. Otherwise, the user can see switching of the patches.

7.3 Summary

Compactness and universality of the function-based representation allowed us to devise a method to incorporate the models created with the modelling system from Chapter 6 into VRML environments. This has provided efficient means to use interactively created models both locally and over the network.

Capabilities of the VRML language and scale-independence of the function representation allowed us to use it for representing editable landscapes without transferring huge amounts of data.

Chapter 8. DISTRIBUTED RAYTRACING

Polygonisation-based rendering can yield high-quality interactive visualisations. However, after the interactive work is complete, it is desirable to perform photorealistic rendering of the model.

Among non-interactive visualisation methods, raytracing is one of the most precise physics-based engines, capable of representing reflection, refraction and different materials faithfully. Universality of the function-based approach allowed us to incorporate the calculating module into a POV-Ray raytracer ([*POVray*]). While quite slow by itself, raytracing can be easily sped-up by distributing the work among several computers.

In this section the guidelines for adapting the function-based model for distributed raytracing will be provided.

8.1 Raytracing

The raytracing principle is quite straightforward – a ray is shot from the viewer's eye through each pixel of the image. For each ray it is required to find an intersection with the shape. It does not need any explicit surface representation, as the information about the ray intersection can be efficiently derived for each separate ray.

Such architecture can be efficiently represented with plugins. There is no need to incorporate some complex structures into an existing raytracer. Instead, just a way to transfer queries to the defining function and back should be provided by implementing a plugin (Figure 47).

The universality of the function-based model definition allowed for reusing of the same plugin that was used for the web browser in the previous chapter, by incorporating it to the POVray raytracer.

Figure 47. Ray tracing architecture.

8.2 Operator clusters for ray tracing.

The described plug-in should be accompanied by an acceleration structure. The polygonisation algorithm from Chapter 5 could use the propagation process as a reference for creating clusters. In case of ray tracing, however, the plugin has no way of knowing where it is going to be evaluated.

Figure 48. Dynamic sparse grid.

In practice, however, some level of coherence does exist in the ray tracing process. To utilise this coherence without significant memory overhead, an expiring hash-based grid is used. Each existing cell of the grid contains an operator cluster, which is calculated during the first hit into a given cell. There is a trade-off between the grid precision and the amount of memory it takes to store the grid. As the same place can be queried several times, the usual sparse grid is augmented with a First-In-Last-Out queue. Each time a cell gets hit, it is moved to the head of the queue. After some threshold is exceeded, the list is purged of unused cells, which are marked for reuse. Schematically the optimisation technique is illustrated in Figure 48.

8.3 Parallel ray tracing

Even with the optimised function evaluation, rendering of a complex model can take up to 10 hours and even more. Therefore, it is desirable to be able to spread the task among available machines.

This can be accomplished through the GRID architecture (see, for example, [Foster. I. 2002] for a survey) in the following manner (Figure 49):

Figure 49. GRID distributed rendering.

Several nodes are allocated to perform raytracing. The allocated nodes connect to the interactive modelling system to receive the model representation and rendering requests. The interactive modeller produces a compact representation of the model to be visualised and distributes it between the computing nodes. Each node communicates its part of the picture back to the user, and receives the list of new area.

As the coordination task is performed by the interactive modelling system, it can be controlled by the user, to achieve interactive updates to the rendered image.

8.4 Results

By applying the methods described above, photo-realistic versions of the pictures were obtained. Transparency and refraction effects (Figure 50 Figure 51(b)) add important part to perceiving the results of a modelling session.

Both interactively coloured and procedural textured objects can be created. (Figure 50(a) and (b)).

In our experiment we have used 20 machines.

Rendering time for Figure 50 (a) using a single machine is around 45 minutes. When all the 20 machines were used, the rendering time was reduced to 5 minutes. Around 30 seconds were spent in start-up of the nodes and 2 minutes for communication overhead.

The communication overhead is substantial, given that raytracing is an easily parallelizable task. The communication layer was implemented in a scriptable language, which provided flexibility at the cost of performance. We are planning to improve on it by employing more advanced communication technologies.

(a)

(b)

Figure 50. Ray-traced examples.

(a) Transparency and colouring. (b) Several function-based models in a scene.

Figure 51. Vase of ruby and face of marble.

Figure 52 demonstrates complex interactively sculptured function-defined object.

