# Interactive Shape Modelling

Marc Alexa, Alexis Angelidis, Marie-Paule Cani, Karan Singh, Denis Zorin

# EG 2005 Tutorial Notes "Interactive Shape Modeling"


## Summary:

The course will present the state-of-the-art in digital modeling techniques, both in commercial software and academic research. The goal of this course is to impart the audience with an understanding of the big open questions as well as the skills to engineer recent research in interactive shape modeling applications.


## Presenters:


**Marc Alexa**
Assistant Professor, Department of Computer Science
Darmstadt University of Technology (TUD)
alexa@informatik.tu-darmstadt.de
http://www.dgm.informatik.tu-darmstadt.de

**Alexis Angelidis**
Department of Computer Science
University of Otago
alexis@cs.otago.ac.nz
http://www.cs.otago.ac.nz/postgrads/alexis/

**Marie-Paule Cani**
Professor, Department of Computer Science
Co-director, GRAVIR
Institut National Polytechnique de Grenoble (INPG)
Marie-Paule.Cani@imag.fr
http://www-evasion.imag.fr/Membres/Marie-Paule.Cani/index.gb.html

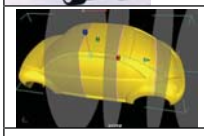
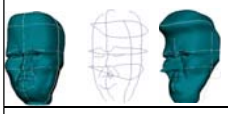
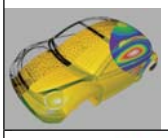
**Karan Singh**
Associate Professor, Department of Computer Science
University of Toronto
karan@cs.toronot.edu
http://www.dgp.toronto.edu/~karan/

**Denis Zorin**
Associate Professor, Department of Computer Science
New York University
dzorin@mrl.nyu.edu
http://mrl.nyu.edu/~dzorin/

## Abstract:

Computer Graphics continues to battle the challenging question: "**How quickly and effectively can a designer transform a mental concept into a digital shape, which is easy to refine and reuse?**" Traditional techniques of sculpting and sketching continue to be among the quickest and most expressive ways for designers to visually manifest their ideas. This course traces the evolution of interactive shape design from traditional media to the state of the art in digital modeling techniques, both in commercial software and academic research. The course will cover the gamut of hardware devices and interaction paradigms used in digital modeling and their underlying mathematical representations of shape. The audience will be presented with the properties of various implicit, explicit and hybrid shape representations and the capabilities, limitations and implementation details of current algorithms for interactive shape creation and manipulation. The goal of this course is to impart the audience with both an understanding of the big open questions as well as the skills to engineer recent research in interactive shape modeling applications.

## Visual Course Agenda:

| | |
|---|---|
| | **Marie-Paule Cani:**<br>**Introduction & Overview** |
|  | **Karan Singh:**<br>**Industrial motivation and approaches**<br>Pages 3-10 |
|  | **Karan Singh:**<br>**Global space & Free form deformations** |
|  | **Karan Singh:**<br>**Wires** |
|  | **Denis Zorin:**<br>**Mathematical representations of shape for modeling** |
|  | **Denis Zorin:**<br>**Multiresolution modeling**<br>Pages 11-36 |
|  | **Marc Alexa:**<br>**Mesh editing based on discrete Laplace and Possion models**<br>Pages 37-48 |
|  | **Marie-Paule Cani:**<br>**Volumetric and implicit surface based shape modeling**<br>Pages 49-66 |
|  | **Alexis Angelidis:**<br>**Gesture-based shape modeling**<br>Pages 67-89 |
|  | **Marie-Paule Cani:**<br>**Modeling by Sketching**<br>Pages 90-102 |
| | **Marie-Paule Cani:**<br>**Future directions and concluding remarks** |

# Industrial motivation for interactive shape modeling: a case study in conceptual automotive design

Karan Singh

Computer Science, University of Toronto.

**Abstract**

*As Computer Graphics makes rapid strides in various aspects of digital shape modeling it is easy to lose perspective of the larger motivations for digital shape modeling in design and animation. This chapter provides a high level view of shape modeling illustrated within the space of conceptual automotive design. Automotive design provides a unique perspective on digital shape modeling, where digital models are critical to downstream production processes but automotive designers almost exclusively work with sketches, clay and other traditional media. Design iterations that transition between physical and digital representations of a prototype are thus a big bottleneck in the industrial design lifecycle. In this chapter we propose a top-down approach, starting with the design desirables and suggesting modeling paradigms that harness skills and creativity of designers.*

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Geometric modeling, User Interaction

## 1. Introduction

Computer Graphics continues to battle the challenging question: **How quickly and effectively can a designer transform a mental concept into a digital object, that is easy to refine and reuse?** If hearing, speech and sight are analogous to the audio IN, audio OUT and video IN of an electronic device, the essence of our problem is that humans do not have an explicit video OUT.

This is a problem of great industrial importance today. Designers almost exclusively prefer traditional design techniques of sculpting and sketching, instead of computer aided digital styling tools that operate on mathematical representations of geometry. Most manufacturing processes, however, use digital models making design iterations a big bottleneck in an industrial design lifecycle. The majority of industry-based surface modeling research is, therefore, focused on incrementally making existing digital styling tools more designer friendly, while the root of the problem lies deeper.

The fundamental pitfall is that current digital tools are unable to decouple the creative process from the underlying mathematical attributes of the surface representation. As an example, when modeling an object using a network of bi-cubic or higher order polynomial spline surface patches, concepts like patch resolution, topological connectivity and continuity across surface patches constrain the creativity of the designer. The solution is to start from scratch with a designers perspective and develop computer interaction paradigms that harness their skills and creativity. These interaction techniques will in turn define the requirements of the underlying mathematical representations of geometry. Studies have shown that designers and people in general abstract shape as aggregations of complex surface attributes, that we will collectively call *surface-features* that are independent of any geometric model representation.

Conceptual modeling should, therefore, focus among other things on the development of new mathematical representations or adapting existing ones, to capture the essence of shape as perceived by designers. To be able to make tangible progress towards such a goal we must first mathematically quantify this *essence of shape* in terms of geometric surface-features. Design methodologies in industry are both complex and diverse and it is important to have a well-defined process to study and within which to evaluate proposed solutions. This chapter will focus on the early stages of conceptual automotive design, which has been slow in adapting to the use of digital styling tools, despite being a trendsetter in digital modeling for the engineering phase of

its design lifecycle. Design iterations and revisions that transition between physical and digital representations of a prototype are currently one of the big bottlenecks in the design lifecycle of an automobile.

The remainder of this chapter is organized as follows: Section 2 discusses the generally desirable properties of systems for conceptual design. Section 3 illustrates these properties within the automotive design space. Section 4 then proposes a framework for conceptual automotive design based on commonalities observed from the current workflows in practice at various automotive design centers. Current trends in geometric shape representation and interactive shape modeling are then discussed in the context of their applicability to the automotive design framework. Section 6 provides concluding remarks.

## 2. Conceptual modeling desirables

Newer generations of industrial designers are increasingly savvy with digital modeling techniques. Their design education, however, continues to be grounded in traditional sketching and sculpting techniques, which embody a number of desirable properties that any digital modeling system should embrace.

- **Abstraction from underlying surface math**
  Most mathematical surface manifolds are represented at some point by a discrete set of points (control points for parametric or subdivision surfaces, vertices for polygon meshes) that often become handles for shape manipulation. This not only exposes the designer to the understanding of the mathematics and topology of the shape representation but also forces the learning and usage of tools that may not have been considered intuitive when decoupled from the geometric representation. Designer interaction paradigms should thus be defined such that the user is oblivious of the underlying mathematical surface representation. [Sin99] provides an example of such design, where the user interacts with sweeps just like in the physical world (see Figure 11) but the underlying curve manipulation is accomplished through splicing and fitting cubic spline curve segments.
- **Invite interactive creative exploration**
  Often digital modeling tools are made easy to use by narrowing their scope to a specific design space. As examples, two successful sketching systems Teddy [IMT99] and SKETCH [ZHH96] simplify the inference of a 3D model from sketched curves by making assumptions of the user design space. While SKETCH is tuned to create simple analytic shapes, Teddy is focussed on the creation of smooth organic forms. Design innovations are often the result of serendipitous exploration. Design tools should thus be interactive and easy to use without compromising their power of creative expression, as far as possible. A major advantage of interactive digital modeling tools is the ability to undo an operation allowing users to experiment without fear of making mistakes. It is thus important that increased complexity and sophistication of a modeling tool does not come at the expense of its interactivity.
- **Allow for precision and constraints**
  Industrial design models typically need to adhere to various engineering constraints before they can be manufactured downstream. Integrating such constraints early into the conceptual design process eliminates costly iterations in the design lifecycle, where models need to be redesigned because they violate some insurmountable constraint.
- **Workflow mimics traditional design media**
  Sketching and sculpting with physical media are both easy to use and creatively unfettered approaches to visual communication. Digital modeling techniques could do well to capture the modalities that make these approaches successful. Systems such as [IMT99],[TBSR04], for example, strive towards the modeless fluidity of sketching and exploit traditionally used gestures to invoke various commands as part of the sketching process
- **Leverages domain expertise**
  Designers often have skills in using specialized physical devices for conceptual design that digital modeling approaches should attempt to benefit from. Many automotive designers, for instance are proficient tape artists [BFKB99], a skill that allows them to lay out designs on large surfaces using tape of varying thickness and tension (see Figure 4).

## 3. Automotive design process

The current automotive design lifecycle is 3-4 years, of which as much as half is spent in the early stages of conceptual design. Automotive designers largely work in traditional media and hand their designs off to modellers. Modellers are technically skilled people that create digital models with surfacing software, using the physical designs as a visual reference. These designs are then evaluated both digitally and physically using rapid prototyping technology and the entire process iterates towards a converging design. In addition to the general desirables of a conceptual modeller there are many aspects of shape modeling that make the automotive design space unique.

**Figure 1:** *Curvature continuous surfaces*

- **Curvature continuous shapes**
  Automobile surfaces display a high degree of continuity,

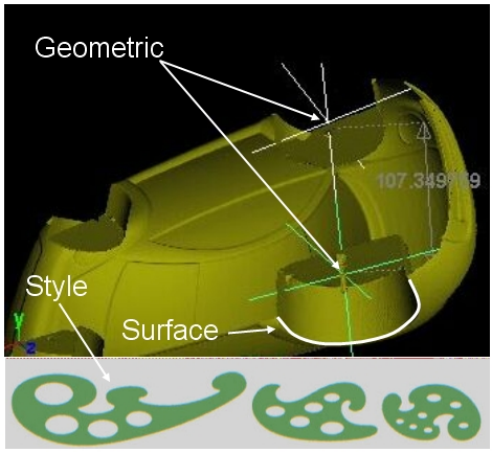



**Figure 2:** *Automotive design constraints*

**Figure 3:** *Editing a physical model prototype*

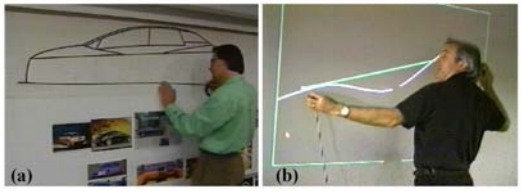



**Figure 4:** *Digital Tape Drawing [BFKB99]*

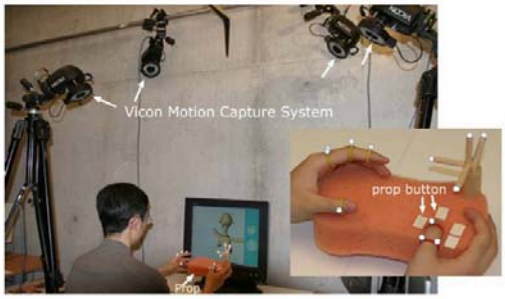



**Figure 5:** *Sculpting with motion capture [She04]*

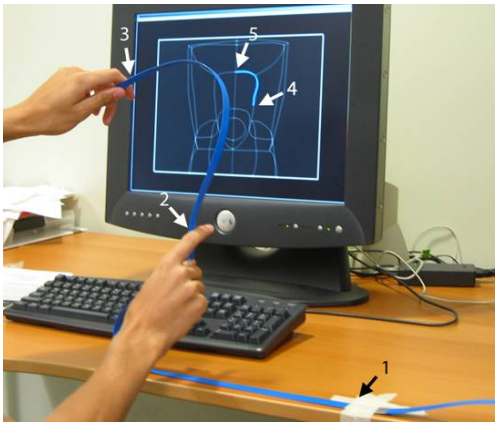



**Figure 6:** *Manipulating curves with ShapeTape [GBS03]*

barring a few sharp features that run along the character lines of the design. Many automotive designers think in terms of the shape, size and location of specular highlights on the design and for these highlights to be smooth and unbroken, the surfaces needs to be at least $C^2$ continuous (see Figure 1).

- **Character or flow lines captured intrinsically**
  Character and flow lines that define the principal curvatures along surfaces are an important characteristic of automotive design.
- **Embodies geometric, surface and style constraints**
  While automobile design can be far more free-form than say marine or airplane design (due to fluid and aerodynamic constraints), the designs must adhere to certain constraints. These constraints can be geometric, such as hard points or dimensions on the engineered design, surface constraints, such as the circular shape of wheel arches, or stylistic, such as a signature look and feel for an entire family of automobiles (see Figure 2).
- **Flexible re-use of legacy data**
  Automotive designs do not change radically over short periods of time. It is thus important for design tools to facilitate the evolution of models and support the re-use of parts of designs that have already been engineered. Operations such as cut and paste play an important role is data re-use (see Figure 7).
- **Interfaces digital and physical modeling**
  Given the production lifecycle and costs that go into automotive design it is unlikely that a design will ever be approved without the creation of physical prototypes. Design updates are often made on these prototypes making it important to build better bridges between physical and digital modeling techniques (see Figure 3).
- **Large scale displays and novel interaction devices**
  Equally important to the automotive design process are design visualizations at the true scale of the models. This implies the need for large scale display devices [WB00]

that are capable of displaying an automobile to scale. A number of high degree of freedom input devices today such as a flock of birds [TG02],[LKG*03], motion capture systems [She04] (see Figure 5) and ShapeTape [GBS03] (see Figure 6) show potential at emulating current large scale modeling techniques in practice in automotive design (see Figure 4).

## 4. A proposed framework for automotive design

We now distil these observations and a study of various automotive design pipelines in practice into a proposed framework for conceptual automotive design illustrated in Figure 7. We broadly structure current and projected modeling technology and techniques into three stages of rough model generation, model refinement and model presentation.

### 4.1. Rough Model Creation

Sketches (on paper or using a pen and tablet), physical sculpture, character lines and basic parameterized shapes typically form the creative input to this earliest phase of digital model creation.

A big challenge in this stage is the ability to take such varied input and transform it appropriately to consistently represent parts of the model in a common 3D space. The side view sketch in Figure 8, for example, needs to be scaled to be consistent in space with top and front view sketches. Early design sketches and sculpts may also have inconsistent or missing information in parts of the design that are resolved with model refinement. Determination of the intended fidelity of different parts of the models in the different pieces of input is thus a non-trivial problem. Precise engineering criteria are left out of the initial design input to leave the designer unencumbered creatively, but they are part of the input to the technique that constructs the rough model from the design input. As an example, while a designers sketch may only adhere roughly to engine block dimensions, the rough model created should make precise allowances for the engineering constraints. The rough model should also have the ability to determine a set of surface-features on the model that can be edited at this stage to make larger stylistic changes to the model.

Physical 3D prototypes can be scanned [Cur] and the data structured using reverse engineering techniques [VK96]. Creating 3D models from 2D sketches is far a trickier problem [EHBE97],[Low91] but sketches do tend to have surface-features and character lines explicitly depicted. In the final analysis there is likely to be an element of user interaction in the creation of a rough digital model from the given design input [TBSR04]. The success of a technique is likely to be in its judicious use of user input to help resolve ambiguities in the given input.

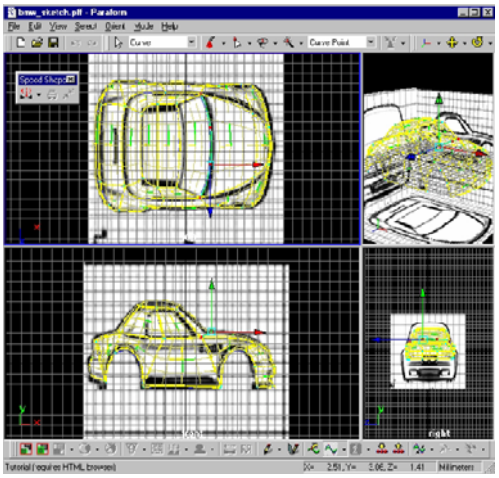


**Figure 8:** *Aligning orthographic sketches into a common 3D space*

### 4.2. Model Refinement

Once a rough digital model that has been structured and parameterized with respect to various surface-features and character lines, it is refined and embellished using tools that capture the design desirables of Section 2 and Section 3. A good suite of tools is one that would provide good coverage over the following functionality (see Figure 7):

- Constraint preserving global deformations [LKG*03].
- Cut and paste [BMBZ02].
- Surface-Feature based editing [SCOL*04].
- Local deformation and sculpting of object detail [MTH94].

### 4.3. Model Presentation

Design reviews on automobiles typically take place on life-sized displays or physical models built to scale with realistic materials and lighting. Indeed many designers conceptualize models based on the interplay between shape, shadows and highlights [PPF98]. The importance of this observation is twofold. First, digital modeling techniques should incorporate surface evaluation tools like curvature comb plots (see Figure 9), reflection and zebra maps, and high quality rendering early in the modeling process. Second, techniques that create lighting or edit shape based on the direct manipulation of shadows and highlights [PP97] are worthwhile additions to an automotive designers toolbox.

Once a version of a digital model is approved it is typically used to generate a physical prototype and is also subjected to a number of design and engineering fidelity checks that may result in further iterations of the design cycle.

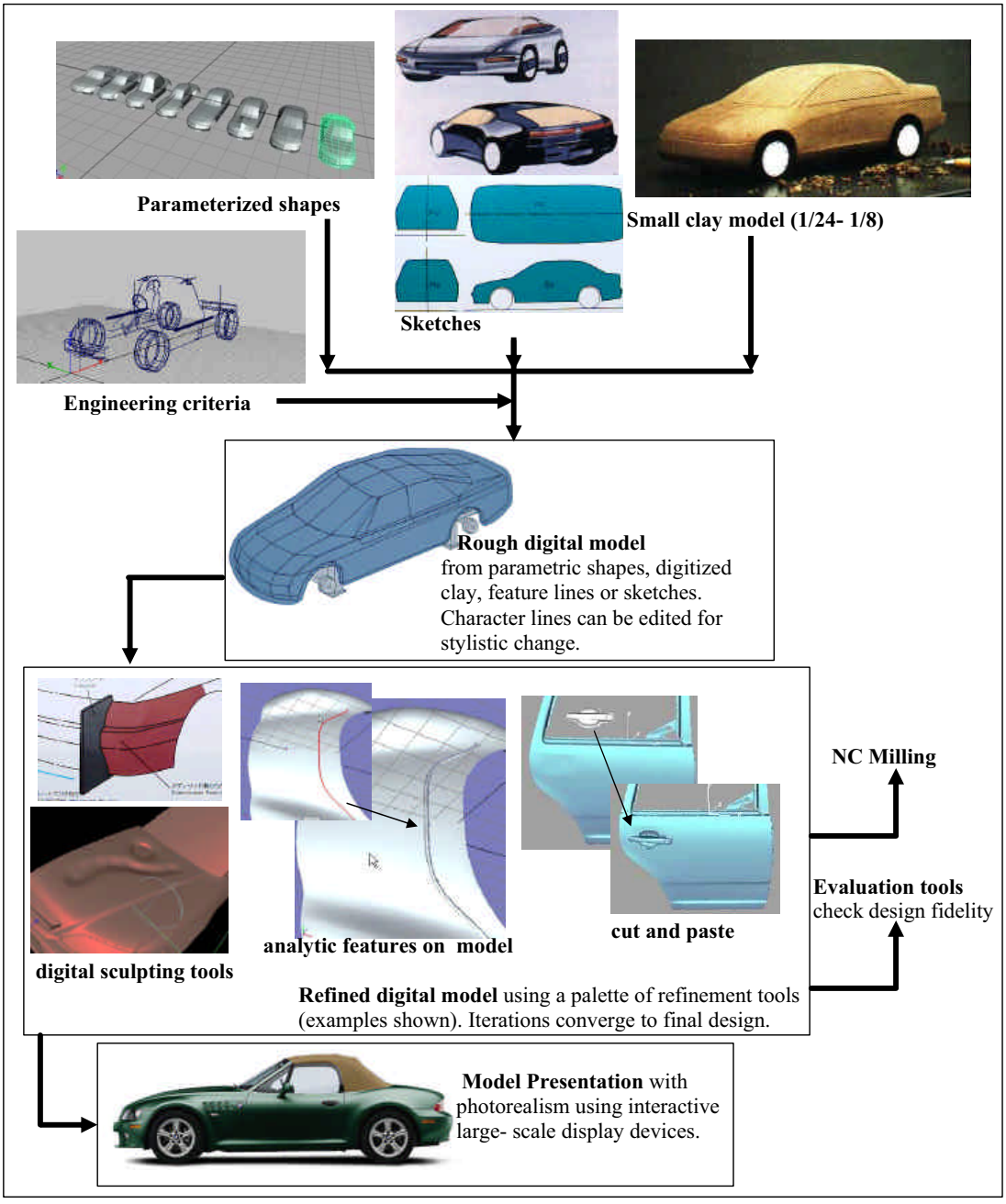



**Figure 7:** *Proposed Automotive Design Workflow*

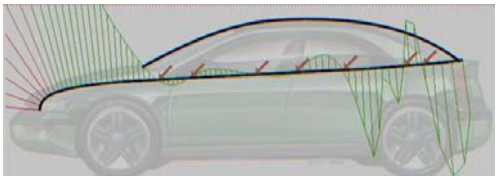


**Figure 9:** *Curvature comb plot showing curvature discontinuities*

## 5. Current modeling trends

It is clear that conceptual design in the future will require the co-existence of both physical and digital representations of objects. Physical models are converted to digital models using scanning devices [Cur] and other data acquisition technology. Manufacturing processes such as milling, injection molding and rapid prototyping machines give physical form to digital models, in materials as varied as metal, synthetic foam and clay. The data acquisition technology and modeling paradigms used, the manufacturing techniques employed and last but not least the industrial application, all critically affect the choice of geometric representation.

### 5.1. Geometric surface representations

There are a number of ways of representing the surface of an object that are in active use in computer graphics today. The important ones are: Point clouds, Polygon meshes, Parametric curve and surface patches, Subdivision surfaces, Analytic shape primitives (cubes, spheres, cylinders for example) with CSG operations and Implicit surfaces (see Figure 10).

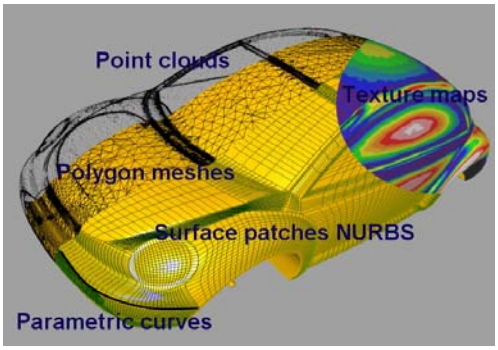



**Figure 10:** *Various geometric representations used in automotive design*

Historically, continuous parametric curve and surface patches constructed from piecewise polynomial splines, have been used to represent industrial design objects [Far01]. There were many reasons for this. Cubic and higher order polynomials allow surfaces to be controlled with $C^2$ continuity. The curves and surfaces have an inherent parametric structure and the control point data structure with patch topology is fairly compact. As a result, Non-Uniform Rational B-Splines (NURBS) are an industrial standard today.

A point-cloud [SR00], in contrast is a dense point sampling of a surface without any explicit surface elements. A point-cloud where the points are connected by polygon elements to form a surface manifold is called a polygon mesh. Polygon meshes provide a faceted linear approximation to continuous object surfaces. Properties such as surface continuity and a structured parameterization are not inherent but can be imposed externally if the mesh resolution is high enough. The lack of computing power to handle high-resolution polygon meshes made them unsuitable for industrial design applications in the past. Subdivision curves and surfaces have existed since the early 70s [Cha] but have only recently drawn great interest in the computer graphics community as a way of bridging the complementary properties of parametric surfaces and polygon meshes. Subdivision surfaces have $C^2$ discontinuities at extraordinary vertices (vertices with a valence other than 4), making them far more popular in film and gaming applications than as a framework to represent surfaces for industrial design. While analytic shapes like spheres and cylinders are commonly found in various industrial objects, they are too restrictive by themselves as a general framework to represent complex shapes accurately.

Finally, implicit surface is a term that encompasses all objects that are represented mathematically as the solution to an implicit equation of points in a Cartesian space [Blo97]. Implicit surfaces are often built as an algebraic combination of analytic primitives. Implicit surfaces are a very compact, continuous representation and are a popular choice for interactive shape sculpting techniques since they deal automatically with changes in genus and topology of objects. Implicit functions such as radial basis functions (RBF), have also been successful in approximating and fitting a continuous surface model to sparse or irregularly sampled data [JC01]. The problem with implicit surfaces historically has been the sampling search required to render the surface represented by the implicit function. This lack of an explicit parameterization also makes local morphological operations hard to define computationally. It should be evident from this last paragraph, that no one existing surface representation technique can be considered to be a comprehensive superset of the others in terms of desirable properties for the design of objects.

Recent advances in graphics hardware and computing power have made it possible to render millions of points and triangles in real-time [SR00]. As mentioned earlier, many industrial designers prefer to build physical prototypes in a real workshop to quickly resolve shape and form in 3D. These prototypes are transformed to digital models by 3D shape

acquisition technology, typically as point clouds of widely varying sampling patterns and densities. These are usually converted into dense polygon meshes [Cur]. Most continuous surface representations, parametric or implicit are also tessellated to a polygon mesh prior to rendering. Meshes, however, are often unstructured and irregularly sampled and display artifacts such as degenerate, flipped or sliver faces, undesirable holes and widely varying polygon sizes. Further, mesh models often need to be parameterized, segmented and built in parts as an assembly of complex shapes. The chief reason for this is that point clouds and polygon meshes do not directly incorporate the notion of surface-features.

In summary, there is a current trend towards preserving hybrid or multiple representations of shape so as to benefit from the complementary properties of different geometric representation schemes.

### 5.2. Devices for display and interaction

It is evident that the standard keyboard and mouse metaphor falls short in the design domain. Automotive design is a prime example, where design prototypes are close to the actual size of an automobile. Large format displays enable a designer to create, manipulate, and view the design of an automobile at full size. They are currently in active use in automotive design centers, strictly as an interface for design presentation but show promise for collaboration and real-time editing of the design by a team, during design reviews.

For novel displays to be used successfully in the design domain they must work well with input technology that conveys human design intent. Haptic input technology, such as the Phantom (Sensable Tech Inc.) allows us to investigate more effective digital sculpting systems [MTH94]. Consequently, our surface representations need to be able to easily handle rapid changes in curvature and even genus of the sculpted object, as well as represent the internal volume of the object. High degree of freedom input devices such as ShapeTape [GBS03] and a motion capture system [She04] can be used to instrument the types of curve and surface physical tools that designers use in the traditional design industry (like the steels car designers use to shape clay) (see Figure 11). Motion-capture and 3D scanning systems can also be used to interactively create and animate digital models of physical objects [Liu03].

In general trends in conceptual shape modeling are moving in the positive direction of decoupling the interaction techniques from the underlying surface representation. Research on surface representation similarly is working towards structures which have the topological flexibility of unstructured data but also capture high level shape concepts of character lines and other surface features.

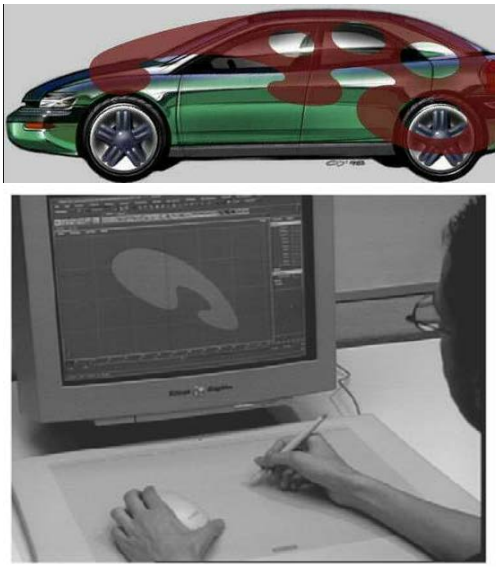


**Figure 11:** *Curve modeling with sweeps [Sin99]*

## 6. Conclusion

In this chapter we have presented industrial motivation for digital conceptual modeling tools. We have illustrated various desirable properties of a conceptual modeller within the automotive design space. We have defined a framework to structure the generally practiced automotive design workflow and touched upon current modeling representations and interfaces within this context. Various chapters in this tutorial further address these issues and propose detailed solutions to the questions raised in this chapter.

### Acknowledgements

Many thanks to Ravin Balakrishnan, Tovi Grossman, Xia Liu, Jia Sheng and members of the DGP lab, for their help with the work presented in this chapter. Thanks also to Paraform Inc. and Alias Inc. for their support of the field work and research presented here. Ongoing work at DGP in conceptual design is supported by MITACS.

### References

[BFKB99] BALAKRISHNAN R., FITZMAURICE G., KURTENBACH G., BUXTON W.: Digital tape drawing. In *Proceedings of the 12th annual ACM symposium on User interface software and technology* (1999), ACM Press, pp. 161–169. 2, 3

[Blo97] BLOOMENTHAL J.: *Introduction to Implicit Surfaces*. Morgan Kauffmann, 1997. 6

[BMBZ02] BIERMANN H., MARTIN I., BERNARDINI F.,

ZORIN D.: Cut-and-paste editing of multiresolution surfaces. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2002), ACM Press, pp. 312–321. 4

[Cha] CHAIKIN G.: An algorithm for high speed curve generation. *IEEE Computer*. 6

[Cur] CURLESS B.: From range scans to 3d models. *IEEE Computer*. 4, 6, 7

[EHBE97] EGGLI L., HSU C., BRUDERLIN B., ELBER G.: Inferring 3d models from freehand sketches and constraints. *Computer-Aided Design 29*, 2 (1997), 101–112. 4

[Far01] FARIN G.: *Curves and srufacs for CAGD, A practical guide 5th edition*. Morgan Kauffmann, 2001. 6

[GBS03] GROSSMAN T., BALAKRISHNAN R., SINGH K.: An interface for creating and manipulating curves using a high degree-of-freedom curve input device. In *Proceedings of the conference on Human factors in computing systems* (2003), ACM Press, pp. 185–192. 3, 4, 7

[IMT99] IGARASHI T., MATSUOKA S., TANAKA H.: Teddy: a sketching interface for 3d freeform design. In *SIGGRAPH '99* (1999), ACM Press/Addison-Wesley Publishing Co., pp. 409–416. 2

[JC01] J.C. CARR R.K. BEATSON J. C. T. M. W. F. B. M.: Reconstruction and representation of 3d objects with radial basis functions. In *Proc. SIGGRAPH '2001* (2001), pp. 67–76. 6

[Liu03] LIU X.: Plasticine surgery: editing digital models using physical materials. *Master Thesis, Computer Science, University of Toronto* (2003). 7

[LKG*03] LLAMAS I., KIM B., GARGUS J., ROSSIGNAC J., SHAW C. D.: Twister: a space-warp operator for the two-handed editing of 3d shapes. *ACM Trans. Graph. 22*, 3 (2003), 663–668. 4

[Low91] LOWE D. G.: Fitting parameterized three-dimensional models to images. *IEEE Transactions on Pattern Analysis and Machine Intelligence 13*, 5 (May 1991), 441–450. 4

[MTH94] MASSIE T. H. S. J. K.: The phantom haptic interface : A device for probing virtual objects. In *Proceedings of ASME'94* (1994). 4, 7

[PP97] P. POULIN K. RATIB M. J.: Sketching shadows and highlights to position lights. In *Computer Graphics International* (1997), pp. 56–63. 4

[PPF98] P. POULIN M. O., FRASSON M.-C.: Interactively modeling with photogrammetry. In *Eurographics Workshop on Rendering '98* (1998). 4

[SCOL*04] SORKINE O., COHEN-OR D., LIPMAN Y., ALEXA M., ROSSL C., SEIDEL H.-P.: Laplacian surface editing. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing* (New York, NY, USA, 2004), ACM Press, pp. 175–184. 4

[She04] SHENG J.: An interface for virtual 3d sculpting via physical proxy. *Master Thesis, Computer Science, University of Toronto* (2004). 3, 4, 7

[Sin99] SINGH K.: Interactive curve design using digital french curves. In *ACM Symposium on Interactive 3D Graphics* (1999), pp. 23–30. 2, 7

[SR00] S. RUSINKIEWICZ M. L.: Qsplat: A multiresolution point rendering system for large meshes. In *SIGGRAPH '00* (2000). 6

[TBSR04] TSANG S., BALAKRISHNAN R., SINGH K., RANJAN A.: A suggestive interface for image guided 3d sketching. In *Proceedings of CHI 2004* (2004), pp. 591–598. 2, 4

[TG02] T. GROSSMAN R. BALAKRISHNAN G. K. G. F. A. K. W. B.: Creating principal 3d curves with digital tape drawing. In *Proceedings of the conference on Human factors in computing systems* (2002), ACM Press, pp. 121–128. 4

[VK96] V. KRISHNAMURTHY M. L.: Fitting smooth surfaces to dense polygon meshes. In *SIGGRAPH '96* (1996), ACM Press/Addison-Wesley Publishing Co., pp. 313–324. 4

[WB00] W. BUXTON G. FITZMAURICE R. B. G. K.: Large displays in automotive design. *IEEE Computer Graphics and Applications* (2000), 68–75. 3

[ZHH96] ZELEZNIK R. C., HERNDON K. P., HUGHES J. F.: Sketch: An interface for sketching 3d scenes. In *Computer Graphics (SIGGRAPH '96 Proceedings)* (1996), pp. 163–170. 2

# Modeling with Multiresolution Subdivision Surfaces

presenter: Denis Zorin

New York University

**Abstract**

*Subdivision surfaces and their multiresolution extensions are a powerful representation for surface modeling and design. In this chapter we survey a variety of subdivision-based modeling methods including multiresolution deformations, boolean operations, cut-and-paste editing of surfaces, defining free-form sharp features and adding topologically complex detail. These notes are based on the articles "A Survey of Subdivision-Based Tools for Surface Modeling" by I. Boier-Martin, D. Zorin and F. Bernardini, and "Interactive modeling of topologically complex geometric detail" by J. Peng, D. Kristjansson and D. Zorin.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling Geometric algorithms Keywords: interactive modeling, subdivision surfaces, multiresolution surfaces, volume textures

## 1. Introduction

Subdivision surfaces and their multiresolution extensions offer several advantages over both irregular meshes and spline patches, two of the most commonly used surface representations today. Subdivision offers a compact way to represent geometry with minimal connectivity information. It generalizes the classical spline patch approach to arbitrary topology, it naturally accommodates multiple levels of detail, and produces meshes with well-shaped elements arranged in almost regular structures, suitable for digital processing. When combined with multiresolution analysis, subdivision offers a powerful modeling tool, allowing for complex editing operations to be applied efficiently at different resolutions.

In recent years, the set of tools available for manipulating subdivision surfaces has been growing steadily. Algorithms for direct evaluation [Sta98, ZK02], editing [BKZ01, BMBZ02, BMZB02, BLZ00], texturing [PB00], and conversion to other popular representations [Pet00] have been devised and hardware support for rendering of subdivision surfaces has been proposed [BAD*01, BKS00, PS96].

We focus on the use of subdivision-based representations for styling and conceptual design. We explore various methods for manipulating subdivision surfaces and, whenever possible, we illustrate the evolution of such methods from related representations. We pay particular attention to interactive tools which are suitable for design as they allow the designer to instantaneously evaluate results. While we are trying to provide an overview of the area and include the most relevant methods, we realize that the volume of published work goes well beyond that covered in these notes which is by no means exhaustive (see also [DL02, Sab02] for additional surveys). Many of the topics presented relate to issues we have addressed in our own work which we hope will provide some insights to those pursueing similar interests. We do not attempt to compare these techniques to tools based entirely on irregular meshes or point-based techniques: each approach has a set of advantages and disadvantages and is preferable for a particular set of problems. Any comparison of stand-alone tools may be misleading as modeling tools usually exist in the context of a larger CAD or computer animation system, and integration with other available tools may be of primary importance when a surface representation is chosen.

## 2. Background

The basic idea of using subdivision to produce smooth curves and later, smooth surfaces, has been around for many years (see [ZSD*00] for a brief incursion into the history

of subdivision). However, it is only recently that powerful design tools based on this representation have emerged. This is partly due to the recent advent of multiresolution techniques that facilitate capturing of non-trivial shapes and partly due to even more recent advances in subdivision theory and methods for direct and efficient evaluation of subdivision surfaces. For the purpose of this survey, we provide a brief review of the basic concepts pertaining to subdivision surfaces. For additional details we refer the reader to [ZSD*00, WW01].

Subdivision defines a smooth surface recursively as the limit of a sequence of meshes (see Figure 1). Each finer mesh is obtained from a coarse mesh by using a set of refinement rules which define a *subdivision scheme*. Many schemes have been proposed in the literature. Examples include Doo-Sabin [DS78], Catmull-Clark [CC78], Loop [Loo87], Butterfly [DLG90, ZSS96], Kobbelt [Kob96a], Midedge [PR97]. Different schemes lead to limit surfaces with different smoothness characteristics. For design purposes, the Catmull-Clark [CC78], Loop [Loo87] schemes are most often employed as they are closely related to splines (a de-facto standard in modeling today) and generate $C^2$-continuous surfaces over arbitrary meshes.

**Figure 1:** *Subdivision defines a smooth surface recursively as the limit of a sequence of meshes.*

Multiresolution subdivision extends the concept of subdivision by allowing *detail vectors* to be introduced at each level. Hence, a finer mesh is computed by adding detail offsets to the subdivided coarse mesh. Given a *semi-regular mesh*, i.e., a mesh with subdivision connectivity, it can be easily converted to a multiresolution surface by defining a smoothing operation to compute a coarse level from a finer level. The details are then computed as differences between levels. This representation was introduced by several authors in different forms [LDW97, PL97, ZSS97]. Figure 2 illustrates the power of multiresolution in capturing complex shapes.

A close connection exists between multiresolutin subdivision and wavelets [SDS96]. In particular, two operations known as *Synthesis* and *Analysis* can be defined to propagate data from coarse to fine and in reverse throughout the subdivision hierarchy, similar to wavelet transforms. *Analysis* computes positions of control points on a coarse level $i-1$ by applying a smoothing filter to points on level $i$. Multiresolution details on level $i$ are computed as differences between the two levels. Conversely, *Synthesis* reconstructs the

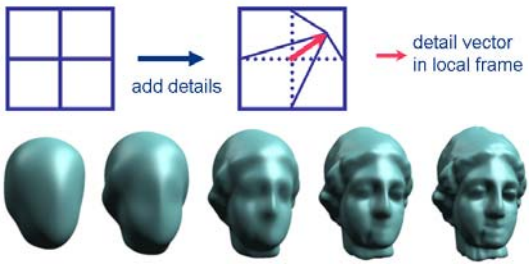



**Figure 2:** *Top: multiresolution subdivision extends the concept of subdivision by introducing detail vectors at each level. Bottom: surfaces obtained by subdivision of the same coarse mesh look very different depending on the amount of detail introduced and the level at which it is introduced. From left to right: no details to progressively more details added on finer levels.*

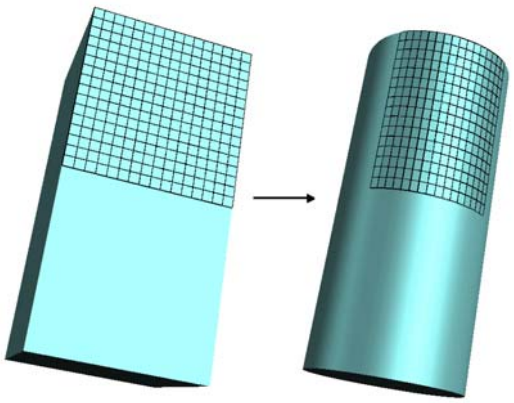


**Figure 3:** *Natural parameterization of a subdivision surface. Each time we apply the subdivision rules to compute the finer control mesh we also apply midpoint subdivision to a copy of the initial control mesh. A mapping from a denser and denser subset of the control polyhedron (left) to the control points of a finer and finer control mesh (right) is obtained through repeated subdivision. In the limit, a map from the control polyhedron to the surface is obtained.*

data on level $i$ by subdividing the control mesh of level $i-1$ and adding the details [ZSS97].

An important property of subdivision surfaces is that they can be naturally interpreted as functions on the domain defined by the base mesh (see Figure 3). This parametric interpretation is useful in many circumstances related to design, from derivation of differential quantities to dealing with constraints along arbitrary curves. Figure 3 illustrates this natural parameterization.

## 3. Free-Form Editing

Free-form manipulation of 3D models is a popular method for modifying existing shapes which attempts to mimic to a certain extent the process of modeling or sculpting a physical object by hand. The applications are numerous, from animated character creation, to virtual restorations, to industrial design.

The sculpting metaphor for geometric modeling has its roots in the parametric surface works of Sabin [Sab71] and Bezier [Béz74] which contain early mentions of surface deformations. Subsequent work has spanned more than three decades and continues to be investigated in the context of modern systems and surface representations (e.g., [Bar84, SP86, Coq90, HKD93, CR94, MJ96] [SF98, Kob96b, ZSS97, PL97, QMV98, Tak98, WW98] [MQ00, TO02, GS01, BMRB04]).

The basic idea of free-form modeling is to introduce a degree of transparency between the designer and the mathematical model of the surface being shaped. Instead of controlling the shape through a set of non-intuitive surface parameters, free-form deformations allow the shape to be controlled through intuitive manipulation of the surface itself or the space surrounding it. The main challenge is to perform the manipulation through a limited set of controls and to define natural deformations of the surface away from the control positions. Different variations of this paradigm have been developed, including axial deformations [Bar84, CST94, LCJ94] which alter the axis of a shape to induce its deformation, lattice deformations [SP86, Coq90, MJ96] which operate on the cells of a space lattice to deform the volume inside the lattice, manipulations on scalar field embeddings [HQ03], control mesh editing methods which shape parameterically-defined surfaces by imposing constraints on their control meshes [ZSS97], and variational methods which operate by optimizing an energy functional over the surface under constraints [Tak98, BMRB04].

We focus our attention on methods that take advantage of subdivision representations and among these, we emphasize those that support interactive multiscale modeling. Subdivision representations are particularly suitable for free-form editing due to their hierarchical nature which easily accommodates multiscale edits, as well as their efficiency in terms of storage and access. For a survey of deformable models based on other representations see [GM97].

### 3.1. Control mesh manipulations

Manipulating control meshes offers a straightforward interface which supports interactive shape deformations. This approach has been extensively employed in spline-based modeling [CRE01] and can be naturally extended to subdivision surfaces. Collections of control mesh vertices, edges, and faces are re-positioned so as to induce modifications of the resulting limit surface. In addition, control points can be added and edges and faces can be split to increase the complexity of the shape as editing progresses. This type of manipulation is very common and can be found at the basis of commercial modeling packages with support for subdivision surfaces. It is routinely used for animated character design (e.g., in Discreet's 3D Studio Max [dsm], in Alias' Maya [may]) and is becoming increasingly popular for industrial modeling (e.g., in Dassault Systèmes' Catia [cat]). Figure 4 illustrates examples of shape modeling through control point manipulation.

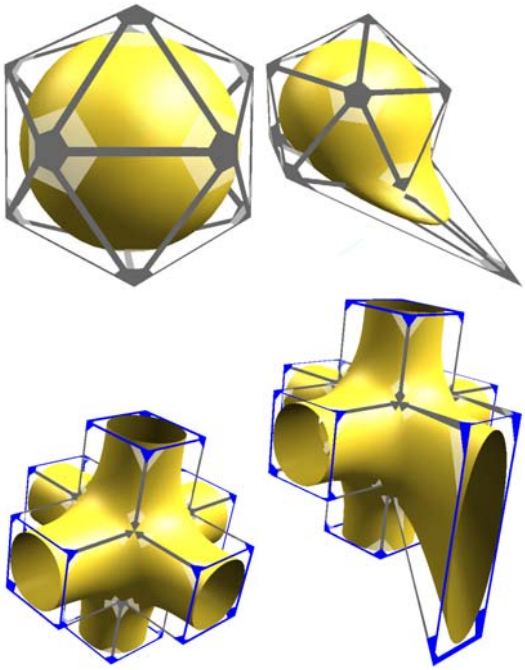


**Figure 4:** *Shape modeling through control point manipulation: Loop subdivision surface (top), Catmull-Clark subdivision surface (bottom).*

Single resolution control mesh manipulations offer only limited flexibility in designing shapes: only coarse shape deformations can be accommodated. Multiresolution subdivision surfaces are a much more powerful representation which lends itself very naturally to multiscale editing. Depending on the level at which the editing occurs, either a global deformation (coarse level) or a local deformation (fine level) is induced. This idea was exploited, for instance, in [ZSS97, PL97] for interactive multiresolution editing of Loop surfaces and in [DKT98] for Catmull-Clark ones. Using a combination of subdivision (i.e., transforming a coarse mesh into a finer one) and smoothing (i.e., transforming a fine mesh into a coarser one), edits performed at different levels of subdivision can be propagated through the hierarchy while keeping the magnitude of multiresolution details

under control. Figure 5 illustrates edits at various scales performed on the Armadillo model.

**Figure 5:** *Multiresolution editing according to [ZSS97]: left – input model; right – editing result. Note the large-scale edit of the belly and the fine-scale edit around the chin.*

Variations of this approach include modeling with *displaced subdivision surfaces* [LMH00] and subdivision surface *fitting* [STKK99, LLS01a, MZ00]. The displaced representation can be viewed as a restricted form of multiresolution subdivision consisting of a control mesh and a single level of scalar details. A domain surface is generated from the control mesh using Loop subdivision [Loo87]. A displacement map computed from the scalar displacement is then applied over the domain to generate the final surface. The displacements can be edited to create fine-level features on the surface, while control mesh edits lead to global shape alterations. In surface fitting a surface is deformed to conform to the shape of another given data set (e.g., points, curves, another surface). This approach is somewhat different than those discussed so far in that it is less suitable for interactive manipulation. Typically some optimization of the surface being fitted is performed in order to determine optimal control point positions which lead to a best fit between the surface and the target. The accuracy of the fit is controlled through a threshold parameter that bounds the error between the target and the fitted surface.