Figure 52. High-quality ray traced image.

Chapter 9. CONCLUSION

In this project, a framework for interactive function-based shape modelling was proposed, where operations on shapes are represented with mathematical operators. This approach offers a flexible way for representing a function-based model during the course of interactive modifications.

The function-based approach places no restrictions on properties, which are to be modelled, therefore it is not limited to geometric representations only, but can also represent colour, opacity and other physical properties.

To support the approach and use its unique capabilities several algorithms and techniques were developed including acceleration techniques, visualisation methods, VRML and raytracing extensions.

9.1 Function-based modelling techniques

Guidelines for implementing interactive operations were proposed, along with several important example operations.

The methods for efficient construction of the operations for the function-based models were devised. The proposed algorithms aim at two different but related goals: first, one has to faithfully represent shape deformations, taking into account properties of material, type of the tool and similar parameters. The variety of operations implemented demonstrates the ease of defining different deformations representing all kinds of shape modifications. Then, the way to ensure compact support of the operators representing the modifications was proposed.

The goal of the developed modelling system was to place little restrictions on the interactive operations. A localisation property was proposed for each operation, and the dependencies were explicitly specified using cluster-based approach.

The cluster-based acceleration method utilises the localisation property and performs efficient filtering for efficient function evaluation.

Guidelines for constructing efficient 3D modelling operators were used to create interactive shape modelling tools, which can be used for modelling different 3D crafts. The developed tools were applied for modelling embossing, sculpting, carving, and 3D painting.

9.2 Visualisation algorithm

A robust visualisation method intended for function based shapes and interactive operations on them was proposed. A two-level resolution allows the method to be used both as a high quality renderer and as an interactive visualisation tool. The image based algorithms makes it possible to combine the properties to achieve interactive rendering of the photographic quality image.

Due to the choice of the propagation unit, which provides 50% better sampling efficiency compared to traditional cubic cell (Section 5.1.3), the visualisation method can handle very complex functions. Every bit of the information obtained from the sampling is utilised for creation of high quality polygonal mesh with the near-one aspect ratio of the triangles. The two-level resolution gradually reveals every finest detail of the shape up to a virtually arbitrary level of precision.

Lower resolution approximation is used for interactive manipulations and alterations while local modifications are handled in a very effective way allowing the user to modify the meshes consisting of hundreds of thousands of triangles, and providing a good interactive experience. Proper rendering and image-based algorithms allow for combining fine details with the interactive response, extending the possible applications of the algorithm even further.

The proposed visualisation method is based on a regular tetrahedral mesh, so it is possible to use the mesh to augment the rendering process with haptic feedback. When the tetrahedra are not available, separate operator clusters are used to provide a sufficient refresh rate for the haptic device.

9.3 Using functions for remote and distributed visualisation

Compactness and universality of the function-based representation allowed for devising a method incorporating the interactively created function-based models into VRML environments. This has provided efficient means to use the models both locally and over the network.

Capabilities of the VRML language and scale-independence of the function representation allowed for using the same interactively modelled functions for representing editable landscapes without transferring large amount of data.

The flexibility of the function-based approach allowed to extend the VRML plugin to distributed raytracing for producing photorealistic renderings of the modelled object.

9.4 Deliverables

The following deliverables were produced in this project:

1. Novel approach to function-based model creation, allowing for modularity and interactive shape modelling.
2. Algorithms for efficient and robust operator construction.
3. Visualisation algorithms:
 - Interactive polygonisation for interactive modelling
 - Web-based visualisation
 - Grid-enabled raytracing
4. Prototype interactive modelling system.

The following research papers were published:

Journals:

K. Levinski, A. Sourin, "*Interactive Function-based Shape Modelling*", submitted to *ACM Transactions on Graphics* 9.12.04.

K. Levinski, A. Sourin, "*Interactive function-based shape modelling*", *Computer Graphics & Geometry*, 2004, vol 6, no. 1, pp. 50-74

A. Sourin, Q. Liu, K. Levinski, "*Virtual Campus - It is Fun and Educational*", Leonardo Electronic Almanac, MIT, 2005. Accepted for publication.

Conferences:

A. Sourin, Q. Liu, K. Levinski, "*Virtual Campus - It is Fun and Educational*", Multimedia Art Asia Pacific International Conference 2004, MAAP2004, 2004.