### 3.2. Variational design

Variational surface design operates on the principle of modifying a shape so that its *fairness* is optimized. Surface fairness is typically measured in terms of its energy and the idea is to find a minimum-energy state which, in turn, corresponds to the fairest possible shape. In Computer Graphics, energy-minimizing surfaces became popular in the context of simulating physical properties of materials [Bar84, TF88, WW92]. Celniker and Gossard [CG99] and later Welch and Witkin [WW92] pointed out the relationship between fair surface design and energy minimization.

Most commonly, fairness is expressed as an integral of a physical parameter associated with a real object bearing the shape of the surface [Hal96]. A widely used measure of fairness is the combination of *stretching* and *bending* energies:

$$Energy(S) = \alpha \int ||I||^2 dS + \beta \int ||II||^2 dS \qquad (1)$$

where $I$ and $II$ denote the first and second fundamental forms of the surface and $||\cdot||$ is a suitably chosen matrix norm [TPBF87].

For practical purposes, discretized linear forms of equation (1) using parametric derivatives are typically employed:

$$E_{stretch} \approx \int_\Omega \left(\frac{\partial S}{\partial u}\right)^2 + \left(\frac{\partial S}{\partial v}\right)^2 du dv \qquad (2)$$

$$E_{bend} \approx \int_\Omega \left(\frac{\partial^2 S}{\partial u^2}\right)^2 + 2\left(\frac{\partial^2 S}{\partial u \partial v}\right)^2 + \left(\frac{\partial^2 S}{\partial v^2}\right)^2 du dv \qquad (3)$$

where $\Omega$ denotes the parametric domain of the surface $S$. Most variational approaches take advantage of these expressions, although alternative approaches have been proposed (e.g., [CSA*02]). The main differences are in the types of parameterizations used to derive the differential quantities. For example, Greiner [Gre94] and later Kobbelt [Kob96a] suggested a discrete exponential map for local parameterizations (see Figure 6) such that each vertex $P_0$ has coordinates $(0,0)$ and its 1-ring neighbors $P_i \in R(P_0)$ are assigned coordinates:

$$(u_i, v_i) = e_i \left( \cos\left( \sum_{j \in R(P_0)} \alpha_j \right), \sin\left( \sum_{j \in R(P_0)} \alpha_j \right) \right) \qquad (4)$$

where

$$\alpha_j = \frac{2\pi \angle(P_j^l P_0^l P_{j+1}^l)}{\sum_{j \in R(0)} \angle(P_j^l P_0^l P_{j+1}^l)}. \qquad (5)$$

In the context of subdivision surfaces, Halstead et al. [HKD93] were among the first to describe a method for interpolating a given shape with a Catmull-Clark surface while minimizing surface fairness. Given the lack of a "natural" parameterization near extraordinary points, they re-formulated the stretch and thin-plate energy definitions in terms of the control meshes at different subdivision levels (rather than the limit surface). In their method subdivision is used to isolate extraordinary vertices and bi-cubic spline evaluation is used to evaluate the fairness norm away from such vertices.

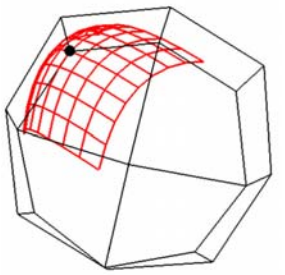


**Figure 6:** *Local quadratic interpolant used to approximate first and second order derivatives [BMRB04].*

Kobbelt [Kob96b] introduced the concept of *variational subdivision* to create interpolatory subdivision rules that place newly inserted vertices so as to minimize a global energy functional. Using a similar idea, Weimer and Warren [WW98] propose two schemes for variational subdivision of thin-plate splines. One scheme provides an exact solution to the variational problem, but the subdivision matrix has to be recomputed at every subdivision level. The other scheme is only approximate, but has the advantage that rules can be precomputed. Both schemes are restricted to rectilinear grids. Another method which connects subdivision with fairing and cascading multigrid methods was proposed in [DMR02]. The basic idea in this case is to interpret the evolution of the surface under curvature motion as a filtering proces.

Later on, Friedel et al. [FMS03] proposed using the characteristic map parametrization to construct first order data-dependent energies. This leads to a nonlinear minimization problem which is solved by re-writing the surface energy as a linear combination of precomputed stiffness matrices.

*Constraints* play an important role in variational design methods. In their absence, the optimization problem has a trivial solution, which usually leads to the collapse of the surface to a single point (an exception is the method of Boier-Martin et al. [BMRB04] in which the trivial solution corresponds to the input surface). We distinguish between two classes of constraints [WW92]:

- *Finite-dimensional*: involve point and normal constraints at discrete locations on the surface. These are the most commonly used. Point constraints are used to enforce spatial interpolation conditions. For subdivision surfaces such constraints typically correspond to control points and are easy to implement by solving linear systems. Normal constraints are used to enforce surface normals at certain points on a surface. Different approaches can be used to constrain normals: expressing the fact that two tangent vectors must be perpendicular to the prescribed normal, enforcing the positions of the vertices of a given face so that the face normal coincides with the prescribed one, or constraining tangent vectors rather than normals (the last two tend to over-constrain the problem).
- *Transfinite*: involve one or two-dimensional surface entities such as embedded curves and patches. Curve constraints are among the most common in this category. Enforcing such constraints involves solving an integral over the entity. For example, to constrain a surface curve $C(t) = S(u(t), v(t))$ along a given space curve $C_0(t)$, the following must be satisfied:

$$\int (C - C_0)^2 = 0 \tag{6}$$

  Such constraints are usually discretized and enforced either by using a least-squares approach [WW92] or by reparameterizing the surface to align control points or edges with constraints [BMRB04] (see also Algorithm 1 in section 4). An alternative approach is to evaluate the curves and to incorporate the result of the evaluation into the subdivision rules to produce a limit surface that interpolates the curves. This is the object of *combined subdivision schemes* [Lev99] (see also [Nas00, NA02, SWZ04]).

Figure 7 illustrates the result of modeling with various types of constraints.

Another important consideration in dealing with constraints is the *region of influence* of a constraint. It is defined as the portion of the surface affected by the constraint. The region of influence can be explicitly enforced [Kob00] by letting the designer encircle an area on the surface. This generates boundary constraints between the surface inside the area of influence and the rest of the surface. Alternatively, in the case of hierarchical representations such as subdivision hierarchies, the region of influence can be controlled indirectly through the levels at which constraints are defined. For example, Takahashi et al. [Tak98] impose constraints at various scales using a wavelet framework. Constraints are being propagated from finer to coarser scales, however, the region of influence of each constraint is not controlled in any way. In [BMRB04] the influence of a constraint is explicitly enforced by the coarse level at which the constraint is propagated. Thus, more global or local edits can be performed depending on the level to which the constraint is restricted: a coarser level will induce a more global deformation, whereas a finer level will produce a more local edit (see Figure 8).

A related issue is that of detail preservation. When a global shape change occurs, it is often expected that the high frequency details are preserved over the modified surface. The face of Venus in Figure 9 is represented as a multiresolution subdivision surface in which non-trivial detail vectors capture the organic shape of the model. If a shape deformation is performed by pulling on a single point at the tip of the nose, a naive energy optimization approach leads to a fair shape that satisfies the constraints, but all the details of the face are lost (note that boundary constraints must also be imposed in this case to avoid the collapse of the surface to a

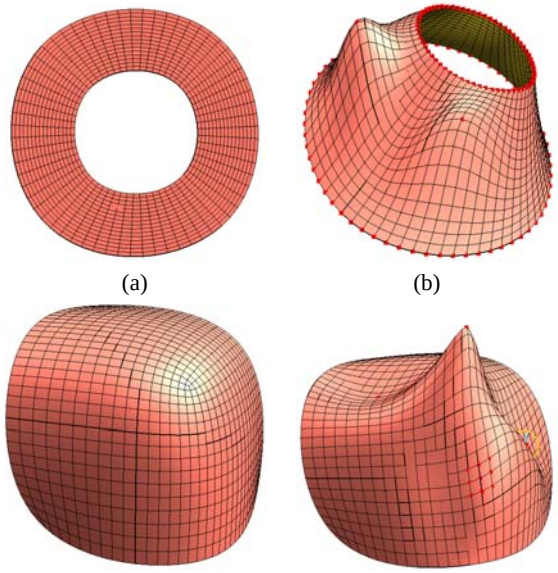



**Figure 7:** *Constraint types: (a)-(b) point and discretized curve constraints; (c)-(d) normal constraints.*

single point). One solution is to separate the high-frequency information before optimization and to "re-apply" it to the new shape [Kob00]. This introduces an overhead related to saving and restoring surface details. To avoid this overhead, Boier-Martin et al. [BMRB04] propose to define a vector field of deformations over the surface and to optimize the energy of this vector field rather than the energy of the surface itself. Initially all deformation vectors are null. When an edit occurs, the corresponding deformation vector (i.e., at the tip of the nose) becomes non-null. The optimization procedure tries to smooth the deformation field under the constraints defined by the non-null vectors. Since the deformations are defined with respect to the detailed shape, the details are preserved during deformation. Note that, in this case, boundary constraints are not necessary as the rest shape in the absence of constraints is the input shape.

An added advantage of subdivision hierarchies is that they facilitate the use of multigrid methods [Bri87] to solve the constrained minimization problem. In the presence of many constraints, however, even multigrid solvers may be too slow to yield results at interactive rates. A possible solution [BMRB04] is to aim for an approximate solution during interaction and a more accurate (non-interactive) result after the interaction has stopped. Figure 10 illustrates the differences between a Catmull-Clark approximation obtained at interactive rates and a more accurate multigrid minimization.

For completeness, we mention the fact that the evolution of energy over time has also been considered to derive *dynamic surface models* [TQ94, QT96]. Dynamic models

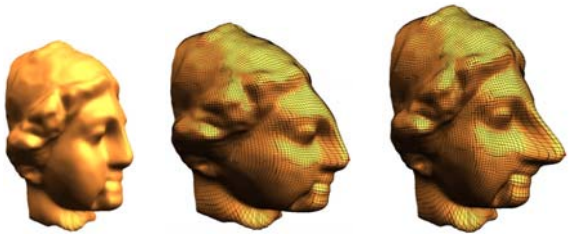


**Figure 8:** *Region of influence of a multiresolution constraint: left – input model; middle – constraint is propagated to the coarsest subdivision level, inducing a global deformation of the head by pulling a single point on the nose; right – constraint is propagated only two levels coarser inducing a more localized edit.*

based on subdivision surfaces have been proposed by Qin et al. [QMV98]. Such models are typically too complex to support interactive design operations.

*Topology modifications.* The free-form modeling methods discussed so far operate by deforming the input surface without changing its topology. Some applications, however, may require topological modifications, such as creating handles and tunnels. An interactive sculpting environment which supports this type of edits was proposed in [GOP99]. The *Localized hierarchy Surface Splines* allow adding handles and punching holes, while maintaining $C^1$ continuity across the surface which is represented explicitly in piece-

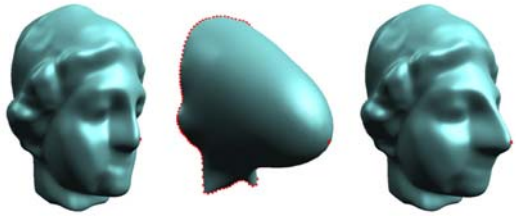


**Figure 9:** *Energy optimization with constraints: left – input multiresolution subdivision surface with details; middle – optimization without detail preservation; right – optimization with detail preservation.*

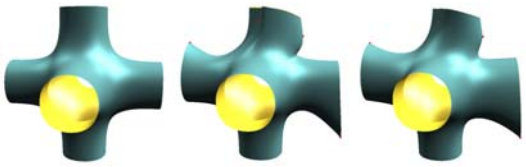


**Figure 10:** *Computing a solution to the energy minimization problem with different accuracies: left - input model; middle - Catmull-Clark solution obtained interactively; right - multigrid solution.*

wise polynomial or spline form. The main idea behind localized hierarchies is to allow local edits on locally refined mesh fragments based solely on coarser level data. Direct manipulation is performed by interacting directly with the surface rather than with control mesh. The types of operations supported include fillets, blends, semi-sharp features, extrusions, holes, and bridges.

Using meshes as an underlying representation, Guskov et al. [GKSS02] propose a user-driven procedure for inducing topological modifications in a semi-regular setting. The so-called *hybrid meshes* are multiresolution surface representations which enhance subdivision-based refinement operations with irregular operations that support changes in topology and approximate detailed features at multiple scales. In [GKSS02], hybrid meshes are defined as quadrilateral meshes on which regular $1-4$ face splits are combined with irregular operations through which groups of quads are removed and/or replaced.

## 4. Boolean Operations

Boolean operations provide a straightforward approach to creating complex models from simpler ones using intuitive combinations. Addition, subtraction, and intersection can be packaged into editing tools for modeling solids bounded by subdivision surfaces.

### 4.1. Mesh-Based Approximations

Traditionally, Boolean operations on boundary representations (B-reps) of solids have required intersecting parametric surfaces, removing the unwanted parts, and building new surfaces from the remaining ones. This approach presents a number of challenges, as intersections are difficult to perform for high-order B-reps and often lead to increasingly complex intersection curves. Exact matching of surfaces bordering such curves is also problematic, as it is not easy to ensure that curves in different parametric domains coincide in 3D. Consequently, subsequent editing of the resulting models may lead to unwanted artifacts in the surface (e.g., cracks) which require special handling.

A substantially simpler approach, proposed by Linsen [Lin00] is to use the control meshes corresponding to the parametric parts being combined, rather than the surfaces themselves. This implies that the intersections between solids are only approximately computed. At the same time, the problem of intersecting arbitrary surfaces translates into the much simpler one of intersecting arbitrary meshes. The meshes are first triangulated to avoid difficulties posed by handling of non-planar faces. Two approaches to building a combined control mesh are discussed: clipping triangles along the intersection boundaries and connecting intersection points and removing faces along the intersection curves and remeshing the resulting gaps. The latter has the advantage that it produces a more visually pleasing result. The main drawbacks in both approaches lie in the inefficiency of computing triangle-mesh intersections and robustness issues associated with such computations as well as gap filling for arbitrary gap topologies (see also [LFKN03] for variations on the topic of computing intersection curves for subdivision surfaces).

Using a similar control-mesh based approach, Biermann et al. [BKZ01] propose an approximate scheme for computing Boolean operations which deals with several important issues: matching the topology and the geometry of the intersection curve, fitting the resulting surface to the original data, and accurately capturing and representing sharp features in the result. The method uses piecewise-smooth multiresolution Loop [Loo87] subdivision surfaces to represent surfaces being combined. The algorithm assumes that each part being used in a Boolean operation is bounded by a closed orientable surface. It follows several steps:

1. Compute intersection curves.
2. Build resulting control mesh and compute an initial parameterization of the resulting surface over this mesh.
3. Optimize the parameterization from the previous step.
4. Use multiresolution fitting to approximate the input data as closely as possible.

For the first step, the authors improve on both the efficiency and the robustness of the naive mesh-mesh intersection approach by using bounding box hierarchies to accel-

erate computations and a perturbation scheme [Sei98] to increase robustness.

After determining the topology of the intersection, control meshes are merged with special consideration for several issues: preserving the topology of the cut, inserting a minimal number of new vertices, and keeping their valence small. The input control meshes are cut along intersection curves and a new control mesh is combined from the remaining pieces. The cutting process takes advantage of the natural parameterization of subdivision surfaces over their control meshes (see section 2) to approximate the intersection curve by alternating so-called *Snapping* and *Refinement* steps:

*Algorithm 1 (snapping and refinement):*

Given a domain mesh $M$ and an intersection curve $c(t)$ in $M$
Repeat
  For each vertex $v$ of a triangle intersected by $c$ do
    1. Find $\alpha \in c$ closest to $v$
    2. Snap $v$ to $\alpha$ if possible
  Adaptively refine parameterization
until (curve adequately approximated)

*Snapping* is performed between points of the curve and parametric mesh vertices, if they are sufficiently close. While optional, this step considerably reduces the complexity of the resulting domain (fewer faces). The role of the *refinement* is to increase the accuracy with which intersection curves are approximated. It is typically performed by midpoint subdivision of triangles which are intersected by curves multiple times or which fully contain curves. Figure 11 illustrates this process. The output of this step consists of piecewise linear approximations of the intersection curves, either along input edges or along newly introduced edges obtained by splitting triangles.

After cutting, the portions of the control meshes not required in the Boolean operation are removed and the meshes are joined along their boundaries. This is also done in two steps: vertices along one boundary are paired to corresponding vertices along the other boundary. When correspondences do not exist, triangles along the boundary are refined so as to introduce new vertices. Paired vertices close to one-another are merged together. During merging, intersection curves are also tagged with sharp feature tags (see also section 5).

By construction, the resulting merged control mesh constitutes a parameterization domain with the property that every one of its vertices belongs to one of the original domains. However, the initial parameterizations of the parts of the input models corresponding to the Boolean operation may not be optimally parameterized over the new domain. An optimization procedure is used to reduce the distortion of the resulting surface over the new domain.

The last step of this method computes optimal positions of control points given the previously computed parameterization. The merged control mesh is subdivided a number of times and the resulting mesh is fitted to the original data in least-squares sense. Results of Boolean operations obtained with this method are shown in Figure 12.

### 4.2. Surface Cut-and-Paste

Surface pasting can be viewed as an instance of a Boolean operation. The basic paradigm implies creating new models by combining pieces of existing models. In its most basic form, a cut-and-paste operation involves selecting and transferring a *feature* of interest from a *source surface* to a *target surface*. There are several fundamental steps involved such an operation:

1. Feature selection
2. Separation of surfaces into base and detail parts
3. Transferring the feature onto the target surface

The idea of pasting surfaces was first introduced in the context of hierarchical splines [BBF94, CMB97]. In this case a tensor-product B-spline surface is designated as the feature to be attached to another surface. Steps (1), (2) are assumed to have been performed in a pre-processing stage and (3) is achieved by representing tensor-product B-splines as Greville displacement B-splines [BBF94] and applying a mapping that takes into account the topology of the target surface and the Greville displacement representation of the feature [BBF94]. The main restriction is that there are no smoothness guarantees at the boundary between the feature and the target surface (not even $C^0$ continuity). One solution is to refine the feature surface so that its boundary better approximates the target. However, this amounts to introducing unnecessary control points over the entire feature (rather than only along boundaries), making subsequent processing of the feature very inefficient. An alternative solution was proposed by [CM00] and makes use of quasi-interpolation [dF73] to improve the result of pasting. In this case, interior feature control points are pasted using Greville displacements, while boundary points are pasted using quasi-interpolation. This leads to a composite surface which still exhibits discontinuities along the pasting boundary, however, less severe than in the original approach. In addition to the lack of continuity, the types of features that can be pasted are also limited by the underlying surface representation. Performance is also an issue due to expensive evaluations. An interactive spline-based interface was developed in [Ma00]. Due to performance limitations, the feature is not positioned directly onto the target surface, but rather is floating in its vicinity and the user is presented with a rough outline of the contour of the feature on the target. Once a position is decided upon, the actual pasting occurs.

Biermann et al. [BMBZ02] describe a more general procedure for cutting and pasting portions of existing surfaces using an intuitive approach, similar to those commonly used

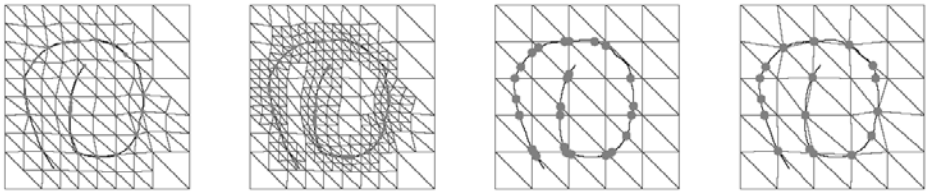


**Figure 11:** *Refinement and snapping: two steps of refinement are shown on the left. The image of the curve in parameter space and vertex snapping are shown on the right (see [BKZ01]).*

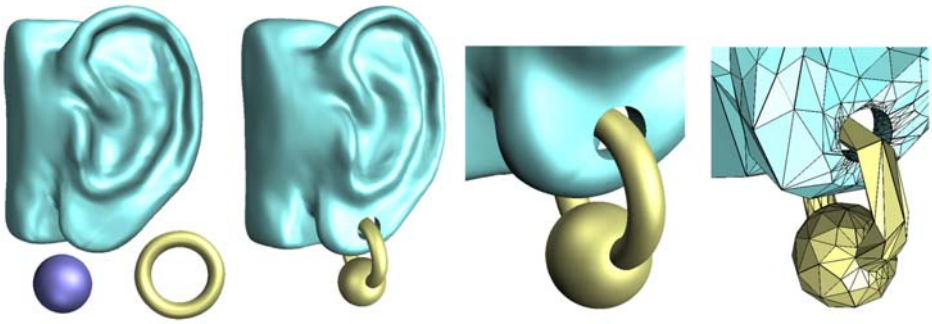


**Figure 12:** *Boolean operations on multireolution subdivision surfaces [BKZ01].*

for 2D image cut-and-paste. The user initiates a cutting operation by selecting a feature of interest on an existing surface (termed the *source* surface (see Figure 13 (a)). She also specifies a position on a *target* surface where the source feature is to be pasted (see Figure 13 (b)). The actual pasting is performed in a sequence of steps (Figures 13(c)-(g)) which take advantage of the underlying semi-regular representation to achieve interactive rates. A discussion of the main steps follows.

**Feature selection** is performed interactively by the user who selects a region of interest on the source surface. A free-form closed space curve is used to outline the selection. The portion of the surface inside the curve constitutes the feature(s) of interest. The curve can have an arbitrary shape and does not have to be aligned with underlying mesh edges. The portion of the surface inside the curve must have disk topology, but it does not have to be a height field (see Figure 14).

**Base / detail separation** must be performed on both the source and target surfaces to define what constitutes feature detail as opposed to the larger-scale surface shape that should be ignored. Since this is largely dependent on the semantics of the operation, it is best left to the user. In [BMBZ02], the authors propose a continuum of base surface choices controlled by a *flatness* parameter. The base surfaces are obtained by smoothing the original surface to various degrees or by simple energy minimization within the feature boundary. Figure 15 illustrates the different effects

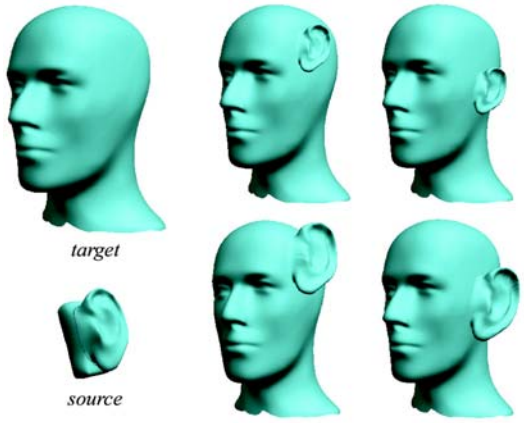



**Figure 14:** *Pasting and interactively placing a complex feature: a digitized model of a clay ear constitutes the feature to be pasted onto the mannequin head. The ear can be interactively scaled, rotated, and translated on the surface of the head.*

obtained using different base surfaces for the source and target models.

**Feature transfer** is a complex process as it generally involves finding a mapping between two arbitrary surfaces. The solution proposed in [BMBZ02] is to build the map-

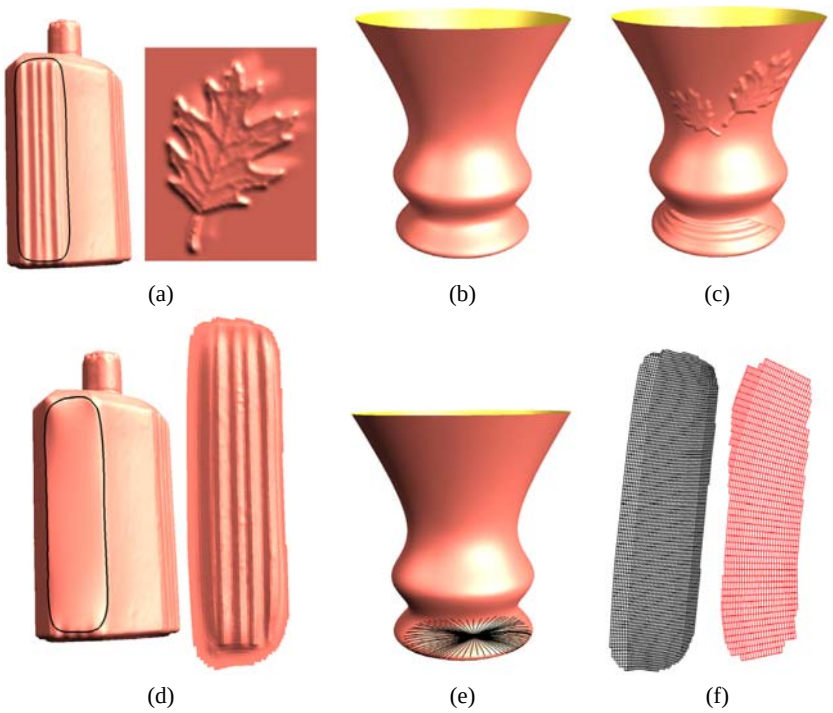



**Figure 13:** *Feature-based design of an ornate vase: (a) input (source) surfaces; (b) target surface; (c) result after multiple pasting operations; Steps of a cut-and-paste sequence according to [BMBZ02] shown for the bottle pattern: (d) source feature selection; (e) finding a target region around a user-selected position on the target surface; (f) parameterization of source (left) and target (right) regions onto a common plane (shown displaced for illustration).*

ping as a composition of maps to an auxiliary plane. The advantage of this approach is that it no longer requires parameterizing an arbitrary (source) surface onto another arbitrary (target) surface. Instead, flattening methods which have been much more extensively researched [SdS01, DMA02] are needed. For the source surface the flattening is relatively easy to perform as the feature is already selected and homeomorphic to a disk. For the target surface the problem is more complicated as the surface can have any shape and can be quite large. In order to avoid flattening the entire target surface, the authors propose a method for approximating the portion of the target surface that is actually involved in pasting. As the user specifies a target location where the feature is to be pasted, the goal is to find a region that resembles the source feature in shape and size. To identify such a region, a generalized radial parameterization of the feature boundary is used [BMBZ02] (see also Figure 16). Once such a region is found, the transfer of the source feature onto the target model is done by aligning the planar parameterizations of the source and target regions followed by resampling the source feature onto the target connectivity.

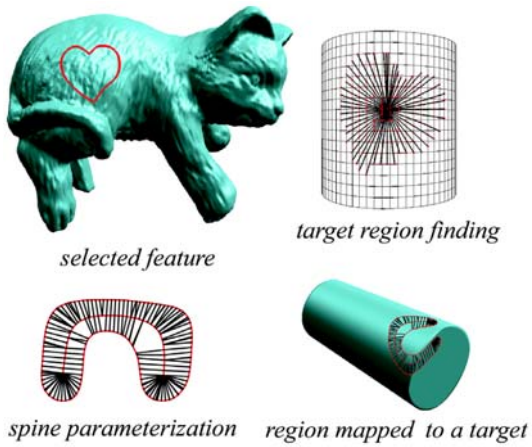



**Figure 16:** *Finding a target region through radial parameterization of the feature outline.*

The pasting method of Biermann et al. [BMBZ02] provides a robust and efficient pipeline for interactive surface pasting. Its main constraints are related to self-intersections that may appear when features are pasted onto highly curved

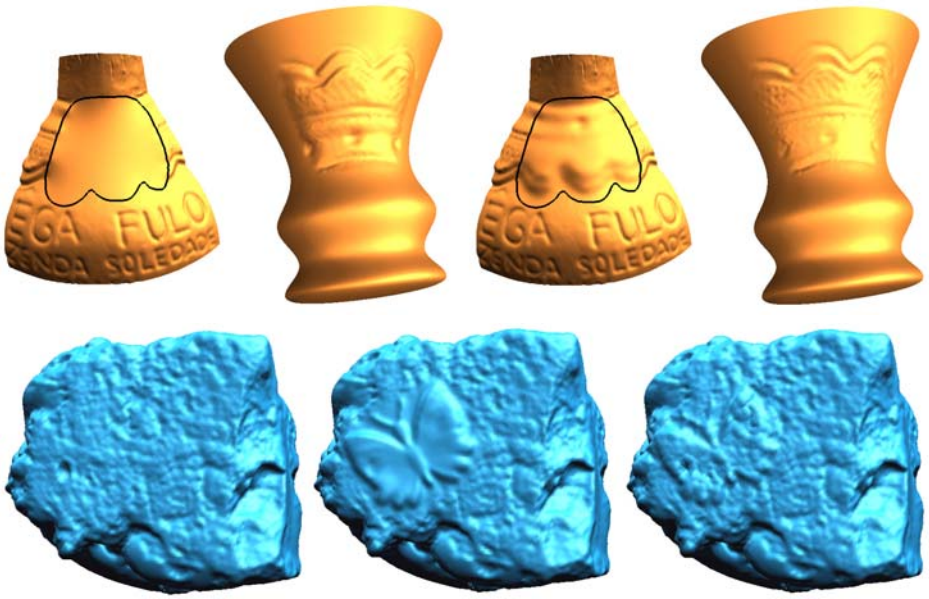


**Figure 15:** *The effects of changing the base surface on the result of pasting: (top) digitized bottle detail appears on the vase differently, depending on the choice of base surface; (bottom) the butterfly feature is pasted on a rock model with and without preservation of target detail.*

surfaces and to topological constraints on the features that can be pasted (i.e., only features with disk topology are handled). The former can be solved using a hierarchical pasting approach, in which the feature to be pasted is decomposed into frequency bands and the pasting is performed progressively, by pasting low-frequency details first, and high-frequency ones on top. The latter problem is more complex and requires more careful handling. A possible solution, albeit outside the subdivision framework, has been recently proposed in [FMMY03]. In this case, a volumetric approach is used to parameterize the feature and B-spline fitting is used to separate base from details. The advantage lies in the generality of features that are handled, including higher genus ones and the ability to paste them on highly curved areas. The main drawbacks are related to B-spline fitting and the need to introduce a large number of points in order to obtain a good fit. The result is not a seamless representation, but rather a composite one consisting of the original and the pasted part. In addition, the feature cannot be interactively dragged on the target surface.

### 4.3. Surface Trimming

Trimming, i.e., cutting holes in the surface of an object along specified curves, can also be considered as an instance of a Boolean operation. Since this type of operation requires special subdivision rules along the trim boundary, we classify it as a special case of a non-smooth feature and we discuss it in section 5.

## 5. Non-Smooth Features

Subdivision surfaces can be naturally used to model smooth surfaces of arbitrary topological type. Many real objects, however, exhibit non-smooth features, such as sharp edges and boundaries, corners, and darts. While multiresolution detail vectors may be used to approximate sharp features (see Figure 17), a different setting is required to represent such features exactly. It entails altering the subdivision rules to produce limit surfaces that are only piecewise smooth, i.e., consist of smooth patches joined together along possibly sharp boundaries. To represent piecewise smooth surfaces, control mesh edges and vertices are typically *tagged* for special handling (see Figure 18). Special subdivision rules are employed in the vicinity of tagged mesh elements so as to avoid smoothing them. An edge can be tagged as a *crease edge* and vertices incident to crease edges may be tagged as one of the following (see Figure 18):

- *crease vertex*: exactly two crease edges join smoothly at this vertex
- *corner vertex*: two or more crease edges join non-smoothly at this vertex
- *dart vertex*: exactly one crease edge is adjacent to this vertex

Early mention of special rules for surfaces with boundaries appeared in the work of Doo [Doo78] and Nasri [Nas91], however accompanied only by partial analyses of the resulting surfaces. The first rules leading to prov-

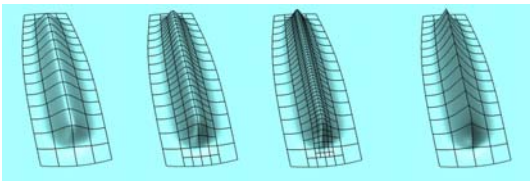



**Figure 17:** *Multiresolution details are required to approximate sharp features using smooth surface representations (left three images). A piecewise smooth surface representation (right) allows sharp features to be modeled with detail vectors.*

ably $C^1$-continuous surfaces were defined in [HDD*94] as a generalization of the Loop subdivision rules [Loo87]. The analysis of the resulting surfaces can be found in [Sch96]. As pointed out in [BLZ00], the rules introduced in [HDD*94] have two main drawbacks: they are not suitable for modeling concave corners and the shape of the generated surface boundaries depends on the number of interior control points adjacent to each boundary point. The latter leads to undesirable gaps between surfaces joined along such a boundary. Both problems were handled by Biermann et al. [BLZ00] for the Loop [Loo87] and Catmull-Clark [CC78] subdivision schemes.

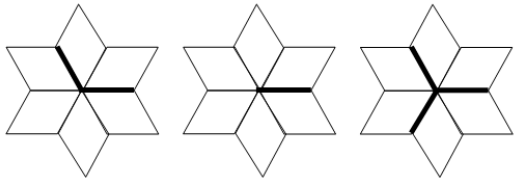


**Figure 18:** *Mesh tags corresponding to (from left to right): crease, dart, and corner sharp features.*

*Modifications* are sometimes applied to subdivision rules to achieve different effects. For example, deRose et al. [DKT98] propose an *edge sharpness* parameter $s$ to vary sharpness along an edge and to allow for different degrees of sharpness. The parameter is used to blend between the positions $p^{smooth}$ of a control point obtained with the smooth subdivision rules and a point $p^{sharp}$ obtained with sharp subdivision rules:

$$p^{new} = (1-s)p^{smooth} + sp^{sharp},\ s \in [0,1]$$

Biermann et al. [BLZ00] propose a *flatness* parameter $f$ and a *normal modification*. The flatness parameter control the speed at which control points in a neighborhood converge to the tangent plane. The subdivision rules are modified to blend between control point positions obtained without flatness modification and points in the tangent plane ($p$ denotes the vector of control points in the neighborhood of a point,$a_0, a_1, a_2$ are the limit position and tangents at that point, and $x^i$ denote the right eigenvectors of the subdivision matrix):

$$p^{new} = (1-f)p + f(a_0x^0 + a_1x^1 + a_2x^2),\ f \in [0,1]$$

The normal modification is somewhat similar, in that it interpolates between the control point position obtained without the modification and positions in a prescribed tangent plane (a normal $n$ is prescribed through a pair of tangent vectors computed as $a_i' = (a_i n)n$):

$$p^{new} = p + t((a_1' - a_1)x^1 + (a_2' - a_2)x^2),\ t \in [0,1]$$

Examples of sharp features modeled as proposed by Biermann et al. [BLZ00] are illustrated in Figure 19. A software library for piecewise smooth subdivision based on these rules is freely available from [BZ99].

A generalization of the subdivision concept that accommodates sharp features was developed by Sederberg et al. [SZSS98]. By drawing an analogy between recursive subdivision schemes and knot insertion for B-splines, the authors propose non-uniform versions of the Doo-Sabin [Doo78] and Catmull-Clark [CC78] subdivision schemes (under the general denomination of *non-uniform recursive subdivision surfaces* or NURSS). Each edge in a non-uniform Catmull-Clark control mesh (each control point in a Doo-Sabin mesh) is assigned a *knot spacing*. When all knot spacings are equal, the standard schemes are obtained. Two types of subdivision rules have to be considered for NURSS: the usual refinement scheme for the geometric positions of control points and an additional refinement scheme for knot spacings. Sharp features can be generated by setting certain knot spacings to zero.

The methods described so far require sharp features to be aligned with control mesh edges. Moreover, they provide little control over the profile of the resulting features. To address these limitations, Khodakovsky et al. [KS99] propose a curve-based feature editing approach. Feature curves are defined directly on the model surface through user interaction and can follow arbitrary paths, unconstrained by the connectivity of the underlying mesh. Features are obtained by perturbing the surface in the vicinity of feature curves. The curves can exist on multiple levels of a subdivision hierarchy. At each level, perturbations are computed with respect to local frames, so any coarse level modifications of the surface are carried through to finer levels. Several parameters are used to control the profile of a feature. In particular, sharp features can be obtained by specifying different normal directions for the profile on either side of the curve. This method brings forth a number of significant contributions with respect to previous approaches: it takes advantage of the multiresolution setting to define features through detail vectors at different levels, it does not impose any restrictions on the location of the feature curves on the surface or on their topology (curves can intersect or self intersect), and varying profiles allow both smooth and sharp

features to be represented. The main drawback is that it does not preserve the input representation: after editing, the result is no longer a pure multiresolution subdivision surface, but rather a combined representation, consisting of a surface and a curve. This means that other subdivision-based tools that require as input a pure multireolution representation cannot be directly applied to the result of an editing operation performed with this method.

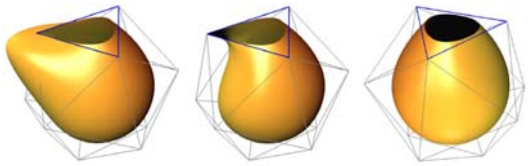


**Figure 19:** *Sharp features generated with the rules proposed in [BLZ00]. From left to right: concave corner, convex corner, and smooth crease.*

This problem is solved in [BMZB02] which uses the reparameterization idea described in section 4.1 to align the parameterization of the surface with the feature curves. Subsequently, sharp subdivision rules can be used along such curves. Figure 20 illustrates this process. An arbitrary feature curve is first projected onto the control mesh at some subdivision level (typically a coarse level which is subsequently refined). A piecewise linear approximation of the curve image in the parametric domain is computed by alternating *Snapping* and *Refinement* steps similar to those of Algorithm 1. The *Snapping* step moves mesh vertices onto the curve if they are sufficiently close, while the *Refinement* step subdivides the parameterization linearly. If $c : [0,1] \to X$ denotes the image of a feature curve in the parameter domain $X$ of the goal is to reparameterize the domain $X$ such that $c$ passes through the vertices of $X$. This reduces to finding a one-to-one mapping $\Pi : X \to X$ which maps vertices of $X$ to curve points: $\Pi(v_i) = c(t_i)$, for some vertices $\{v_0, v_1, \dots\}$ and curve parameters $\{t_0, t_1, \dots\}$. After a finite number of iterations of snapping and refinement, the resulting curve $[v_0, v_1, \dots]$ is guaranteed to have the same topology as $c$ and to follow along mesh edges (and / or diagonals in the case of the Catmull-Clark scheme). After reparameterization, the input surface is resampled according to the new parameterization. Intuitively, this moves the control mesh on the surface and places mesh vertices on the feature curve. Subsequently, the actual feature can be created by tagging the appropriate mesh edges and applying sharp subdivision rules.

In the case of Catmull-Clark meshes, there is an additional complication: the feature curve may pass through mesh diagonals after reparameterization (see Figure 21) and the standard crease rules do not support this situation. Biermann et al. [BMZB02] introduce new subdivision rules to deal with creases along quad diagonals. Sample results obtained with this method are shown in Figures 22. Note that the output

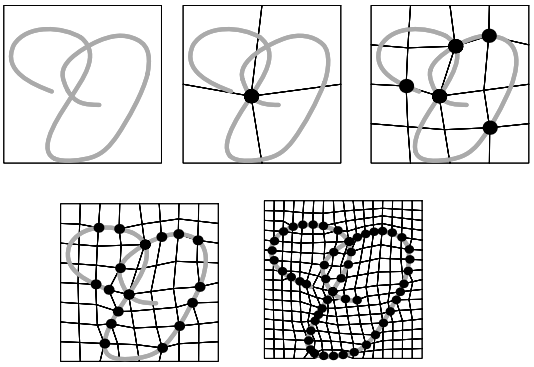


**Figure 20:** *Reparameterization for approximating a feature curve: quads in parameter domain are recursively split and vertices are snapped to the curve. After several subdivision steps the curve is approximated by a sequence of vertices and follows along quad edges or diagonals.*

surface is a multiresolution subdivision surface which can be manipulated with other tools designed to operate on such a representation. In addition, the framework is suitable not only for creating interior sharp features with various profiles (e.g., engravings, embossings), but also to create boundaries, i.e., to trim the input surface along the feature curves. An example of a trimmed surface is shown in Figure 22.

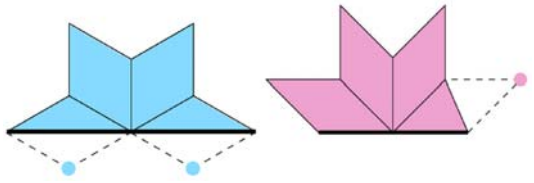


**Figure 21:** *Standard sharp rules do not cover cases when the sharp edge (thick line) passes through a quad diagonal. New rules are necessary for such cases. Dotted lines and circles indicate vertices obtained by reflection used to define subdivision rules for such cases (see [BMZB02] for details).*

For completeness, we also mention the trimming method proposed by Litke et al. [LLS01b] which is complementary to that of Biermann et al. [BMZB02]. In this case quasi-interpolation is used to approximate a trimmed surface with a combined subdivision surface [Lev99].

## 6. Adding topologically complex detail

### 6.1. Overview

Common surface representations, subdivision-based representations in particular, work well for objects of relatively simple topology and continuous geometric structure. However, for many types of objects, the local geometry can be

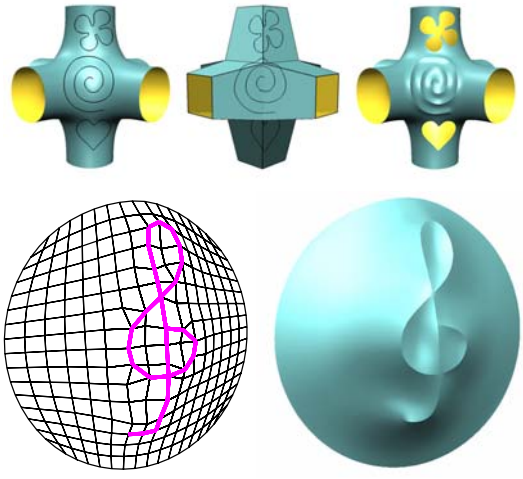


**Figure 22:** *Surfaces obtained after trimming and embossing with sharp features using the method described in [BMZB02]. Top: input curves are shown on the surface (left) and projected into parameter space (middle). The surface obtained is shown on the right. Bottom: a self-intersecting feature.*

**Figure 23:** *A surface with fine-scale detail added as volume texture.*

highly complex. Examples include fur, bark, cracked surfaces, grilles, peeling paint, chain-link fences and others. In these cases, using meshes or patches to represent small-scale geometry is often prohibitively expensive. But if we ignore the small-scale structure, a complex surface often has a simple overall shape, well represented by a mesh or a smooth surface.

In this section we describe a combined volume-surface representation for handling geometry of this type, extending the idea of volume textures. Volume textures aligned with the surface make it possible to represent geometrically and topologically complex details in implicit form, encoding the surface as an isosurface in a layer. This idea was explored by a number of researchers in the past as discussed below.

This representation has important advantages:

- It uses simple and efficient data structures (textures) to represent highly irregular geometry.
- Small features of high topological complexity can be easily introduced and modified.
- Image processing techniques can be used to modify small-scale geometry without topological constraints.
- Hierarchical representations can be naturally constructed using filtering on volumetric textures.
- One can easily use procedural modeling and simulation to produce complex effects near the surface.

Two algorithms central to the goal of using this approach in modeling applications. In addition to surface parametrization required by 2D texturing, volume textures require parameterizing a region of space near a surface. Most of the previous work on volume textures used techniques such as normal displacement, which results in self-intersections near concave features. We describe an algorithm for computing volume layer parametrizations with a number of desirable properties, which can be used to update the parametrization interactively; this is the central geometric algorithm of this representation.

While isosurfaces are convenient for many types of operations, they are much more difficult to render than conventional meshes. We describe algorithms for volume texture rendering that enable interactive manipulation of volume-textured objects. Our algorithm for rendering volume-textured surfaces extends the approach of direct slice-based isosurface rendering for volumes. We take advantage of the programmable graphics hardware to reduce the geometry requirements of the slice-based methods, which is crucial for interactive rendering of volume textures. The description of this rendering algorithm can be found in [PKZ04].

### 6.2. Related work

Our work builds on research in several areas.

**Volume textures.** The idea of volume textures goes back to the work by Kajiya and Kay [KK89]. Our work was motivated by the work of F. Neyret and co-workers (e.g. [Ney95, Ney98, MN98]) as well as recent work on fur rendering [Len00, LPFH01].

Our geometry is to some extent similar to the slab representation used for modeling weathered stone ([DEL*99])

and for volume sculpting in [Aga99]. ([DEL*99]) uses the fast marching method (e.g. [Set99]) to construct layers around a surface. Envelope construction [CVM*96] provides another alternative. Our method is compared with both in Section 6.4.

**Stable medial axes.** Our construction is closely related to the work in vision and medical imaging on using various types of medial axis approximations to analyze shape and extract surfaces from volume data (e.g. [PBC*94, EGM*94]). In these papers a form of the medial axis of an object implicitly defined by a density function is first constructed without recovering the boundary of the object. Our generalized distance function (Section 6.4) is similar to some of the medialness functions used to construct stable medial axes. Our work is closest to [SBTZ99] which solves the Hamilton-Jacobi equations for the medialness function on a regular grid to recover a skeleton. [YP02] uses a Laplacian equation solved on a regular grid to compute correspondences between nested surfaces. Our generalized distance function has the property of pruning away insignificant medial axis branches close to the surface (see Section 6.4). R-functions have been used to achieve similar effects ([Ric73, Roc87, PASS95]), but with more complex computation based on a CSG representation of the surface.

**Implicit surfaces and volume modeling.** There is an extensive body of literature related to volume-based representations (see [Blo97] for a list of references); some recent important work includes [FPRJ00, CBC*01]. Interactive and procedural volume sculpting techniques [WK95, Aga99, CDM*02] can be applied to our surface representation. Most work on volume modeling focuses on volume data in pure form, i.e. objects are represented as level sets of a function defined by volume samples. We concentrate on an approach which blends parametric and surface representations.

**Structured mesh generation.** Constructing a collection of layers aligned with a surface is a common problem in structured mesh generation. Mesh generation is a large and complex field aiming to build meshes suitable for a variety of numerical algorithms for solving PDEs (see, e.g. surveys [Hen96, BP00] and the book [SK93]). Such meshes often have to satisfy stringent requirements for the algorithms to achieve optimal or nearly-optimal convergence rates, especially for CFD problems, for which object-aligned grids are particularly important [Hen96].

Our goal is more modest: we aim to construct a shell aligned with the surface efficiently, maintaining nondegeneracy without explicitly minimizing a distortion measure. At the same time, the criteria used to formulate the PDEs in hyperbolic mesh generation methods (volume preservation and orthogonality), are not necessarily the best for our applications.