K. Levinski, A. Sourin, "*Interactive Function-Based Shape Modelling for Cyberworlds*", 2004 International Conference on Cyberworlds, 2004, pp. 54-61.

A. Sourin, Lai Feng Min, K. Levinski, "*Web visualization of function-defined shapes*", Second International Workshop on Computer Graphics and Geometric Modelling, CGGM'2003, 2003, vol 2669 / 2003, no. LNCS, ISSN: 0302-9743, January 2003, pp. 428 - 437

F. M. Lai, A. Sourin, K. Levinski, "*Function-based 3D web visualization*", Cyberworlds: Theory and Practice 2002, 2002, pp. 428-435

K. Levinski, A. Sourin, "*Interactive function-based artistic shape modelling*", Cyberworlds: Theory and Practice 2002, 2002, pp. 521-528

K. Levinski, A. Sourin, "*Interactive polygonisation for function-based modelling*", Eurographics 2002, 2002, vol Short Presentation, no. 71-79

K. Levinski, A. Sourin, "*Virtual Function-based Artistic Shape Modelling*", Virtual Environments of Personal Computers, VEonPC 2002, 2002, pp. 52-59

F. M. Lai, A. Sourin, K. Levinski, "*Web Visualization of the Function-defined Virtual Worlds*", Virtual Environments on Personal Computers, VEonPC 2002, 2002, pp. 60-67

Others:

A. Sourin, K. Levinski, "*Interactive function-based shape modeling*" Computer Engineering Research ISSN 0219-5771, 2002, no. 1/02, pp. 1-3.

This work received 2003 Prize of the College of Engineering Exhibition, NTU for the exhibit "Interactive Free-form Shape Modelling".

9.5 Future work

Function-based modelling approach lends itself to parallelisation, therefore possible ways to use parallel architectures should be explored to deal with even more complex objects and modification types on parallel hardware.

Parallel visualisation techniques will be explored further, and extended to collaborative shape modelling. Extension to the Grid architecture presents important problem, which is to be solved for the function-defined shape representation.

Another challenge is to use parallel processing to perform haptic rendering of the complex models, when one computer is not enough to perform fast function evaluations.

Combining existing modelling framework with power of parallel computations will open new horizons for interactive modelling applications.

Bibliography

(109 references)

3DSMaxSoftware "*Interactive modelling application*"

<http://www4.discreet.com/3dsmax/>

Adzhiev V., Cartwright R., Fausett E., Ossipov A., Pasko A., Savchenko V. "*HyperFun project: a framework for collaborative multidimensional F-rep modeling*", Implicit Surfaces '99, Eurographics/ACM SIGGRAPH Workshop (Bordeaux, France, September 13-15 1999), J. Hughes and C. Schlick (Eds.), pp. 59-69.

Akkouche S., Galin E., "*Adaptive Implicit Surface Polygonization using Marching Triangles*", Computer Graphic Forum, 2001, 20(2): pp.67-80

AlibreSoftware "*Collaborative CAD modelling system*" <http://www.alibre.com/>

Allgower E.L., Schmidt P.H., "*An algorithm for piecewise linear approximation of implicitly defined manifolds*". SIAM Jnl. Numerical Analysis, 1985, 22:322-334.

Angelidis A., Cani M.-P., "*Adaptive Implicit Modeling using Subdivision Curves and Surfaces as Skeletons*", ACM Solid Modelling and Applications, June 2002, pp. 45 - 52.

Angelidis A., Jepp P., Cani M.-P., "*Implicit Modeling with Skeleton Curves: Controlled Blending in Contact Situations*", Shape Modeling International, 2002, p. 137.

Avila, R. S. "*Volume haptics*". Computer Graphics (1998), SIGGRAPH'98 Course Notes #01, pp. 103-123.

Avila, R. S., Sobierajski, "*A Haptic Interaction Method for Volume Visualization*" Visualization '96 Proceedings, Oct. 1996, p197.

Baerentzen, J., 1998 Baerentzen, J., "*Octree-based Volume Sculpting*", Proc. Late Breaking Hot Topics, IEEE Visualisation'98, 1998, pp. 9-12.

Bajaj C., Chen J., Xu G., 1995. "*Modeling with cubic α -patches*". ACM Transactions on Graphics 14, 2 (April), 103-133.

Bajaj C., Bernardini F., and Xu G. "*Reconstruction of Surfaces and Functions-on-Surfaces from Unorganized Three-Dimensional Data*" Algorithmica, 19, (1997), 243-261.

Barbier, A. and Galin, E., "*Complex Skeletal Implicit Surfaces with Levels of Detail*", Proc of WSCG, 2004, pp. 35-42.

Behr, J. and Alexa, M., "*Volume visualization in VRML*", Proc. 6th Int Conf on 3D Web technology. ACM Press, 2001, pp. 23-27.

BlobWeb BlobWeb "*BlobTree Modeling System*"

<http://pages.cpsc.ucalgary.ca/~jungle/software/JungleSP/index.html>

Bloomenthal J., "*An Implicit Surface Polygonizer*", Graphics Gems IV, Academic Press, 1994.

Bloomenthal J., "*An Introduction to Implicit Surfaces*", Morgan-Kaufmann, 1997.

Bloomenthal J., Wyvill B., "*Interactive Techniques for Implicit Modeling*", Computer Graphics (1990 Symposium on Interactive 3D Graphics), 1990, 24(2), pp. 109-116.

Cani M.-P., "*Implicit Representations in Computer Animation: a Compared Study*", Proceedings of Implicit Surface '99, Sep. 1999.

Cani M.-P., Hornus S., "*Subdivision Curve Primitives: a New Solution for Interactive Implicit Modeling*", Shape Modelling International, May 2001, p. 82

CEREC3D Sirona Dental 3D modeling software.

Chan S. L., Purisima E. O. "*A New Tetrahedral Tessellation Scheme for Isosurface Generation*." Computers and Graphics, 1998, 22(1):pp. 83-90.

Chen, B. Y. and Nishita, T. , *"Multiresolution streaming mesh with shape preserving and qos-like controlling"*, Proc.7th Int Conf. 3D Web technology. ACM Press, 2002, pp. 35-42.

Cortona *"Cortona VRML browser"*, www.parallelgraphics.com/products/cortona/

Coxeter H. S. M., *"Discrete groups generated by reflections"*, Ann. Math, July 1934 pp. 588-621.

Desbrun M., Cani M.-P. *"Animating Soft Substances with Implicit Surfaces"*, SIGGRAPH 95 Conference Proceedings 29:287-290, 1995

Desbrun M., Cani M.-P., *"Highly deformable material for animation and collision processing"*, Fifth Eurographics Workshop on Animation, Simulation, 1994

DigitalArtForms *"Collaborative modelling environment"*

<http://www.digitalartforms.com/smartVis.htm>

Dobkin, D. A., Levy, S. V. F., Thurston, W. P., Wilks, A. R. *"Contour tracing by piecewise linear approximations,"* 1990, 9(4), pp. 389-423,

Ehmann S., Gregory A., and Lin M.. *"A Touch-Enabled System for Multiresolution Modeling and 3D Painting"* Journal of Visualization and Computer Animation, 12:3, 2001, pp. 145-157.

FREPWeb *"F-Rep web-site"* <http://wwwcis.k.hosei.ac.jp/~F-rep/>

Ferley E., Cani M.-P., Gascuel J.-D., *"Resolution Adaptive Volume Sculpting"* Graphical Models (GMOD), 2001, 63:3, pp. 459-478.

Ferley E., Cani M.-P., Gascuel J.-D., *"Virtual Sculpture"*, The Visual Computer 2000, 16:8, pp. 469-480.

Fogel, E. , Cohen-Or D., Ironi, R. , and Zvi, T. , *"A web architecture for progressive delivery of 3d content"*, Proc. 6th Int. Conf. on 3D Web technology. ACM Press, 2001, pp. 35-41.