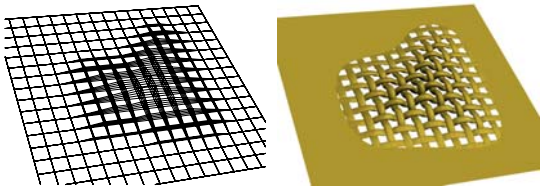


**Figure 24:** *Surface with a volume layer attached.*

### 6.3. Representation

We refer to the initial surface for which we construct a shell as the *base surface*. We consider shells which are obtained by displacing points of the base surface along line segments defined at vertices, which we call *directors*. At each vertex of the surface, we store shell thickness, the number of shell layers stored and texture coordinates. The shell consists of slabs corresponding to the faces of the mesh or individual patches. Each slab is a deformation of a prism.

Shells can be *exterior* (e.g. for fur modeling), *interior* (e.g. for cracks) and *envelope* with layers located on both sides of the surface. Our technique works for all shell types.

The main additional storage is the 3D textures associated with the surface. The number of layers in the shell corresponds to the number of pixels in the texture in the direction perpendicular to the surface. The alpha channel of the texture defines the effective surface implicitly as the isosurface corresponding to a fixed alpha value. The remaining texture channels are used to store the gradient of $\alpha$. The number of layers can vary across the surface. In an extreme case, as shown in Figure 24 the number of layers can go down to one. If there are no features on a portion of the surface we do not need textures for that region.

In our implementation we use multiresolution surfaces with subdivision connectivity, which make it easy to parametrize slices of 3D textures and construct consistent hierarchies for surface and implicit volumentric geometry. However, the basic techniques that we have developed can be applied to arbitrary meshes with 2D texture coordinates.

### 6.4. Constructing shells

In this section we describe our basic algorithm for constructing shells around surfaces. Intuitively, one can think about this process as growing thick skin on the surface; shells constructed by our method behave more or less like elastic compressible skin, which was our goal.

To make a shell useful for volumetric texturing, a number of properties are desirable:

- The layers should not intersect. This requirement is motivated by the "skin" metaphor which we believe to be natural for manipulating this type of surface representation in many cases.

- The layers should have the same connectivity. This is crucial for defining a vertex's volumetric texture coordinates $(s,t,r)$. They can be obtained as follows in this case: $(s,t)$ are given by the base surface parametrization which is assumed to be known, and $r$ is incremented proportionally along the displacement director from the base surface.
- The shell should maintain prescribed thickness whenever possible. However, if thickness cannot be maintained due to geometric obstacles, a valid shell with locally decreased thickness should be produced. This corresponds to the intuitive idea of elastic "sponge-like" skin; note that volume preservation is somewhat undesirable as it is likely to result in fold formation.
- The shell should be close to the one obtained by normal displacement whenever possible.
- The shell at a point should depend only on the parts of the surface close to that point. This property is important for modeling applications and for efficient implementation.

Next, we describe our shell construction algorithm motivated by these requirements.

### 6.5. The basic algorithm

To describe our algorithm in detail, we need some formal notation. We assume that our base surface is a mesh or a higher order surface associated with a mesh (subdivision surface, spline surface etc.) without self-intersections. Formally, our goal of constructing a shell around the surface can be described as follows: given a surface $M$ in $\mathbf{R}^3$, construct a one-to-one map $f(\mathbf{x},t)$ from the direct product $M \times [0,1]$ into $\mathbf{R}^3$. We focus on shells for which $f(\mathbf{x},t)$ is linear, i.e. at each point, $f(\mathbf{x},\cdot)$ is entirely defined by the direction of displacement and shell thickness.

**Main ideas.** Our algorithm is based on a simple idea: to construct a shell, we always need to move away from the surface. In the places where this is impossible (the simplest example is the center of a sphere) the shell cannot be extended further.

To understand how this idea can be made more formal, we consider the example shown in Figure 25 in more detail. Suppose we are building an *interior* shell, offsetting a surface $M$ (in this case, a box) in the direction opposite to the outside normal. If the gradient of the point-to-surface distance function $d(\mathbf{x},M)$ is defined, "moving away" from the surface more formally can be characterized as moving along the gradient of the distance function. This gradient points exactly along a normal direction to the surface whenever it is defined. In such cases we can propagate the shell away from the surface simply moving along the normal. However, the distance function is singular at some points of space which are called (*medial axis points*). Unfortunately the medial axis comes close to the surface at concavities and extends all the way to the object at sharp features, as shown in Figure 25. However, even on the medial axis it is often possible to move

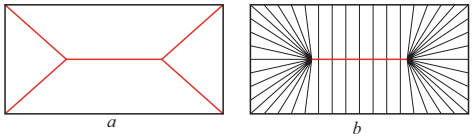



**Figure 25:** *a. Medial axis of a box. b. The shell with target thickness exceeding one half of the box size constructed using the gradient along the medial axis. The shell director lines are shown.*

away from the surface. E.g., if we start from the corner of the box, we just move along the branch of the medial axis. While the complete gradient of the distance function is not defined, it is defined along the medial axis, i.e. the derivatives can be computed for any direction tangent to the medial axis. Define the *extended distance function gradient* by setting the value of the gradient at the medial axis to the gradient along the medial axis, whenever it is defined. The magnitude of this gradient is not necessarily one: the sharper the angle of the concavity, the smaller it is. For the horizontal part of the medial axis of the box, it is identically zero. We note that these are exactly the points where no further motion is possible, because shell parts extended from two sides of the box run into each other. This shows that the magnitude of the gradient of the distance function along the medial axis can be used as a measure of how easy it is to move a particle located at that point of space away from the surface.

These observations suggest the following simple abstract algorithm for constructing the director of a shell: *to obtain the director of a shell of thickness $h$ at point $\mathbf{x}$, first follow the extended gradient field $g(\mathbf{x}) = \nabla_{\mathbf{x}} d(\mathbf{x},M)$ of the distance function, solving the ODE*

$$\frac{\partial F(\mathbf{x},t)}{\partial t} = hg(\mathbf{x}) \tag{7}$$

*where $h$ is the desired thickness, and $F(\mathbf{x},t)$ is position along the integral line of the gradient field passing through $\mathbf{x}$. Then define $f(\mathbf{x},t)$ by linear interpolation between $\mathbf{x}$ and $F(\mathbf{x},1)$.* Note that as long as the integral curve $F(\mathbf{x},t)$ does not reach the medial axis, it remains a straight line with unit speed parametrization, as $\|g(\mathbf{x})\| = 1$. For our box example and sufficiently large $h$ this yields a shell completely filling the box (Figure 25b). Unfortunately, it is difficult to solve Equation 7, as the field is discontinuous, and we would have to compute the medial axis and the gradient along it. To make the algorithm practical, we replace the distance function with a function we call *$L_p$-averaged distance function*.

**Averaged distance functions.** The basis of our definition is the following simple observation. We can rewrite the dis-

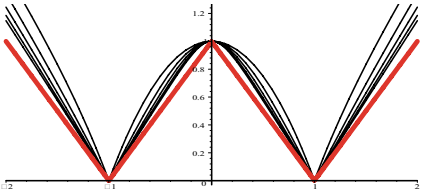


**Figure 26:** *The red plot shows the standard distance function from a point on the line to the set of two points $\{-1,1\}$. Other lines show the averaged distance functions for different values of p.*

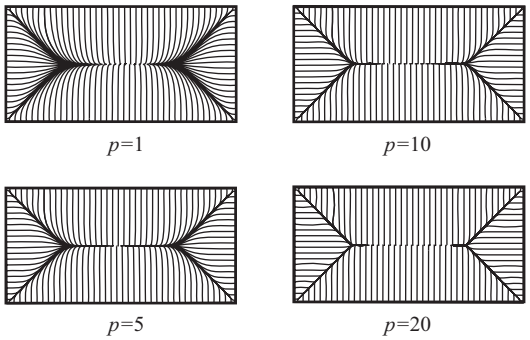



**Figure 27:** *Field lines of the gradient field of the distance function for several values of p.*

tance function from a point $\mathbf{x}$ to a surface $M$ as

$$d(\mathbf{x},M) = \inf_{\mathbf{y}\in M} |\mathbf{x}-\mathbf{y}| = \left(\sup_{\mathbf{y}\in M} |\mathbf{x}-\mathbf{y}|^{-1}\right)^{-1} = \left(\||\mathbf{x}-\mathbf{y}|^{-1}\|_{L_\infty(M)}\right)^{-1} \quad (8)$$

This definition lends itself to a natural generalization:

$$d_p(\mathbf{x},M) = \left(\|A^{-1}|\mathbf{x}-\mathbf{y}|^{-1}\|_{L_p(M)}\right)^{-1} = A^{1/p}\left(\int_M |\mathbf{x}-\mathbf{y}|^{-p} d\mathbf{y}\right)^{-1/p} \quad (9)$$

where $A$ is the area of the surface $M$. This normalization by the area is introduced to ensure that the gradient of this distance function is nondimensional and close to magnitude 1 at infinity, which mimicks the properties of the gradient of the euclidean distance. Intuitively, one can expect the gradient direction field of this function to have similar properties to the gradient field of the euclidean distance function as $p$ approaches $\infty$. Another intuitive interpretation of this distance function is as a potential field generated by charges on the surface raised to the power $-1/p$. In practice, we have observed that even for small values of $p$, the fields are quite similar. This is illustrated in Figure 26 and 27. The one-dimensional averaged distance functions are compared to the standard distance function in Figure 26, and fields of several values of $p$ in a two-dimensional box are shown in Figure 27. However, unlike the case of the euclidean distance function, the gradient of this function is well defined away from the surface, as the integration and differentiation can be exchanged. Using the averaged $d_p(\mathbf{x},M)$ yields the analog of Eq. 7 in which the gradient has an explicit expression and *the medial axis does not have to be computed explicitly.*

It can be proved that for $p > 1$ in 3D (and $p > 0$ in 2D), the direction of the gradient $g_p$, at points on a smooth surface, coincides with the normal[†]. Furthermore, in all our experiments we have observed that the magnitude of the gradient remains close to one near the surface, and decays in the area close to the conventional medial axis. So our function defines a fuzzy medial axis, pruning away insignificant branches corresponding to concavities, and with the gradient field close to zero only in areas where the shell genuinely cannot be expanded (see Figure 28 for the results of our two-dimensional experiments on deforming curves).

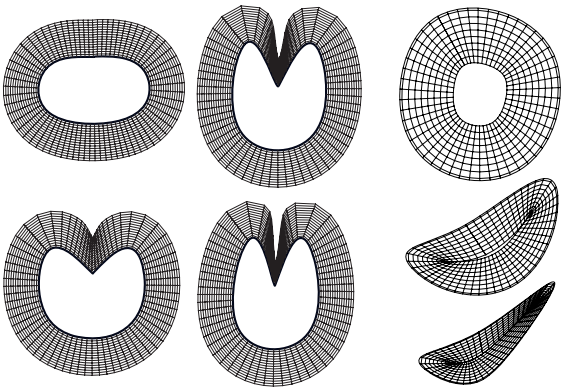


**Figure 28:** *The four diagrams on the left show self-adjusting shell behavior of an exterior shell in the concave region. With an angle of up to 90 degrees, no compression in shell thickness is observed, but at greater angles the shell starts to compress. The three diagrams in the right column illustrate an interior shell. The first image shows the interior shell for which a prescribed thickness is achieved. As the object is deformed, the shell compresses to avoid folds (prescribed thickness remains the same).*

[†] An interesting observation that $p = 1$ in 3D corresponds to the the electric field potential which makes it clear that this value cannot be used: e.g. the potential is constant inside a hollow uniformly charged sphere.

**Localization.** The function is supported over the whole surface. However, it does not make sense to take into account portions of the surface which are much further away than double the target shell thickness; thus, we integrate only over the parts of the surface which fit inside a sphere of radius $2h$, making our calculation local. The extra distance beyond $h$ is neccessary to ensure stability.

**Boundaries.** So far we have assumed that $M$ does not have a boundary. Near the boundary, the averaged distance function is likely to yield shells with considerable distortion due to the fact that the distance field has to make a 180-degree turn. The standard distance function handles this case well, but the averaged function gradient field turns in the outward direction. This problem is solved by adding artificial faces at the boundary. A single additional vertex is added for each boundary vertex. The direction to the new vertex is obtained by using a tangent direction across the boundary, and the distance is taken to be equal to the shell thickness. It should be noted that such an extension is satisfactory if there are no other parts of the surface near the boundary. Otherwise, the extension can overlap a different surface part.

### 6.6. Numerical and performance considerations

There are two main difficulties in using the averaged distance function to construct shells: we need to solve the ODE, which is stiff if the trajectory approaches the medial axis, and we need to compute the field gradient efficiently. While the ODE in most cases is well-behaved, it is stiff near the medial axis. The gradient, which is an integral over the surface, is also expensive to evaluate. We have evaluated several solution techniques (variants of explicit and implicit Euler and Runge-Kutta methods) and obtained the best performance and stability using an adaptive explicit Euler method. This algorithm is given below in somewhat simplified form, where $\Delta$ is the variable step size, $\mathbf{x}_0$ is the starting point on the surface, $\mathbf{x}$ is the current position along the trajectory, $g$ is the gradient of the averaged distance function at the point, $h$ is the prescribed thickness, and $\varepsilon$ is the adaptivity threshold for the change in the direction.

$\mathbf{x} = \mathbf{x}_0; \quad t = 0;$
$g = Field(\mathbf{x}_0);$
**while** $t < h$
  $\Delta = 2\Delta_0;$
  **do**
    $\Delta = \Delta/2;$
    $\mathbf{x}_{new} = \mathbf{x} + \Delta * g;$
    $g_{new} = Field(\mathbf{x}_{new});$
  **while** the angle between $g$ and $g_{new}$ is above $\varepsilon$;
  $\mathbf{x} = \mathbf{x}_{new}; \quad t += \Delta;$
  $g = g_{new};$
**end while**

**Computing integrals per face.** The simplest method for calculating the integrals is to do pointwise summation over the surface. However, this approach does not work well in the case where the sampling is fixed and the surface has very sharp angles. This is easy to understand if the sample points are thought of as charges, considering the field as a surface charge density field. The approximate gradient field may "escape" between points when a surface region with high curvature is not sampled densely enough for numerical methods; this results in shell inversion. This "escape" problem can be avoided by integrating analytically over triangles of the surface mesh (quads can be split into triangles for this purpose). Fortunately, it is possible to integrate $1/r^3$ over a wedge, and a triangle can be represented as a complement of three wedges in a plane, obtained by extending each triangle side in one direction. For a single wedge, the integral can be computed explicitly. Without losing generality, we can assume that the non-negative $x$ axis is the starting edge of the wedge, then the integral when $p = 3$ is

$$\int_{\angle} |\mathbf{x}-\mathbf{y}|^{-3} d\mathbf{y} = \frac{2}{w} \arctan\left(\frac{w}{(|\mathbf{x}|-u)\cot\frac{\beta}{2} - v}\right) \quad (10)$$

where $(u,v,w)$ is the coordinates of the point $\mathbf{x}$, and $\beta$ is the counter-clockwise angle of the wedge $\angle$. This formula is then differentiated on $u$, $v$, and $w$ for the calculation of gradient. Using these formulas, the integral over the mesh can be evaluated *precisely* if desired. While computing the gradient in this way is more expensive, this eliminates the need for refinement, and in fact using a coarser resolution version of the mesh yields good results.

**Accelerating integral computation.** The expense of computing the gradient can be considerable for an interactive application since it involves a surface integral.

Although we only integrate over a small part of the sur-

**Figure 29:** *Left: cross-section of the shell for a shape with sharp corners; Right: same object with volume texture added.*

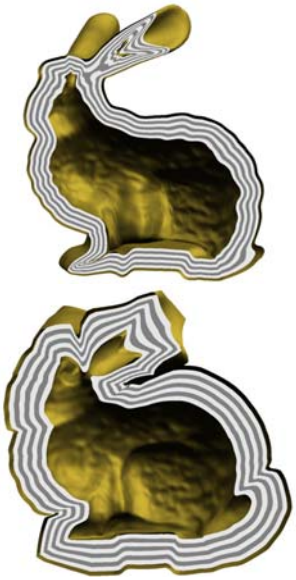


**Figure 30:** *Cross-sections of interior and exterior shells of the bunny.*

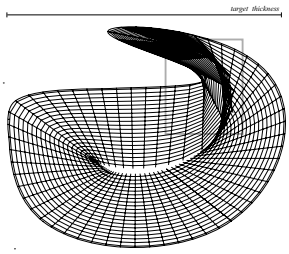



**Figure 31:** *Folding for extreme shell thickness (prescribed thickness equal to the objects bounding box size, only 70% of the shell shown to show the fold clearly.)*

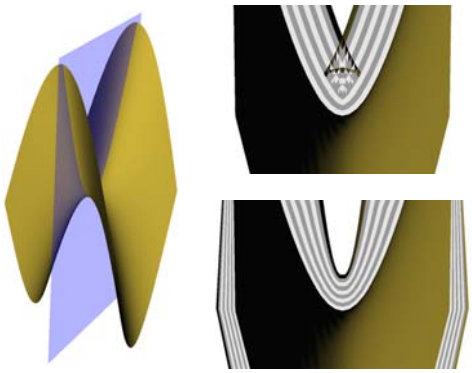


**Figure 32:** *Comparison of the results of normal displacement method (upper right) and our method (lower right) for a saddle.*

face, inside a ball near a given point, further acceleration helps. We use the Barnes and Hut algorithm [BH86] to compute the integral hierarchically. Although the calculation is already constant time, this algorithm is easy to implement, and provides a substantial speedup.

**Examples.** Several examples of external and internal shells and textures are shown in Figures 29,30 and 33-37. The timings for simpler objects were fractions of a second. For the bunny mesh in Figure 30, the external shell was generated in 1.8 sec on a 1GHz Pentium III, and the internal shell in 8 sec. The longer time for the internal mesh is due to refinement necessary to compute a valid shell inside the ears. The target thickness for the exterior shell was set at 10% of the bounding box size, and at 5% for the interior shell.

**Limitations of the approach.** The resulting shell is not guaranteed to be one-to-one; this is essentially inevitable, as we require directors to be straight. However, as shown in Figure 31, a rather large shell thickness needs to be prescribed with a special type of geometry for the failure to occur; for this figure, the requested shell thickness was close to the size of the bounding box of the whole object.

**Comparison with alternatives.** Shells created with normal displacement and with our method are compared in Figure 32. For a saddle as shown in the picture, the normal displacement method inevitably generates a self intersecting shell. It does not matter on which side of mesh the shell is expanded.

Level set methods, the fast marching method in particular, present the main alternative to our approach. However, the level set methods do not solve the problem of shell construction directly. The methods do not readily provide any mapping from the original surface to the advancing front, and the topology of the front may change. In fact this is an advantage for many applications but makes shell construction difficult. An additional step is required to establish the correspondence, as described in [Set94]. Another alternative is the envelope construction [CVM*96] which preserves the topology of the original surface. We have explored this approach and found that the thickness of such envelopes is very low in the regions of concavities, and the shape of the surface of the envelope tends to be undesirable in such areas.

Finally we note that [YP02] uses a conceptually similar (although numerically quite different) approach for constructing correspondences between surfaces, if we note that computing our integrals over the whole surface for $p = 1$ corresponds to solving the Laplace equation using Poisson formula. As we have pointed out, the value $p = 1$ does not work for constructing shells.

### 6.7. Results

In this section we describe a variety of operations that we have implemented in our modeling system using algorithms described in the previous sections.

**6.7.0.1. Deformations.** When the base surface is deformed, the shell needs to be recomputed. We take advantage of the locality of the field defining the shell, and recompute only the part which is within the field influence distance from the modified surface part. This can be done at interactive rates (Figure 33). We note that if a volume deformer is used to modify the surface, the same volume deformation can be applied to the shell and no interactive recomputation is necessary; however, for significant deformations it is still better to recompute the shell.

**6.7.0.2. Moving geometry along the surface.** Image editing operations can be relatively easily applied to volumetric textures, which results in changes in the implicitly defined geometry (Figure 34). However, when these operations are implemented, geometric distortion of the 2D texture mapping should be taken into account. This problem is identical to the one addressed in [BMBZ02]. The target area, to which the texture is moved, and the source area are reparameterized on a common planar domain with a distortion-minimizing parametrization. The common parameterization is used to resample the source texture over the target geometry. The same approach can be used for volume textures.

**6.7.0.3. Boolean operations and carving.** One of the advantages of volume geometry representations is that boolean operations become relatively simple (Figure 33). In the case of volume textures, the situation is complicated by the fact that the transformation from world coordinates to texture coordinates is nonlinear. However, it is still relatively straightforward to compute a boolean operation between a regular nondistorted volume object and the volume-textured surface: this requires resampling the volume object over the shell grid, which is straightforward.

Applying a boolean operation to two volume-textured surfaces is much more difficult.

**6.7.0.4. Animated Textures.** Removing details from the underlying geometric representation and placing them into 3D textures makes some animations much easier to execute. One example of this is the boiling man (Figure 35). The texture is procedurally animated to show the bubbles. Bubbles can easily appear, separate from the surface and burst, as they are represented implicitly. Another example of texture animation is growing trees on the surface. The speed of our shell generation algorithm also enables us to animate the base mesh and the texture at the same time (Figure 35,36).

**6.7.0.5. Rendering Performance.** The performance of the rendering algorithm is quite good, especially for large textures. The turbine blade shown in the video uses 128 slices through a 512x512x512 texture (compressed to 134 MB) and exhibits real-time performance. With 512 slices shown near the end of the clip, the quality is slightly greater, and the rendering time is still acceptable for interactive tasks.

On the other hand, when we try to stress geometric complexity, we run into performance limitations. For example, with the shirt shown in the video, we are limited to about 16 slices while still obtaining close to real-time performance (17fps with either normal or texture coordinate interpolation). The video was created using a Quadro 3000 card clocked at the standard 400/850 Mhz (core/memory).

### References


[Aga99] AGARWALA A.: *Volumetric Surface Sculpting*. Master's thesis, MIT, 1999.

[BAD*01] BOO M., AMOR M., DOGGETT M., HIRCHE J., STRASSER W.: Hardware support for adaptive subdivision surface rendering. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware* (2001).

[Bar84] BARR A. H.: Global and local deformations of solid primitives. *Proc. of SIGGRAPH 84* (1984), 21–30.

[BBF94] BARGHIEL C., BARTELS R., FORSEY D.: Pasting spline surfaces. In *Mathematical Methods for Curves and Surfaces: Ulvik, Norway* (1994), Vanderbilt University Press, pp. 31–40. Available at ftp://cgl.uwaterloo.ca/pub/users/rhbartel/Paste.ps.gz.

[Béz74] BÉZIER P.: Mathematical and practical possibilities of UNISURF. *CAD* (1974), 127–152.

[BH86] BARNES J., HUT P.: A hierarchical $O(N \log N)$ force calculation algorithm. *Nature 324* (1986), 446.

[BKS00] BISCHOFF S., KOBBELT L. P., SEIDEL H.-P.: Towards hardware implementation of loop subdivision. In *Proc. of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics Hardware* (2000), pp. 41–50.

[BKZ01] BIERMANN H., KRISTJANSSON D., ZORIN D.: Approximate boolean operations on free-form solids. In *Proceedings of SIGGRAPH 01* (August 2001), pp. 185–194.

[Blo97] BLOOMENTHAL J. (Ed.): *Introduction to implicit surfaces*. Morgan Kaufmann, 1997.

[BLZ00] BIERMANN H., LEVIN A., ZORIN D.: Piecewise smooth subdivision surfaces with normal control. In *Proceedings of SIGGRAPH 00* (2000), pp. 113–120.

**Figure 33:** *Editing operations: deforming a volume-textured surface and cutting a hole on the chain-mail shirt.*

**Figure 34:** *The first two are simple objects with small-scale geometry added. The last two show the operation of moving a geometric texture on a surface.*

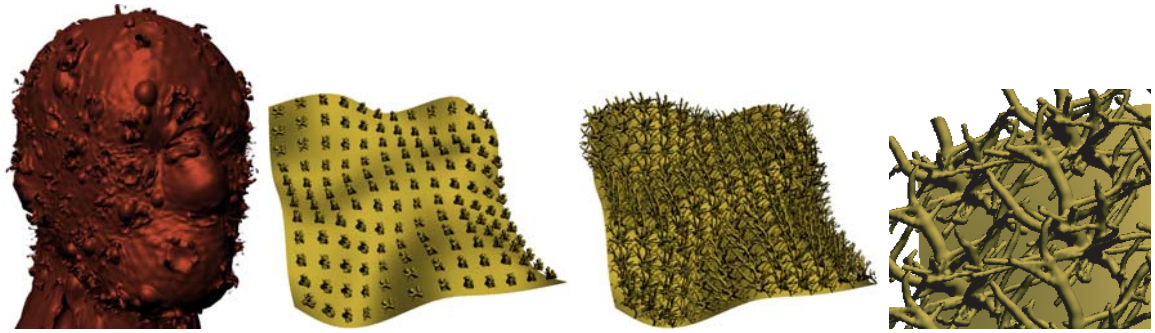


**Figure 35:** *An animated bubbling texture applied on a deforming head and two stages of a growing bush texture with a zoomed-in view.*

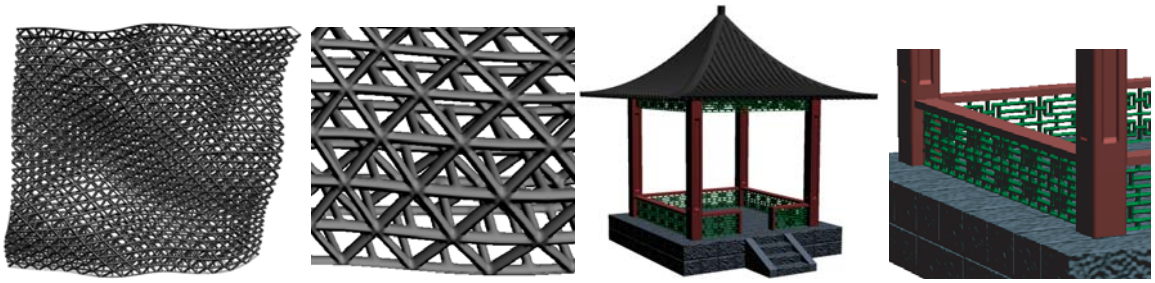


**Figure 36:** *A structural texture on a deforming plane and a kiosk. The second and the forth pictures are showing zoomed-in details. Volume textures are used on the kiosk roof and walls.*

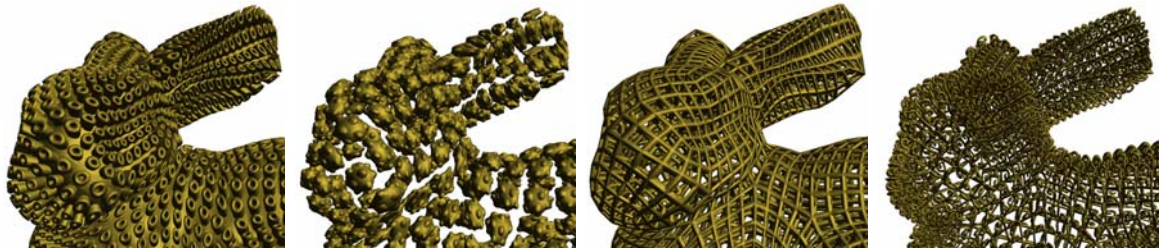


**Figure 37:** *Several different volumetric textures applied on the bunny. Only the heads are shown to view the geometric details.*


[BMBZ02] BIERMANN H., MARTIN I., BERNARDINI F., ZORIN D.: Cut-and-paste editing of multiresolution surfaces. *ACM TOG. Special issue for SIGGRAPH conference 21*, 3 (2002), 312–321.

[BMRB04] BOIER-MARTIN I., RONFARD R., BERNARDINI F.: Detail-preserving variational surface design with multiresolution constraints. In *Proc. Shape Modeling International, SMI'04* (2004).

[BMZB02] BIERMANN H., MARTIN I., ZORIN D., BERNARDINI F.: Sharp features on multiresolution subdivision surfaces. *Graphical Models 64*, 2 (2002), 61–77.

[BP00] BERN M., PLASSMANN P.: Mesh generation. In *Handbook of computational geometry*. North-Holland, Amsterdam, 2000, pp. 291–332.

[Bri87] BRIGGS W. L.: *Multigrid Tutorial*. SIAM, 1987.

[BZ99] BIERMANN H., ZORIN D.: Subdivide 2.0 software, 1999.

[cat] In *http://www.catia.ibm.com*.

[CBC*01] CARR J. C., BEATSON R. K., CHERRIE J. B., MITCHELL T. J., FRIGHT W. R., MCCALLUM B. C., EVANS T. R.: Reconstruction and representation of 3d objects with radial basis functions. In *Proceedings of ACM SIGGRAPH 2001* (2001), pp. 67–76.

[CC78] CATMULL E., CLARK J.: Recursively generated B-spline surfaces on arbitrary topological meshes. *CAD 10*, 6 (1978), 350–355.

[CDM*02] CUTLER B., DORSEY J., MCMILLAN L., MÜLLER M., JAGNOW R.: A procedural approach to authoring solid models. In *ACM Transactions on Graphics (SIGGRAPH 2001)* (2002), vol. 21-3, pp. 302–311.

[CG99] CELNIKER G., GOSSARD D.: Energy-based models for free-form surface shape design. In *Proc. ASME Design Automation Conference* (1999).

[CM00] CONRAD B., MANN S.: Better pasting via quasi-interpolation. In *Curve and Surface Design: Saint-Malo, 1999* (Nashville, TN, 2000), Laurent P.-J., Sablonnière P.„ Schumaker L. L., (Eds.), Vanderbilt University Press, pp. 27–36.

[CMB97] CHAN L. K. Y., MANN S., BARTELS R.: World space surface pasting. In *Proceedings of Graphics Interface* (May 1997), Davis W., Mantei M.„ Klassen V., (Eds.), pp. 146–154.

[Coq90] COQUILLART S.: Extended free-form deformation: a sculpturing tool for 3d geometric modeling. In *Proc. of SIGGRAPH 90* (1990), pp. 187–196.

[CR94] CHANG Y., ROCKWOOD A. P.: A generalized de casteljau approach to 3D Free-Form deformation. *Proc. of SIGGRAPH 94* (1994), 257–260.

[CRE01] COHEN E., RIESENFELD R. F., ELBER G.: *Geometric Modeling with Splines*. A K Peters Ltd, 2001.

[CSA*02] CIRAK F., SCOTT M., ANTONSON E., ORTIZ M., SCHRÖDER P.: Integrated modeling, finite-element analysis, and engineering design for thin-shell structures using subdivision. *Computer Aided Design 43* (2002), 137–148.

[CST94] CHANG M. M., SEZAN M. I., TEKALP A. M.: Adaptive bayesian segmentation of color images. *Journal of Electronic Imaging 3* (1994), 404–414.

[CVM*96] COHEN J., VARSHNEY A., MANOCHA D., TURK G., WEBER H., AGARWAL P., JR. F. P. B., WRIGHT W.: Simplification envelopes. In *Proceedings of SIGGRAPH 96* (1996), pp. 119–128.

[DEL*99] DORSEY J., EDELMAN A., LEGAKIS J., JENSEN H. W., PEDERSEN H. K.: Modeling

and rendering of weathered stone. In *Proceedings of ACM SIGGRAPH 99* (August 1999), pp. 225–234.

[dF73] DEBOOR C., FIX G. J.: Spline approximtion by quasiinterpolants. *Journal of Approximation Theory 8* (1973), 19–45.

[DKT98] DEROSE T., KASS M., TRUONG T.: Subdivision surfaces in character animation. In *Proceedings of SIGGRAPH 98* (1998), pp. 85–94.

[DL02] DYN N., LEVIN D.: Subdivision schemes in geometric modelling. *Acta Numerica 11* (2002).

[DLG90] DYN N., LEVIN D., GREGORY J. A.: A butterfly subdivision scheme for surface interpolation with tension control. *ACM TOG 9*, 2 (April 1990), 160–169.

[DMA02] DESBRUN M., MEYER M., ALLIEZ P.: Intrinsic parameterizations of surface meshes. In *Eurographics conference proceedings* (2002), pp. 209–218.

[DMR02] DIEWALD U., MORIGI S., RUMPF M.: A cascadic geometric filtering approach to subdivision. *CAGD 19*, 9 (2002), 675 – 694.

[Doo78] DOO D.: A subdivision algorithm for smoothing down irregularly shaped polyhedrons. In *Proceedings on Interactive Techniques in Computer Aided Design* (Bologna, 1978), pp. 157–165.

[DS78] DOO D., SABIN M.: Analysis of the behaviour of recursive division surfaces near extraordinary points. *CAD 10*, 6 (1978), 356–360.

[dsm] In *http://www.discreet.com*.

[EGM*94] EBERLY D., GARDNER R., MORSE B., PIZER S., SCHARLACH C.: Ridges for image analysis. *Journal of Mathematical Imaging and Vision 4* (1994), 351–371.

[FMMY03] FURUKAWA Y., MASUDA H., MIURA K. T., YAMATO H.: Cut-and-paste editing based on constrained b-spline volume fitting. In *Proceedings Computer Graphics International* (2003).

[FMS03] FRIEDEL I., MULLEN P., SCHRODER P.: Data-dependent fairing of subdivision surfacesn. In *Proc. of SM 03* (2003).

[FPRJ00] FRISKEN S. F., PERRY R. N., ROCKWOOD A. P., JONES T. R.: Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proceedings of ACM SIGGRAPH 2000* (2000), pp. 249–254.

[GKSS02] GUSKOV I., KHODAKOVSKY A., SCHRÖDER P., SWELDENS W.: Hybrid meshes: Multiresolution using regular and irregular refinement. In *Proc. Symp. Comp. Geom.* (2002), pp. 264–272.

[GM97] GIBSON S. F. F., MIRTICH B.: *A Survey of Deformable Modeling in Computer Graphics*. Tech. Rep. TR-97-19, MERL, Cambridge, MA, 1997.

[GOP99] GONZALEZ-OCHOA C., PETERS J.: Localized-hierarchy surface splines (less). In *Proc. Symp. on Interactive 3D Graphics* (1999), pp. 7–15.

[Gre94] GREINER G.: Surface construction based on variational principles. In *Wavelets, Images and Surface Fitting*, Laurent P. J., LeMéhauté A.„ Schumaker L., (Eds.). AK Peters, 1994, pp. 277–286.

[GS01] GRINSPUN E., SCHRÖDER P.: Normal bounds for subdivision-surface interference detection. In *Proc. of IEEE Visualization 01* (2001).

[Hal96] HALSTEAD M. A.: *Efficient Techniques for Surface Design Using Constrained Optimization*. PhD thesis, Univ. of California at Berkeley, 1996.

[HDD*94] HOPPE H., DEROSE T., DUCHAMP T., HALSTEAD M., JIN H., MCDONALD J., SCHWEITZER J., STUETZLE W.: Piecewise smooth surface reconstruction. In *Computer Graphics Proceedings* (1994), Annual Conference Series, ACM Siggraph, pp. 295–302.

[Hen96] HENSHAW W. D.: Automatic grid generation. In *Acta numerica, 1996*, vol. 5 of *Acta Numer.* Cambridge Univ. Press, Cambridge, 1996, pp. 121–148.

[HKD93] HALSTEAD M., KASS M., DEROSE T.: Efficient, fair interpolation using Catmull-Clark surfaces. In *Proc. SIGGRAPH 1993* (1993), pp. 35–44.

[HQ03] HUA J., QIN H.: Free-Form deformations via sketching and manipulating scalar fields. In *Proc. ACM Symp. on Solid Modeling and Applications* (2003), pp. 328 – 333.

[KK89] KAJIYA J. T., KAY T. L.: Rendering fur with three dimensional textures. In *Proceedings of the 16th annual conference on Computer graphics and interactive techniques (SIGGRAPH 89)* (1989), pp. 271–280.

[Kob96a] KOBBELT L.: Interpolatory subdivision on openquadrilateral nets with arbitrary topology.

In *Proc. of Eurographics 96* (1996), pp. 409–420.

[Kob96b] KOBBELT L.: A variational approach to subdivision. *Comput. Aided Geom. Design 13*, 8 (1996), 743–761.

[Kob00] KOBBELT L. P.: Discrete fairing and variational subdivision for freeform surface design. *The Visual Computer 16*, 3-4 (2000), 142–150.

[KS99] KHODAKOVSKY A., SCHRÖDER P.: Fine level feature editing for subdivision surfaces. In *Proceedings of ACM Solid Modeling* (1999).

[LCJ94] LAZARUS F., COQUILLART S., JANCENE P.: Axial deformations: an intuitive deformation technique. *CAD 26*, 8 (1994), 607–61.

[LDW97] LOUNSBERY M., DEROSE T., WARREN J.: Multiresolution analysis for surfaces of arbitrary topological type. *ACM TOG 16*, 1 (January 1997), 34–73.

[Len00] LENGYEL J.: Real-time hair. In *Rendering Techniques 2000: 11th Eurographics Workshop on Rendering* (June 2000), Eurographics, pp. 243–256.

[Lev99] LEVIN A.: Interpolating nets of curves by smooth subdivision surfaces. *Proc. of SIGGRAPH 99* (1999), 57–64.

[LFKN03] LANQUETIN S., FOUFOU S., KHEDDOUCI H., NEVEU M.: Computing subdivision surface intersection. In *Proc. WSCG '03* (2003).

[Lin00] LINSEN L.: Netbased modelling. In *Proc. SCCG '00* (2000), pp. 259–266.

[LLS01a] LITKE N., LEVIN A., SCHRÖDER P.: Fitting subdivision surfaces. In *Proc. of IEEE Visualization 2001* (October 2001), pp. 319–324.

[LLS01b] LITKE N., LEVIN A., SCHRODER P.: Trimming for subdivision surfaces. *Computer Aided Geometric Design 18*, 5 (2001), 463–481.

[LMH00] LEE A., MORETON H., HOPPE H.: Displaced subdivision surfaces. In *Proc. of SIGGRAPH 00* (2000), pp. 85–94.

[Loo87] LOOP C.: *Smooth Subdivision Surfaces Based on Triangles*. Master's thesis, University of Utah, Department of Mathematics, 1987.

[LPFH01] LENGYEL J. E., PRAUN E., FINKELSTEIN A., HOPPE H.: Real-time fur over arbitrary surfaces. In *2001 ACM Symposium on Interactive 3D Graphics* (2001), pp. 227–232.

[Ma00] MA M.: *The Direct Manipulation of Pasted Surfaces*. Master's thesis, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1, 2000. Available on WWW as ftp://cs-archive.uwaterloo.ca/cs-archive/CS-2000-15/.

[may] In *http://www.alias.com*.

[MJ96] MACCRACKEN R., JOY K. I.: Free-Form deformations with lattices of arbitrary topology. In *Proc. of SIGGRAPH 96* (1996), pp. 181–188.

[MN98] MEYER A., NEYRET F.: Interactive volumetric textures. In *Eurographics Rendering Workshop 1998* (New York City, NY, July 1998), Drettakis G., Max N., (Eds.), Eurographics, Springer Wein, pp. 157–168.

[MQ00] MCDONNELL K., QIN H.: Dynamic sculpting and animation of Free-Form subdivision solids. In *Proc. of IEEE Computer Animation* (2000).

[MZ00] MA W., ZHAO N.: Catmull-clark surface fitting for reverse engineering applications. In *Proceedings of Geometric Modeling and Processing* (2000), pp. 274–282.

[NA02] NASRI A. H., ABBAS A.: Designing Catmull-Clark subdivision surfaces with curve interpolation constraints. *Computers and Graphics* (2002).

[Nas91] NASRI A. H.: Surface interpolation on irregular networks with normal conditions. *CAGD 8* (1991), 89–96.

[Nas00] NASRI A.: Interpolating meshes of boundary intersecting curves by subdivision surfaces. *The Visual Computer 16* (2000).

[Ney95] NEYRET F.: A general and multiscale model for volumetric textures. In *Graphics Interface '95* (May 1995), Davis W. A., Prusinkiewicz P., (Eds.), Canadian Information Processing Society, Canadian Human-Computer Communications Society, pp. 83–91.

[Ney98] NEYRET F.: Modeling animating and rendering complex scenes using volumetric textures. *IEEE Transactions on Visualization and Computer Graphics 4*, 1 (Jan.–Mar. 1998), 55–70.

[PASS95] PASKO A., ADZHIEV V., SOURIN A., SAVCHENKO V.: Function representation in geometric modeling: concepts, implementation and applications. *The Visual Computer 11*, 8 (1995), 429–446.

[PB00] PIPONI D., BORSHUKOV G.: Seamless texture mapping of subdivision surfaces by model pelting and texture blending. In *Proceedings of SIGGRAPH 00* (2000), pp. 471–478.

[PBC*94] PIZER S., BURBECK C., COGGINS J.,

FRITSCH D., MORSE B.: Object shape before boundary shape: Scale-space medial axes. *Journal of Mathematical Imaging and Vision 4* (1994), 303–313.

[Pet00] PETERS J.: Patching Catmull-Clark meshes. In *Proceedings of SIGGRAPH 00* (2000), pp. 255–258.

[PKZ04] PENG J., KRISTJANSSON D., ZORIN D.: Interactive modeling of topologically complex geometric detail. *ACM Transactions on Graphics (SIGGRAPH 2004 Processings) 23*, 3 (2004), 635–643.

[PL97] PULLI K., LOUNSBERY M.: *Hierarchical Editing and Rendering of Subdivision Surfaces*. Tech. Rep. UW-CSE-97-04-07, Dept. of CS&E, University of Washington, Seattle, WA, 1997.

[PR97] PETERS J., REIF U.: The simplest subdivision scheme for smoothing polyhedra. *ACM TOG 16*, 4 (1997).

[PS96] PULLI K., SEGAL M.: Fast rendering of subdivision surfaces. In *Proc. Eurographics Workshop on Rendering* (1996), pp. 61–70.

[QMV98] QIN H., MANDAL C., VEMURI B.: Dynamic Catmull-Clark subdivision surfaces. *IEEE TVCG 4*, 3 (1998), 215–229.

[QT96] QIN H., TERZOPOULOS D.: D-NURBS: A physics based framework for geometric design. *IEEE TVCG 2*, 1 (1996), 85–96.

[Ric73] RICCI A.: A constructive geometry for computer graphics. *The Computer Journal 16*, 2 (1973), 157–160.

[Roc87] ROCKWOOD A.: *Blending surfaces in solid geometric modeling*. PhD thesis, Cambridge University, 1987.

[Sab71] SABIN M.: Interrogation techniques for parametric surfaces. In *Advanced computer graphics - economics, techniques and applications*, Parslow R. D., Green R. E., (Eds.). Plenum Press, 1971, pp. 1095–1118.

[Sab02] SABIN M. A.: Subdivision surfaces tutorial. *SMI 02* (2002).

[SBTZ99] SIDDIQI K., BOUIX S., TANNENBAUM A., ZUCKER S. W.: The hamilton-jacobi skeleton. In *ICCV (2)* (1999), pp. 828–834.

[Sch96] SCHWEITZER J. E.: *Analysis and Application of Subdivision Surfaces*. PhD thesis, University of Washington, Seattle, 1996.

[SDS96] STOLLNITZ E. J., DEROSE T., SALESIN D. H.: *Wavelets for computer graphics: theory and applications*. Morgan Kaufmann, 1996.

[SdS01] SHEFFER A., DE STURLER E.: Parameterization of faceted surfaces for meshing using angle based flattening. *Engineering with Computers 17(3)* (2001), 326–337.

[Sei98] SEIDEL R.: The nature and meaning of perturbations in geometric computing. *Discrete Computational Geometry 19*, 1 (1998), 1–17.

[Set94] SETHIAN J. A.: Curvature flow and entropy conditions applied to grid generation. *J. Comput. Phys. 115*, 2 (1994), 440–454.

[Set99] SETHIAN J.: *Level Set Methods and Fast Marching Methods*. Cambridge University Press, 1999. ISBN 0521645573.

[SF98] SINGH K., FIUME E.: Wires: A geometric deformation technique. In *Proc. of SIGGRAPH 98* (1998), pp. 405–414.

[SK93] STEINBERG S., KNUPP P. M.: *Fundamentals of Grid Generation*. CRC Press, 1993.

[SP86] SEDERBERG T. W., PARRY S. R.: Free-form deformation of solid geometric models. In *Proc. of SIGGRAPH 86* (1986), pp. 151–160.

[Sta98] STAM J.: Exact evaluation of catmull-clark subdivision surfaces at arbitrary parameter values. In *Proceedings of SIGGRAPH 98* (July 1998), pp. 395–404.

[STKK99] SUZUKI H., TAKEUCHI S., KIMURA F., KANAI T.: Subdivision surface fitting to a range of points. In *Proc. Pacific Graphics* (1999).

[SWZ04] SCHAEFFER S., WARREN J., ZORIN D.: Lofting curve networks using subdivision surfaces. *submitted* (2004).

[SZSS98] SEDERBERG T. W., ZHENG J., SEWELL D., SABIN M.: Non-uniform recursive subdivision surfaces. In *Proceedings of SIGGRAPH 98* (1998), pp. 387–394.

[Tak98] TAKAHASHI S.: Variational design of curves and surfaces using multiresolution constraints. *The Visual Computer 14(5/6)* (1998), 208–227.

[TF88] TERZOPOULOS D., FLEISCHER K.: Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. In *Proc. of SIGGRAPH '88* (1988), pp. 269–278.

[TO02] TURK G., O'BRIEN J.: Modelling with implicit surfaces that interpolate. *ACM TOG 21*, 4 (2002), 855 – 873.

[TPBF87] TERZOPOULOS D., PLATT J., BARR A., FLEISCHER K.: Elastically deformable models. In *Proc. of SIGGRAPH 87* (1987), pp. 205–214.

[TQ94] TERZOPOULOS D., QIN H.: Dynamic NURBS with geometric constraints for interactive sculpting. *ACM TOG 13*, 2 (1994), 103–136.

[WK95] WANG S. W., KAUFMAN A. E.: Volume sculpting. In *Proceedings of the 1995 symposium on Interactive 3D graphics* (1995), ACM Press, pp. 151–156.

[WW92] WELCH W., WITKIN A.: Variational surface modeling. In *Proceedings of SIGGRAPH '92* (1992), vol. 26, pp. 157–166.

[WW98] WEIMER H., WARREN J.: Subdivision schemes for thin-plate splines. *Proc. EUROGRAPHICS '98 17*, 3 (1998).

[WW01] WARREN J., WEIMER H.: *Subdivision Methods for Geometric Design: A Constructive Approach*. Morgan Kaufmann, 2001.

[YP02] YEZZI A., PRINCE J. L.: A PDE approach for thickness, correspondence, and gridding of annular tissues. In *ECCV (4)* (2002), vol. 2353 of *Lecture Notes in Computer Science*, Springer, pp. 575–589.

[ZK02] ZORIN D., KRISTJANSSON D.: Evaluation of piecewise smooth subdivision surfaces. *The Visual Computer 18*, 5–6 (2002), 299 – 315.