- Foster. I. *"The Grid: A New Infrastructure for 21st Century Science"* Physics Today, 2002, 55(2):42-47.
- Friskin S.F., Perry, R.N., Rockwood, A.P. and Jones, T.R., *"Adaptively Sampled Distance Fields: A General Representation of Shape for Computer Graphics"*, SIGGRAPH'2000, pp. 249-254.
- Funkhouser T., Min P., Kazhdan M., Chen J., Halderman A., Dobkin D., and Jacobs, D., *"A Search Engine for 3D Models"* ACM Transactions on Graphics, January 2003, 22(1), pp. 83-105.
- Galin E., Akkouche S., *"Fast Surface Reconstruction from Contours using Implicit Surfaces"*, Implicit Surfaces'98 Conference, Seattle, USA, June 1998, pp. 139-144.
- Galin E., Akkouche S., *"Incremental Polygonization of Implicit Surfaces"*, Graphic Models and Image Processing, 2000, 62, pp.19-39.
- Galin E., Leclercq A., Akkouche S., *"Blob-Tree Metamorphosis"*, Implicit Surfaces'99 Conference, , Bordeaux, France, September 1999, 4: pp. 9-16
- Galin E., Leclercq A., Akkouche S., *"Morphing the BlobTree"*, Computer Graphic Forum, 19(4): November 2000, pp.257-270.
- Galyean T., Hughes J., *"Sculpting: An Interactive Volume Modeling Technique"*, SIGGRAPH'91, pp. 267-274.
- Hart J.C., Bachta E., Jarosz W., Fleury T., *"Using particles to sample and control more complex implicit surfaces"* Proc. Shape Modeling International 2002, May 2002, p.129.
- Hartmann E., *"A Marching Method for the Triangulation of Surfaces"*, The Visual Computer 1998, 14:3, pp.95-108
- Hornus S., Angelidis A., Cani M.-P., *"Implicit Modelling Using Subdivision-curves"*, The Visual Computer, 2003, 2-3, vol. 19, pp. 94-104.

Huang, C., Qu, H., and Kaufman, A. "*Volume rendering with haptic interaction*" In Proceedings of the Third PHANTOM Users Group Workshop (1998), vol. 3, pp. 14-18.

Igarashi T. and Hughes J. "*Smooth Meshes for Sketch-based Freeform Modeling*" ACM SIGGRAPH 2003 Symposium on Interactive 3D Graphics, ACM I3D'03, Monterey, California, April 2003, 27-30, pp.139 142,

IFWeb "*Interactive Function-based Shape Modeling Project Web-site*"
<http://www.ntu.edu.sg/home/assourin/CompArt.html>

ImpactXoft "*Collaborative CAD software*" <http://www.impactxoft.com/solutions/spd.asp>

Jing Hua, Hong Qin, "*Haptics-Based Dynamic Implicit Solid Modeling*" IEEE Transactions on Visualization and Computer Graphics Volume 10 , Issue 5 (September 2004) pp: 574 - 586.

Karkanis T., Stewart A.J., "*Curvature Dependent Triangulation of Implicit Surfaces*", IEEE Computer Graphics Applications, 2, 2001, pp. 60-69.

Karpenko O., John F. Hughes, and Ramesh Raskar. "*Free-form sketching with variational implicit surfaces*" Computer Graphics Forum, September 2002, 21(3): pp. 585-594,

Kazakov M., Adzhiev V., Pasko A. "*Fast Navigation Through an FRep Sculpture Garden*", Shape Modeling International 2001, 2001, pp.104-113.

Klosowski J. T., Held M., Mitchell J., Sowizral H., Zikan K., "*Efficient Collision Detection Using Bounding Volume Hierarchies of k -DOPs*", IEEE Transactions on visualization and computer graphics, 1996, 4(1), pp. 21 36.

Kronrod, B. and Gotsman, C. , "*Efficient coding of nontriangular mesh connectivity*", Proc. Pacific Conf on Comp Graphics and Applications 2000, pp. 235-242.

Levinski, K. and Sourin, A. *"Interactive function-based artistic shape modeling"*, Proc of International Symposium Cyber Worlds: Theory and Practice 2002, Japan, IEEE CS, 2002, pp. 521-528.

Levoy M., Whitted T., *"The Use of Points as a Display Primitive"*, Technical Report 85-022, Computer Science Department, University of North Carolina at Chapel Hill, January, 1985.

Lewis, J.P. *"Algorithms for solid noise synthesis"*, Proc. 16th Annual Conf on Comp Graph and Int Techniques, 1989, pp.263-270.

Liu, Q. Sourin, A. *"Analytically-defined Collaborative Shape Modeling in VRML"*, 2004 International Conference on Cyberworlds, Tokyo, 18-20 November, 2004, pp. 70-77.