[ZSD*00] ZORIN D., SCHRÖDER P., DEROSE T., KOBBELT L., LEVIN A., SWELDENS W.: Subdivision for modeling and animation. SIGGRAPH'00 Course Notes, 2000.

[ZSS96] ZORIN D., SCHRÖDER P., SWELDENS W.: Interpolating subdivision for meshes with arbitrary topology. In *Proc. of SIGGRAPH 96* (August 1996), pp. 189–192.

[ZSS97] ZORIN D., SCHRÖDER P., SWELDENS W.: Interactive multiresolution mesh editing. In *Proc. of SIGGRAPH 97* (1997), pp. 259–268.

# Mesh Editing based on Discrete Laplace and Poisson Models

Marc Alexa

Discrete Geometric Modeling Group
Darmstadt University of Technology

**Abstract**

*Surface editing operations commonly require geometric details of the surface to be preserved as much as possible. We argue that geometric detail is an intrinsic property of a surface and that, consequently, surface editing is best performed by operating over an intrinsic surface representation. This intrinsic representation could be derived from differential properties of the mesh, i.e. its Laplacian. The modeling process poses nonzero boundary constraints so that this idea results in a Poisson model. Different ways of representing the intrinsic geometry and the boundary constraints result in alternatives for the properties of the modeling system. In particular, the Laplacian is not invariant to scaling and rotations. Either the intrinsic representation is enhanced to be invariant to (linearized) transformations, or scaling and rotation are computed in a preprocess and are modeled as boundary constraints. Based on this representation, useful editing operations can be developed: Interactive free-form deformation in a region of interest based on the transformation of a handle, transfer and mixing of geometric detail between two surfaces, and transplanting of a partial surface mesh into another surface. The main computation involved in all operations is the solution of a sparse linear system, which can be done at interactive rates. We demonstrate the effectiveness of this approach in several examples, showing that the editing operations change the shape while respecting the structural geometric detail.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modelingcurve, surface, solid and object representations

## 1. Introduction

Surfaces in computer graphics are mostly represented in global coordinate systems: explicit representations are based on points, vertices, or nodes that are typically described using absolute Euclidean coordinates. Implicit representations describe the shape as the level set of a function defined in Euclidean space. A global coordinate system is the natural choice for all operations involving other objects such as rendering, intersection testing and computation, transformations, or CSG modeling. On the other hand, for local surface modeling, it would be desirable that the representation captures the local shape (i.e. the intrinsic geometry of the surface) rather than the absolute position or orientation in Euclidean space.

Manipulating and modifying a surface while preserving the geometric details is important for various surface editing operations, including free-form deformations [SP86, Coq90], cut and paste [RIKM93, BMBZ02], fusion [KSMK99], morphing [Ale03a], and others. Note that the absolute position of the vertices in a mesh is not important for these operations, which calls for an intrinsic surface representation.

A partially intrinsic surface mesh representation are multi-resolution decompositions [FB88, ZSS97, KCVS98, KVS99, GSS99]. In a multi-resolution mesh, the geometry is encoded as a base mesh and several levels of refinement. The refinement is typically described locally, so that geometric details are mostly captured in a discrete set of intrinsic coordinates. Using this representation, several modeling operations can be performed on an appropriate user-specified level-of-detail. Note, however, that the locality of multi-resolution representations is potentially limited: The support (or extent) of the representation of a single vertex increases from fine to coarse levels of the hierarchy. Thus, modeling operations are restricted to a discrete set of regions and levels-of-detail. For example, when cutting out a partial mesh for transplanting operations, the original multi-resolution representation is in-

validated because parts of the base domain and (potentially) other levels of the hierarchy are missing.

This approach to encode geometric details is to use differentials as coordinates for the vertices [Ale03b, BK04, SLCO*04, YZX*04]. This provides a fully intrinsic representation of the surface mesh, where the reconstruction of global coordinates from the intrinsic representation always preserves the intrinsic geometry as much as possible given the modeling constraints. Using a differential representation for editing operations has been shown to be quite effective in image domain [FLW02, PGB03]. The image domain has a natural regular parameterization and resulting inherent definition of a gradient, which allows modeling many editing tasks as a discrete Poisson equation. However, this approach cannot be directly applied or adapted to discrete (as well as continuous) surfaces.

## 2. The Laplacian representation

Let the mesh $\mathcal{M}$ be described by a pair $(K,V)$, where $K$ is a simplicial complex representing the connectivity of vertices, edges, and faces, and $V = \{\mathbf{v}_1,\ldots,\mathbf{v}_n\}$ describes the geometric positions of the vertices in $\mathbb{R}^3$. We use the following terminology: the *neighborhood ring* of a vertex $i$ is the set of adjacent vertices $\mathcal{N}_i = \{j|(i,j) \in K\}$ and the *degree* $d_i$ of this vertex is the number adjacent edges (or vertices), i.e. the number of elements in $\mathcal{N}_i$. We assume that $d_i > 0$, i.e. that the mesh is connected.

Instead of using absolute coordinates $V$, we would like to describe the mesh geometry using a set of differentials $\Delta = \{\delta_i\}$. Specifically, coordinate $i$ will be represented by its Laplace vector. There are different ways to define a discretized version of the Laplace operator for meshes, and each of them has certain advantages. Most of them are based on the one-ring of a vertex

$$\delta_i = \mathbf{v}_i - c_{ij}\sum_{j\in\mathcal{N}_i}\mathbf{v}_j, \quad \sum_j c_{ij} = 1, \tag{1}$$

and differ in the definition of the coefficients $c_{ij}$. The topological Laplacian ([Tau95]) simply uses the similar weights for all neighboring vertices, i.e. $c_{ij} = 1/d_i$. In our and others' experience the the cotangent weights (e.g. [**?**]) perform best in most applications.

The transformation between $V$ and $\Delta$ can be described in matrix algebra. Let $C = \{c_{ij}\}$, then

$$\Delta = (I - C)V. \tag{2}$$

The Laplacian $L = I - C$ is invariant under translation, however, sensitive to linear transformations. Thus, $L$ is expected to have rank $n-1$, which means $V$ can be recovered from $\Delta$ by fixing one vertex and solving a linear system of equations.

## 3. Mesh modeling framework

The basic idea of the modeling framework is to satisfy linear modeling constraints (exactly, or in the least squares sense), while preserving differential properties of the original geometry in the least squares sense [Ale03b, LSCOL04]. Without additional linear constraints the deformed geometry $V'$ is then defined by

$$\min_{V'}\sum_{i=1}^{n}\left\|\delta_i - \left(\mathbf{v}'_i - \frac{1}{d_i}\sum_{j\in\mathcal{N}_i}\mathbf{v}'_j\right)\right\|^2. \tag{3}$$

If the original surface was a membrane, the necessary constraints for the minimizer lead to $L^2V = 0$, which has been advocated by Botsch and Kobbelt [BK04] in the context of modeling smooth surfaces. If, in contrast, the original surface contained some detail, the right-hand side is non-zero and we arrive at a variant of the discrete Poisson modeling approach of Yu et al. [YZX*04].

The basic type of linear modeling constraints is to prescribe the absolute position of some vertices, i.e. $\mathbf{v}'_i = \hat{\mathbf{v}}_i$. These constraints are best incorporated by also satisfying them in the least squares sense, possibly weighted to trade-off between modeling constraints and the reproduction of original surface geometry.

We found that the easiest way of implementing the approach is to write the conditions to be satisfied in the least squares sense as a large rectangular system $\mathbf{AV}' = \mathbf{b}$ and then solve $\mathbf{A}^T\mathbf{AV}' = \mathbf{A}^T\mathbf{b}$. Prescribing positions for some vertices then simply yields additional rows of the form

$$w_i\|\mathbf{v}'_i = \hat{\mathbf{v}}_i. \tag{4}$$

Note that in fact these are three rows for each constraint, as $\mathbf{v}$ are column vectors with three elements.

This framework can be extended towards constraints on arbitrary points on the mesh. Note that each point on the surface is the linear combination of two or three vertices. A point on an edge between vertices $i$ and $j$ is defined by one parameter as $(1-\lambda)\mathbf{v}_i + \lambda\mathbf{v}_j$, $0 \le \lambda \le 1$. Similarly, a point on a triangle is defined by two parameters. We can put positional constraints $\hat{\mathbf{v}}_{ij}$ on such a point by adding rows of the form

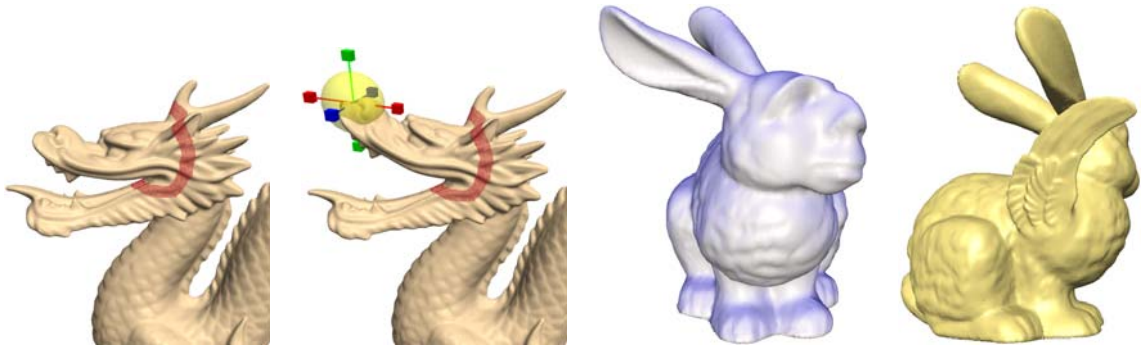
$$(1-\lambda)v'_i + \lambda v'_j = \hat{\mathbf{v}}_{ij} \tag{5}$$

to the system matrix $A$.

Furthermore, also differentials could be prescribed. Note that $\delta_i$ points roughly in normal direction at vertex $i$ and that its length is proportional to the mean curvature. This allows us to prescribe a certain normal direction and/or curvature for a vertex, simply by adding a row of the form

$$\mathbf{v}'_i - \sum_{j\in\mathcal{N}_i} c_{ij}\mathbf{v}'_j = \hat{\delta}_i. \tag{6}$$

The modeling operation is typically localized on a part of



(a) (b) (c) (d)

**Figure 1:** *Advanced mesh editing operations using Laplacian coordinates: free-form deformations (a-b), detail transfer (c) and mesh transplanting (d). Representing the geometry using the Laplacian coordinates enables preservation of detail.*

the mesh. This part of the mesh is selected by the user as the region of interest (ROI) during the interactive modeling session. The operations are restricted to this ROI, padded by several layers of anchor vertices. The anchor vertices yield positional constraints $\mathbf{v}_i' = \hat{\mathbf{v}}_i$ in the system matrix $A$, which ensure a gentle transition between the altered ROI and the fixed part of the mesh.

Based on the constraints formulated so far, local surface detail is preserved if parts of the surface are translated, but changes with rotations and scales. There are several ways of dealing with linear transformations:

- They could be defined and prescribed based on the modeling operation [YZX*04].
- They could be deduced from the membrane solution (i.e. $LV' = 0$) [LSCOL04].
- They could be implicitly defined by the solution, if the rotational part is linearized [SLCO*04].

In any case, we first need to extend the definition of the local intrinsic representation to incorporate linear transformations.

## 4. Incorporating linear transformations

The main idea to account for local linear transformations is to assign each vertex $i$ an individual transformation $T_i$. These transformation are then applied to the original geometry by a transforming each local Laplacian $\delta_i$ with $T_i$. This results in a slightly modified functional defining the resulting geometry $V'$:

$$\min_{V'} \sum_{i=1}^{n} \left\| T_i \delta_i - \left( \mathbf{v}_i' - \frac{1}{d_i} \sum_{j \in \mathcal{N}_i} \mathbf{v}_j' \right) \right\|^2 \qquad (7)$$

Note that in the formulation of this minimization as solving a system $AV' = b$ the part $T_i\delta_i$ is contained in the right-hand side column vector $b$. This is important because it implies the system $A$ can be solved independent of the transformations $T_i$ to be applied to vertex $i$, allowing the $T_i$ to be changed during interactive modeling.

The following approaches vary in how the local transformations $T_i$ are computed.

### 4.1. Prescribing the transformations

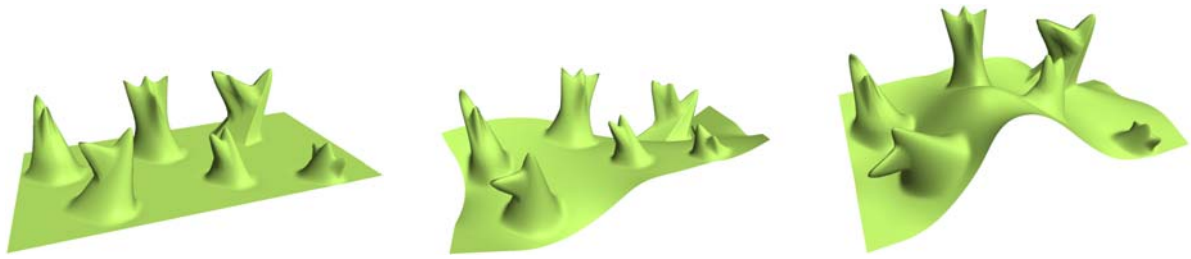
Yu et al. [YZX*04] let the user specify a few constraint transformations and then interpolate them over the surface. In particular, the rotational and scaling parts are treated independently, i.e. the transformation is factored as $T_i = R_i S_i$, where $R_i$ is the local rotation and $S_i$ is a symmetric matrix containing scale and shear. Initially all vertices are assumed to be not rotated, scaled or sheared. Modeling operations might induce local linear transformations.

One could view this (slightly more general as in [YZX*04]) as a scattered data interpolation problem: In few vertices a (non-zero) rotation or non-unity scale are given. All vertices should then be assigned a scale and rotation so that the given constraints are satisfied and the field of rotations and scales is smooth. In order to apply well-known techniques only a distance measure for the vertices is necessary. Yu et al. [YZX*04] use the topological distance of vertices in the mesh.

Then, each local rotation and scale are a distance-weighted average of given transformations. The easiest way to derive the distance weights would be Shephard's approach. This defines $T_i$ for each vertex and, thus, $V'$. Note that transformations can be changed interactively.

### 4.2. Transformations from the membrane solution

Lipman et al. [LSCOL04] compute the rotations from the membrane solution. They first solve $\Delta V' = 0$ and then compute each transformation $T_i$ based on comparing the one-rings in $\mathbf{V}$ and $\mathbf{V}'$ of vertex $i$.



(a) (b) (c)

**Figure 2:** *Deformations of a model (a) with detail that cannot be expressed by height field. The deformation changes the global shape while respecting the structural detail as much as possible.*

The basic idea for a definition of $T_i$ is to derive it from the transformation of $\mathbf{v}_i$ and its neighbors to $\mathbf{v}'_i$ and its neighbors:

$$\min_{T_i} \left( \|T_i\mathbf{v}_i - \mathbf{v}'_i\|^2 + \sum_{j\in\mathcal{N}_i} \|T_i\mathbf{v}_j - \mathbf{v}'_j\|^2 \right). \qquad (8)$$

This is a quadratic expression, so the minimizer is a linear function of $V'$.

Note that this is not significantly slower than computing the solution for the initial local identity transformations: The system matrix $A$ has to be factored once, from the first solution all $T_i$ are computed, $b$ is modified accordingly, and the final positions $V'$ are computed using back-substitution.

### 4.3. Linearized implicit transformations

The main idea of [SLCO*04] is to compute an appropriate transformation $T_i$ for each vertex $i$ based on the eventual new configuration of vertices $V'$. Thus, $T_i(V')$ is a function of $V'$.

Note that in Eq. 7 both the $T_i$ and the $V'$ are unknown. However, if the coefficients of $T_i$ are a linear function in $V'$, then solving for $V'$ implies finding $T_i$ (though not explicitly) since Eq. 7 is still a quadratic function in $V'$. If we define $T_i$ as in Eq. 8, it is a linear function in $V'$, as required.

However, if $T_i$ is unconstrained, the natural minimizer is a membrane solution, and all geometric detail is lost. Thus, $T_i$ needs to be constrained in a reasonable way. We have found that $T_i$ should include rotations, isotropic scales, and translations. In particular, we want to disallow anisotropic scales (or shears), as they would allow removing the normal component from Laplacian representation.

The transformation should be a linear function in the target configuration but constrained to isotropic scales and rotations. The class of matrices representing isotropic scales and rotation can be written as $T = s\exp(H)$, where $H$ is a skew-symmetric matrix. In 3D, skew symmetric matrices emulate a cross product with a vector, i.e. $H\mathbf{x} = \mathbf{h}\times\mathbf{x}$. Drawing upon several other properties of $3\times 3$ skew matrices (see Appendix A), one can derive the following representation of

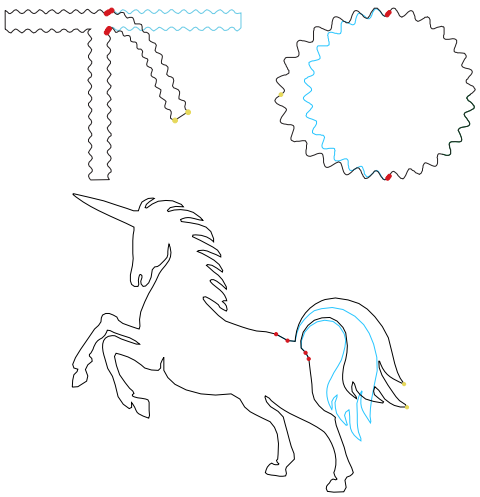


**Figure 3:** *Editing 2D meshes using Laplacian-coordinates fitting. The red dots denote fixed anchor points and the yellow are the pulled handle vertices. The original meshes are colored blue.*

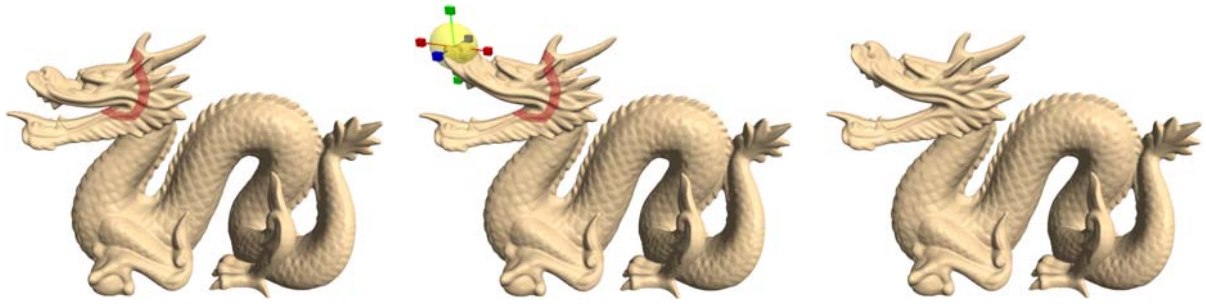
the exponential above:

$$s\exp H = s(\alpha I + \beta H + \gamma\mathbf{h}^T\mathbf{h}) \qquad (9)$$

Inspecting the terms we find that only $s$, $I$, and $H$ are linear in the unknowns $s$ and $\mathbf{h}$, while $\mathbf{h}^T\mathbf{h}$ is quadratic†. As a linear approximation of the class of constrained transforma-

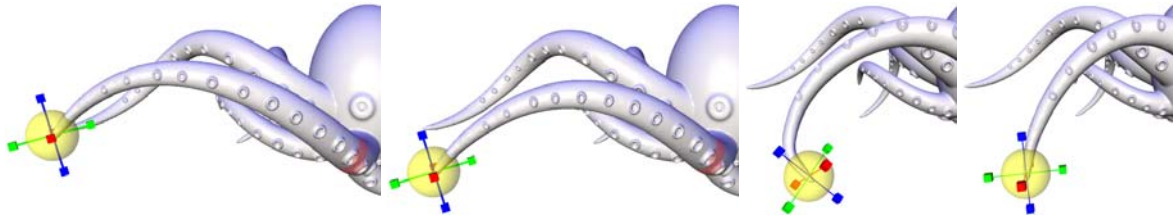
† Figure 3 illustrates editing of a 2D mesh. Note that in 2D the matrices of class $s\exp(H)$ can be completely characterized with the linear expression

$$T_i = \begin{pmatrix} a & w & t_x \\ -w & a & t_y \\ 0 & 0 & 1 \end{pmatrix}.$$



(a) (b) (c)

**Figure 4:** *The editing process. (a) The user selects the region of interest – the upper lip of the dragon, bounded by the belt of stationary anchors (in red). (b) The chosen handle (enclosed by the yellow sphere) is manipulated by the user: translated and rotated. (c) The editing result.*



(a) (b) (c) (d)

**Figure 5:** *Different handle manipulations. (a) The region of interest (arm), bounded by the belt of stationary anchors, and the handle. (b) Translation of the handle. (c), (d) Rotation of the handle. Note that the detail is preserved in all the manipulations.*

tions we, therefore, use

$$T_i = \begin{pmatrix} s & h_1 & -h_2 & t_x \\ -h_1 & s & h_3 & t_y \\ h_2 & -h_3 & s & t_y \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{10}$$

This matrix is a good linear approximation for rotations with small angles.

Given the matrix $T_i$ as in Eq. 10, we can write down the linear dependency (cf. Eq. 8) of $T_i$ on $V'$ explicitly. Let $(s_i, \mathbf{h}_i, \mathbf{t}_i)^T$ be the vector of the unknowns in $T_i$, then we wish to minimize

$$\|A_i(s_i, \mathbf{h}_i, \mathbf{t}_i)^T - \mathbf{b}_i\|^2, \tag{11}$$

where $A_i$ contains the positions of $\mathbf{v}_i$ and its neighbors and $\mathbf{b}_i$ contains the position of $\mathbf{v}'_i$ and its neighbors. The structure of $(s_i, \mathbf{h}_i, \mathbf{t}_i)^T$ yields

$$A_i = \begin{pmatrix} v_{k_x} & v_{k_y} & -v_{k_z} & 0 & 1 & 0 & 0 \\ v_{k_y} & -v_{k_x} & 0 & v_{k_z} & 0 & 1 & 0 \\ v_{k_z} & 0 & v_{k_x} & -v_{k_y} & 0 & 0 & 1 \\ \vdots & & & & & & \end{pmatrix}, k \in \{i\} \cup \mathcal{N}_i, \tag{12}$$

and

$$\mathbf{b_i} = \begin{pmatrix} v'_{k_x} \\ v'_{k_y} \\ v'_{k_z} \\ \vdots \end{pmatrix}, k \in \{i\} \cup \mathcal{N}_i. \tag{13}$$

The linear least squares problem above is solved by

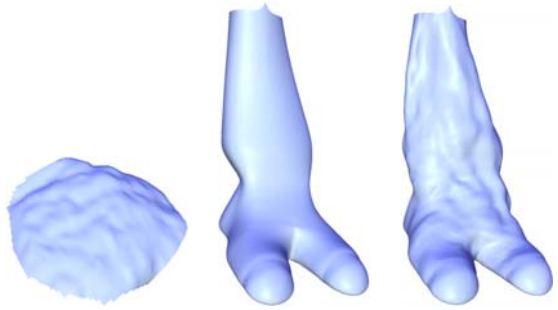
$$(s_i, \mathbf{h}_i, \mathbf{t}_i)^T = \left(A_i^T A_i\right)^{-1} A_i^T \mathbf{b}_i, \tag{14}$$

which shows that the coefficients of $T_i$ are linear functions of $\mathbf{b}_i$, since $A_i$ is known from the initial mesh $V$. The entries of $\mathbf{b}_i$ are simply entries of $V'$ so that $(s_i, \mathbf{h}_i, \mathbf{t}_i)$ and, thus, $T_i$ is a linear function in $V'$, as required.