Lorensen W.E., Cline H.E., *"Marching Cubes: a High Resolution 3D Surface Reconstruction Algorithm"*, SIGGRAPH'87, 1987, pp. 163-169.

MayaSoftware *"Integrated 3D modeling, animation, effects, and rendering solution from Alias"* <http://www.alias.com/eng/products-services/maya/index.shtml>

McCormack J., Sherstyuk A., *"Creating and rendering convolution surfaces"*, Computing Graphics Forum, 1998, vol. 17, No.2, pp. 113-120

Mizuno S., Okada M., Toriwaki J., *"Virtual Sculpting and Virtual Woodcut Printing"*, The Visual Computer, 1998; vol. 14 pp. 39-51.

Ohtake Y., Belyaev A., Pasko A. *"Dynamic meshes for accurate polygonization of implicit surfaces with sharp features"*, Shape Modeling International 2001, IEEE Computer Society, 2001, pp. 74-81.

Otaduy A., Lin C. *"Sensation preserving simplification for haptic rendering"*, ACM Transactions on Graphics (TOG) Vol. 22 , 3 (July 2003) p 543 - 553

PGraphics PGraphics *"ParallelGraphics Company"* <http://www.parallelgraphics.com>

POVray Persistence of Vision Raytracer, <http://www.povray.org>

- Pasko A., Adzhiev V., Schmitt B., Schlick C., "*Constructive hypervolume modeling*", Graphical Models, special issue on Volume Modeling, 64(2), 2001, 63 vol.6 pp. 413-442.
- Pasko A., Adzhiev V., Sourin A., Savchenko V. "*Function representation in geometric modeling: concepts, implementation and applications*", The Visual Computer, vol.11, No.8, 1995, pp. 429-446.
- Pasko A., Savchenko V., Sourin A., "*Synthetic carving using implicit surface primitives*", Computer-Aided Design, Elsevier, vol. 33, No. 5, April 2001, pp. 379-388.
- Pasko G., Pasko A., Ikeda M., Kunii T., "*Bounded blending operations*", Shape Modeling International 2002, Banff (Canada, May 17-22), IEEE Computer Society, 2002, pp. 95-103.
- Pauly M., Keiser R., Kobbelt L., Gross M., "*Shape Modeling with Point-sampled Geometry*" ACM Transactions on Graphics, 22(3), pp. 641-650.
- Peachey, D. R. , "*Solid texturing of complex surfaces*", Proc. 12th annual conference on Computer graphics and interactive techniques, ACM Press, 1985, pp. 279-286.
- Perry R.N., Frisken S.F., "*Kizamu: A System for Sculpting Digital Characters*", SIGGRAPH'01, pp. 47-56.
- Rusinkiewicz S., Levoy M., "*QSplat: A Multiresolution Point Rendering System for Large Meshes*", SIGGRAPH'00, pp. 343-352
- Rusinkiewicz S., Levoy M., "*Streaming QSplat: A Viewer for Networked Visualization of Large, Dense Models*", I3D 2001, pp. 63-68
- SFFWeb "*Sensible FreeForm Modeling System*"
<http://www.sensible.com/freeform/freeform.htm>
- Savchenko V., Pasko A., "*Transformation of functionally defined shapes by extended space mappings*", The Visual Computer, vol. 14, No. 5/6, 1998, pp. 257-270.

- Sherstyuk A., *"Convolution Surfaces in Computer Graphics. Ph.D. dissertation"* School of Computer Science and Software Engineering, Monash University, Australia.
<http://www.ugcs.caltech.edu/~andrei/thesis/>
- Skala V., *"Precision of Iso-Surface Extraction from Volume Data and Visualization"*, Algoritmy'2000 Int.Conf., Slovakia, pp.368-378.
- Sourin A., *"Functionally Based Virtual Computer Art"*, Proc of the 2001 ACM Symposium on Interactive Computer Graphics, I3D2001, 2001, pp.77-84.
- Sourin A., *"Functionally Based Virtual Embossing"*, The Visual Computer, 2001, 17:4, pp.258-271.
- Sourin A., Pasko A. *"Function representation for sweeping by a moving solid"*, IEEE Transactions on Visualization and Computer Graphics, vol.2, No.1, March 1996, Special issue on solid modeling, pp.11-18.
- Sourin A., Pasko A., Savchenko V., *"Using real functions with application to hair modelling"*, Computers and Graphics, 1996, vol.20, no.1, pp. 11-19.
- Sutter H., *"The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software"*, Dr. Dobbs's Journal, 30(3), March 2005, to appear
- Stander B.T., Hart J.C., *"Interactive re-polygonization of blobby implicit curves"* Proceedings of Western Computer Graphics Symposium. Mar. 1995. pp. 33-44
- Stander T., Hart C., *"Guaranteeing the Topology of an Implicit Surface Polygonization for Interactive Modeling"*, SIGGRAPH'97, 1997, pp. 279 286.
- Terzopoulos D., Qin H., *"Dynamic NURBS with Geometric Constraints for Interactive Sculpting"*, ACM Transactions on Graphics, 13:2, 1994, pp. 103 136.
- Theul T., Moller T., Meister Groller E., *"Optimal Regular Volume Sampling"*, IEEE Vizualization'2001, pp. 91-98

Thiebaut M., Tangmunarunkit H., Czajkowski K., Kesselman C., *"Scalable Grid-based Visualization Framework"*, Submitted for High Performance Distributed Computing 2004

Treece G. M., Prager R. W., A. H. Gee. *"Regularised marching tetrahedra: Improved Iso-Surface Extraction"* Computers and Graphics, 1999, 23:4, pp. 583-598.

TrueSpaceSoftware *"Advanced Modelling Applications"* <http://www.caligari.com/>

Turk G., O'Brien J. F., *"Shape Transformation Using Variational Implicit Functions"* The Proceedings of ACM SIGGRAPH 99. Los Angeles, California, August 8-13, pp. 335-342.

VRML *"Virtual Reality Modeling Language Specification"*
<http://www.web3d.org/x3d/specifications/vrml>

Verse *"Collaborative modeling system"* <http://www.blender.org/modules/verse/index.php>

WWW3 *"HTML Markup"* <http://www.w3.org/MarkUp>

Wang S., Kaufman A., *"Volume Sculpting"*, SIGGRAPH'95, pp. 151-156.

WebCMP *"Virtual Campus"* <http://www.ntu.edu.sg/home/assourin/VirCampus.html>

WebFLab *"FVRML interactive lab"*
http://www.ntu.edu.sg/home/assourin/vircampus/fshapelab/contact_fshapelab.htm

WebFVRML WebFVRML *"Function-based VRML"*
<http://www.ntu.edu.sg/home/assourin/FVRML.htm>

Welch W., Witkin A., *"Free form shape design using triangulated surfaces"*, Computer Graphics (Proc. SIGGRAPH '94), Vol. 28, pp. 247-256.

Witkin A., Heckbert P., *"Using Particles to Sample and Control Implicit Surfaces"*, SIGGRAPH'94, 1994, pp. 269-277.

Wyvill B., Guy A., Galin E., "*Extending the CSG Tree (Warping, Blending and Boolean Operations in an Implicit Surface Modeling System)*", Computer Graphics Forum, June 1999, 18(2), pp. 149-158.

Wyvill G. McPhetters C., Wyvill B., "*Data Structure for Soft Objects*", The Visual computer, 2(4), 1986, 227-234

Wyvill G., Wyvill B., and McPheeters C., "*Solid texturing of soft objects*", CG International '87 on Computer graphics 1987. Springer-Verlag New York, Inc., 1987, pp. 129-141.

Wyvill, B. and Guy, A. , "*The blob tree: Implicit modeling and VRML*", Proc. of Int Conf From the Desktop to the Webtop: Virtual Environments on the Internet, WWW and Networks, 1997, pp. 193-206.

Xu G., Bajaj C. and Huang H. "*C¹ Modeling with A-patches from Rational Trivariate Functions*" Computer Aided Geometric Design, 18:3(2001), 221-243

Zachmann G., "*Rapid Collision Detection by Dynamically Aligned DOP-Trees*", Proc. of IEEE Virtual Reality Annual International Symposium; VRAIS '98. Atlanta, Georgia; March 1998 p. 90

ZBrushSoftware "*Hybrid 3D modelling software*" <http://pixologic.com/home/home.shtml>

Zhang X., C. Bajaj, and Ramachandran V., "*Parallel and out-of-core view-dependent isocontour visualization using random data distribution*" Proceedings of Data Visualisation'02, pages 9-18