### 4.4. Adjusting $T_i$

In many modeling situations solving for absolute coordinates in the way explained above is sufficient. However, there are exceptions that might require adjusting the transformations.

A good way of updating transformations for all three mentioned approaches is this: The current set of transformations $\{T_i\}$ is computed from $V$ and $V'$. Then each $T_i$ is inspected, the corresponding Laplacian coordinate $\delta_i$ is updated appro-



(a) (b) (c)

**Figure 6:** *Detail transfer; The details of the Bunny (a) are transferred onto the mammal's leg (b) to yield (c).*

(a) (b) (c) (d)

**Figure 7:** *The details of the* Max Planck *are transferred onto the* Mannequin. *Different levels of smoothing were applied to the* Max Planck *model to peel the details, yielding the results in (c) and (d).*

priately depending on the effect to be achieved, and the system is solved again. For example, if anisotropic scaling has been suppressed but is wanted, the $\{\delta_i\}$ are scaled by the inverse of the anisotropic scale implied by the constraints.

## 5. Mesh Editing

There are many different tools to manipulate an existing mesh. Perhaps the simplest form consists of manipulating a *handle*, which is a set of vertices that can be moved, rotated and scaled by the user. The manipulation of the handle is propagated to the shape such that the modification is intuitive and resembles the outcome of manipulating an object made of some physical soft material. This can be generalized to a free-form deformation tool which transforms a small set of control points defining a complex of possibly weighted handles, enabling mimicking other modeling metaphors (see e.g., [BK03] and the references therein).

The editing interaction is comprised of the following stages: First, the user defines the region of interest (ROI) for editing. Next, the handle is defined. In addition, the user can optionally define the amount of "padding" of the ROI by *stationary anchors*. These stationary anchors form a *belt* that supports the transition between the ROI and the untouched part of the mesh. Then, the user manipulates the handle, and the surface is reconstructed with respect to the relocation of the handle and displayed.

The submesh of the ROI is the only part considered during the editing process. The positions of the handle vertices and the stationary anchors constrain the reconstruction and hence the shape of the resulting surface. The handle is the means of user control, therefore its constraints are constantly updated. The unconstrained vertices of the submesh are repeatedly reconstructed to follow the user interaction. The stationary anchors are responsible for the transition from the ROI to the fixed untouched part of the mesh, resulting in a soft transition between the submesh and stationary part of the mesh. Selecting the amount of these padding anchor vertices depends on the user's requirements, as mentioned above. We have observed in all our experiments that setting the radius of the "padding ring" to be up to 10% of the ROI radius gives satisfying results.

The reconstruction of the submesh requires solving linear least-squares system as described in Section 2. The method of building the system matrix (Eq. 14), including the computation of a sparse factorization, is relatively slow, but constructed only once when the ROI is selected. The user interaction with the handle requires solely updating the positions of the handle vertices in the right-hand-side vector, and solve.

Figures 4 and 5 illustrate the editing process. Note that the details on the surface are preserved, as one would intuitively expect. Figure 2 demonstrates deformation of a model with large extruding features which cannot be represented by a height field.

## 6. Detail Transfer

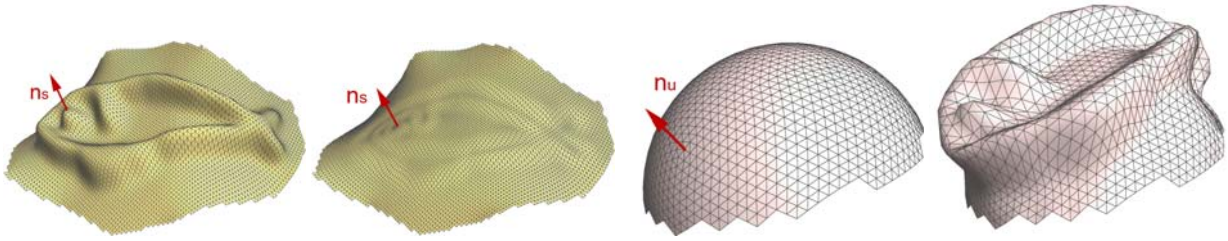
Detail transfer is the process of peeling the coating of a *source* surface and transferring it onto a *target* surface. See Figure 6 for an example of such operation.

Let $S$ be the source surface from which we would like to extract the details, and let $\tilde{S}$ be a smooth version of $S$. The surface $\tilde{S}$ is a low-frequency surface associated with $S$, which can be generated by filtering [DMSB99, FDCO03]. The amount of smoothing is a user-defined parameter, and it depends on the range of detail that the user wishes to transfer.

We encode the details of a surface based on the Laplacian representation. Let $\delta_i$ and $\tilde{\delta}_i$ be the Laplacian coordinates of the vertex $i$ in $S$ and $\tilde{S}$, respectively. We define $\xi_i$ to be the encoding of the detail at vertex $i$ defined by

$$\xi_i = \delta_i - \tilde{\delta}_i \,. \tag{15}$$

The values of $\xi_j$ encode the details of $S$, since given the




(a) (b) (c) (d)

**Figure 8:** *Detail transfer. The orientation of details (a) are defined by the normal at the corresponding vertex in the low frequency surface in (b). The transferred detail vector needs to be rotated to match the orientation of the corresponding point in (c) to reconstruct (d).*

bare surface $\tilde{S}$ we can recover the original details simply by adding $\xi_j$ to $\tilde{\delta}_i$ and reconstructing $S$ with the inverse Laplacian transform $L^{-1}$. That is,

$$S = L^{-1}(\tilde{\delta} + \xi) \,. \tag{16}$$

In this case of a detail transfer of $S$ onto itself, $S$ is faithfully reconstructed. However, in general, instead of coating $\tilde{S}$ with $\xi$, we would like to add the details $\xi$ onto an arbitrary surface $U$. If the target surface $U$ is not smooth, it can be smoothed first, and then the detail transfer is applied. In the following we assume that the target surface $U$ is smooth. Before we move on, we should note that the detail transfer from $S$ onto $\tilde{S}$ is simple, since the neighborhoods of the corresponding vertices $i$ have the same *orientation*. We define the orientation of a vertex $i$ in a surface $S$ by the normal direction of $i$ over $\tilde{S}$. Loosely speaking, the orientation of a point reflects the general orientation of its neighborhood, without respecting the high-frequencies of the surface.

When applying a detail transfer between two surfaces, the detail $\xi$ should be first aligned, or rotated with respect to the target. This compensates for the different local surface orientations of corresponding points in the source and target surfaces.

The following is an important property of the Laplacian coordinates:

$$R \cdot L^{-1}(\delta_j) = L^{-1}(R \cdot \delta_j) \,, \tag{17}$$

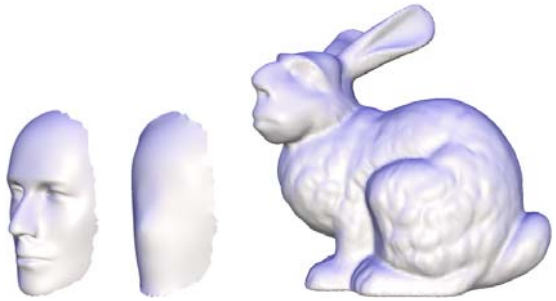
where $L^{-1}$ is the transformation from Laplacian coordinates to absolute coordinates, and $R$ a *global* rotation applied to the entire mesh. The mapping between corresponding points in $S$ and $U$ defines different local orientations across the surfaces. Thus, our key idea is to use the above property of the Laplacian coordinates locally, assuming that locally the rotations are similar.

Assume that the source surface $S$ and the target surface $U$ share the same connectivity, but different geometry, and that the correspondence between their vertices is given. In the following we generalize this to arbitrary surfaces.

The local rotation $R_i$ at each vertex $i$ in $S$ and $U$ is taken to be the local rotation between their corresponding orientations. Let $\mathbf{n}_s$ and $\mathbf{n}_u$ be the normals associated with the orientations of $i$ in $S$ and $U$, respectively. We define the rotation operator $R_i$ by setting the axis of rotation as $\mathbf{n}_s \times \mathbf{n}_u$ and requiring $\mathbf{n}_u = R_i(n_s)$. Denote the rotated detail encoding of vertex $i$ by $\xi_i' = R_i(\xi_i)$. Having all the $R_i$ associated with the $\xi_i$, the detail transfer from $S$ onto $U$ is expressed as follows:

$$U' = L^{-1}(\Delta + \xi') \tag{18}$$

where $\Delta$ denotes the Laplacian coordinates of the vertices of $U$. Now the new surface $U'$ has the details of $U$.

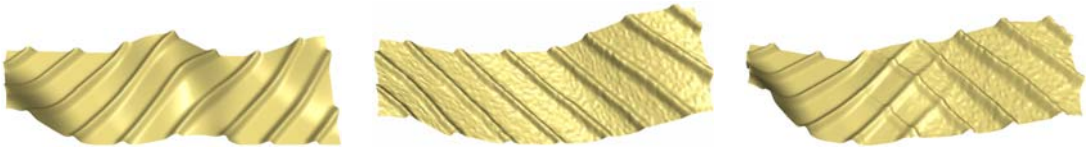


(a) (b) (c)

**Figure 9:** *Transferring the details of the* Mannequin *onto the face of the* Bunny. *(a) The source surface S. It is significantly smoothed to peel the details. (b) The smoothed surface $\tilde{S}$. (c) The result of detail transfer onto the* Bunny.

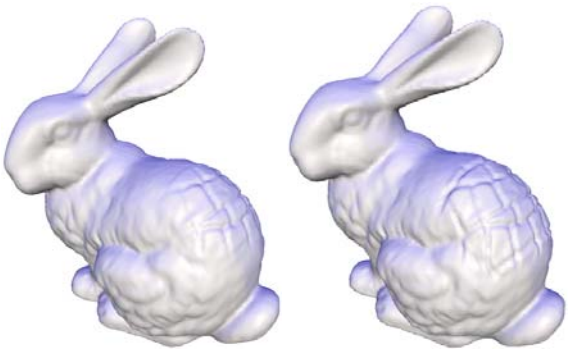
### 6.1. Mapping and Resampling

So far we assumed that the source and target meshes ($S$ and $U$) share the same connectivity, and hence the correspondence is readily given. However, detail transfer between arbitrary surfaces is more involved. To sample the Laplacian



(a) (b) (c)

**Figure 10:** *Mixing details using Laplacian coordinates. The Laplacian coordinates of surfaces in (a) and (b) are linearly blended in the middle to yield the shape in (c).*



(a) (b)

**Figure 11:** *Transplanting of* Armadillo*'s details onto the* Bunny *back with a soft transition (a) and a sharp transition (b) between the two types of details. The size of the transition area in which the Laplacians are blended is large in (a) and small in (b).*

coordinates, we need to define a mapping between the two surfaces.

This mapping is established by parameterizing the meshes over a common domain. Both patches are assumed to be homeomorphic to a disk, so we may chose either the unit circle or the unit square as common domain. We apply the mean-value coordinates parameterization [Flo03], as it efficiently produces a quasi-conformal mapping, which is guaranteed to be valid for convex domains. We fix the boundary conditions for the parameterization such that a correspondence between the source and target surfaces is achieved, i.e. we identify corresponding boundary vertices and fix them at the same domain points. In practice, this is a single vertex in $S$ and in $U$ that constrains rotation for the unit circle domain, or four boundary vertices for the unit square domain.

Some applications require a more careful correspondence than what can be achieved from choosing boundary conditions. For example, the mapping between two faces (see Figure 7) should link relevant details like facial features such as the brow wrinkles of the *Max Planck*. In this case the user provides a few additional (inner) point-to-point constraints which define a warp of the mean-value parameterization. In our implementation we use a radial basis function elastic warp, but any reasonable warping function can do.

In general, a vertex $i \in U$ is mapped to some arbitrary point inside a triangle $\tau \in S$. We experimented with several methods for sampling the Laplacian for a vertex. The best results are obtained by first mapping the 1-ring of $i$ onto $S$ using the parameterization, and then computing the Laplacian from this mapped 1-ring. Note that this approach assumes a locally similar distortion in the mapping. This is usually the case for the detail transfer; we used the 1-ring sampling in all the respective examples. We obtain similar results by linear interpolation of the three Laplacian coordinates sampled at the vertices of the triangle $\tau$. While this approach leads to some more "blurring" compared to the first one, it is even simpler and does not suffer from extremely different parametric distortion. In addition, no special treatment is required at the boundary of the domain in case the patch was initially cut to be homeomorphic to a disk.

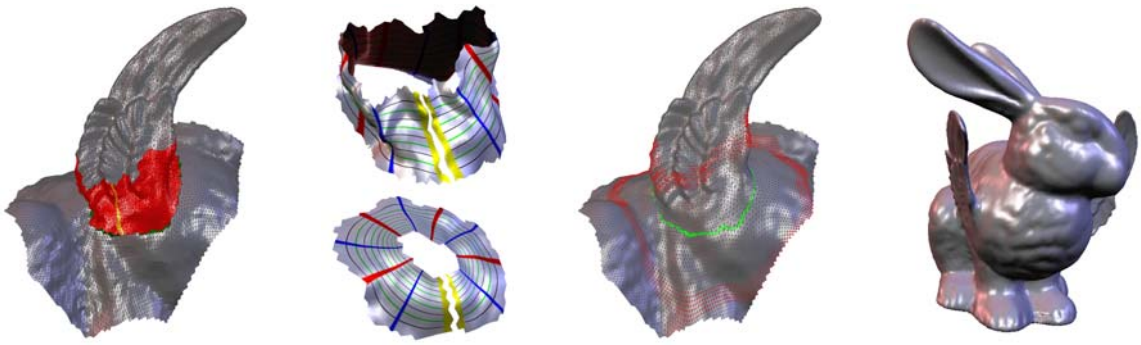
After the mapping between $U$ and $S$ has been established and the Laplacians have been sampled, the detail transfer proceeds as explained before. Note that now the corresponding $\xi_i$ is the difference between the *sampled* Laplacian coordinates in $S$ and $\tilde{S}$. See the examples in Figures 6, 7 and 9.

### 6.2. Mixing Details

Given two meshes with the same connectivity and different details, the above transfer mechanism can be applied on a third target mesh from the two sources. Figure 10 illustrates the effect of blending the details. This example emphasizes the mixing of details, as the details of the two source meshes differ in the smoothness, form and orientation. Note that the details are gradually mixed and the global shape of the target mesh is deformed respectively. By adding anchor points over the target, its shape can be further deformed. Figure 11 shows the application of this mechanism to transplant *Armadillo*'s details onto the *Bunny*'s back with a soft transition. In the next section we further discuss this transplanting operation.

## 7. Transplanting surface patches

In the previous sections we showed how the Laplacian coordinates allow to transfer the details of surface onto another



(a) (b) (c) (d)

**Figure 12:** *Transplanting of* Feline*'s wings onto the* Bunny. *(a) After cutting the parts and fixing the desired pose, the zipping (in green) defines the target connectivity D. The transitional region $D'$ is marked red. Additional cut in $D'$ (in yellow) enables mapping onto a square. (b) $D'$ is sampled over the respective regions $U' \subset U_\circ$ ($U_\circ$ is the cut part of the* Bunny*'s back) and $S'$ (the bottom of the wing). The texture with uv-isolines visualizes the mapping over the unit square. The cut (in yellow) aligns the two maps. (c) The result of reconstruction. The ROI is padded by a belt of anchors (in red). Note the change of the zipping seam triangles (green) and the details within the transition region. (d) The flying* Bunny *(see also Figure 1(d)).*

and how to gradually mix details of two surfaces. These techniques are refined to allow a seamless transplanting of one shape onto another. The transplanting operation consists of two apparently independent classes of operations: topological and geometrical. The topological operation creates one consistent triangulation from the connectivities of the two submeshes. The geometrical operation creates a gradual change of the geometrical structure of one shape into the other. The latter operation is based on the Laplacian coordinates and the reconstruction mechanism.

Let $S$ denote the mesh that is transplanted onto a surface $U$. See Figure 12, where the right wing ($S$) of the *Feline* is transplanted onto the *Bunny* ($U$). The transplanting requires the user to first register the two parts in world coordinates. This defines the desired location and orientation of the transplanted shape, as well as its scale.

The user selects a region $U_\circ \subset U$ onto which $S$ will be transplanted. In the rest of the process we only work with $U_\circ$ and do not alter the rest of $U$. $U_\circ$ is cut such that the remaining boundary is homeomorphic to the boundary of $S$. We simply project the boundary of $S$ onto $U_\circ$. The two boundary loops are zipped, thus creating the connectivity of the resulting mesh $D$ (Figure 12(a)).

The remaining transplanting algorithm is similar to detail transfer and mixing. The user specifies a region of interest on $D$, vertices outside the ROI remain fixed.

Next, the respective *transitional regions* $S' \subset S$ and $U' \subset U_\circ$ are selected starting from the cut boundaries on $S$ and $U_\circ$. Since $S' \subset D$, this implicitly defines the transitional region $D' \subset D$ along with a trivial mapping between vertices of $S'$ and $D'$.

For sampling, we require an additional correspondence between $S'$ and $U'$, hence we parameterize both meshes over the unit square. The user guides this construction by cutting $S'$ and $U'$ such that both meshes are homeomorphic to a disk. The cuts enable the mapping to the common domain, and in addition they serve as intuitive means to align the mappings such that there is a correspondence between the patches. In our experiments no further warping was necessary to improve the correspondence (cf. Section 6.1).

Once the transitional regions and the mappings are defined, the transplanting procedure is ready to sample the Laplacian coordinates of $S'$ and $U'$ over $D'$. The corresponding Laplacian coordinates are linearly blended with weights defined by their relative position in the unit square parameter domain. More precisely, if $v \in [0,1]$ defines the coordinate along the "height" axis (the blue and red lines in Figure 12(b), then the weights are $v$ and $(1-v)$, respectively. Since the length distortion of the maps may significantly differ, we linearly interpolate the Laplacian coordinates for sampling (cf. Section 6.1). The remainder of the ROI is sampled over $D$, and the reconstruction respects the belt of anchors which is placed to pad the boundaries of the ROI. Figures 12(c),(d) show the result.

## 8. Implementation details

All the techniques presented in this paper are implemented and tested on a Pentium 4 2.0 GHz computer. The main computational core of the surface reconstruction algorithm is solving a sparse linear least-squares problem. We use a direct solver which first computes a sparse triangular factorization of the normal equations and then finds the minimizer by back-substitution. As mentioned in Section 5, constructing the matrix of the least-squares system and factorizing it takes the bulk of the computation time. This might seem

as a heavy operation for such an application as interactive mesh editing; however, it is done only once per ROI selection. The solve by back-substitution is quite fast and enables to reconstruct the surface interactively, following the user's manipulations of the handle. It should be noted that the system is comprised only of the vertices that fall into the ROI; thus the complexity is not directly dependent on the size of the entire mesh, but rather on the size of the ROI. We experimented with various ROIs of sizes in the order of tens of thousands of vertices. The "intermediate preprocess" times observed were a few seconds, while the actual editing process runs at interactive framerates. Some short editing sessions are demonstrated in the accompanying video.

## 9. Conclusions

Intrinsic geometry representation for meshes fosters several local surface editing operations. Geometry is essentially encoded using differential properties of the surface, so that the local shape (or, surface detail) is preserved as much as possible given the constraints posed by the user. We show how to use this representation for interactive free-form deformations, detail transfer or mixing, and transplanting partial surface meshes.

It is interesting to compare the Laplacian-based approach to multi-resolution approaches: Because each vertex is represented individually as a Laplacian coordinate, the user can freely choose the editing region and model arbitrary boundary constraints, however, computing absolute coordinates requires the solution of a linear system. On the other hand, the non-local bases in multi-resolution representations limit the choice of the editing region and boundary constraints, but absolute coordinates are computed much simpler and faster by summing displacements through the hierarchy. Additionally, we would like to mention that we have found the Laplacian approach to be easier to implement and less brittle in practice.

In general, modeling geometry should be coupled to modeling other surface properties, such as textures. The machinery of discrete Poisson equations has already shown to be effective for image editing, so that editing textured surface should probably be performed on a combined differential geometry/texture representation.

## Acknowledgment

Models are courtesy of Stanford University and Max-Planck-Institut für Informatik.

## References

[Ale03a] ALEXA M.: Differential coordinates for local mesh morphing and deformation. *The Visual Computer 19*, 2 (2003), 105–114.

[Ale03b] ALEXA M.: Differential coordinates for local mesh morphing and deformation. *The Visual Computer 19*, 2 (2003), 105–114.

[BK03] BENDELS G. H., KLEIN R.: Mesh forging: editing of 3d-meshes using implicitly defined occluders. In *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing* (2003), pp. 207–217.

[BK04] BOTSCH M., KOBBELT L.: An intuitive framework for real-time freeform modeling. *ACM Trans. Graph. 23*, 3 (2004), 630–634.

[BMBZ02] BIERMANN H., MARTIN I., BERNARDINI F., ZORIN D.: Cut-and-paste editing of multiresolution surfaces. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (2002), pp. 312–321.

[Coq90] COQUILLART S.: Extended free-form deformation: A sculpturing tool for 3D geometric modeling. In *Proceedings of SIGGRAPH 90* (Aug. 1990), pp. 187–196.

[DMSB99] DESBRUN M., MEYER M., SCHRÖDER P., BARR A. H.: Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of ACM SIGGRAPH 99* (Aug.8–13 1999), pp. 317–324.

[FB88] FORSEY D., BARTELS R.: Hierarchical b-spline refinement. In *Proceedings of ACM SIGGRAPH 88* (1988), pp. 205–212.

[FDCO03] FLEISHMAN S., DRORI I., COHEN-OR D.: Bilateral mesh denoising. In *Proceedings of ACM SIGGRAPH 2003* (2003), pp. 950–953.

[Flo03] FLOATER M. S.: Mean-value coordinates. *Computer Aided Geometric Design 20* (2003), 19–27.

[FLW02] FATTAL R., LISCHINSKI D., WERMAN M.: Gradient domain high dynamic range compression. In *Proceedings of ACM SIGGRAPH 2002* (2002), pp. 249–256.

[GSS99] GUSKOV I., SWELDENS W., SCHRÖDER P.: Multiresolution signal processing for meshes. In *Proceedings of ACM SIGGRAPH 99* (Aug.8–13 1999), pp. 325–334.

[KCVS98] KOBBELT L., CAMPAGNA S., VORSATZ J., SEIDEL H.-P.: Interactive multi-resolution modeling on arbitrary meshes. In *Proceedings of ACM SIGGRAPH 98* (1998), pp. 105–114.

[KSMK99] KANAI T., SUZUKI H., MITANI J., KIMURA F.: Interactive mesh fusion based on local 3D metamorphosis. In *Graphics Interface '99* (June 1999), pp. 148–156.

[KVS99] KOBBELT L., VORSATZ J., SEIDEL H.-P.: Multiresolution hierarchies on unstructured triangle meshes. *Computational Geometry: Theory and Applications 14* (1999), 5–24.

[LSCOL04] LIPMAN Y., SORKINE O., COHEN-OR D., LEVIN D.: Differential coordinates for interactive mesh editing. In *International Conference on Shape Modeling and Applications 2004 (SMI'04)* (2004), pp. 181–190.

[PGB03] PÉREZ P., GANGNET M., BLAKE A.: Poisson image editing. In *Proceedings of ACM SIGGRAPH 2003* (2003), pp. 313–318.

[RIKM93] RANTA M., INUI M., KIMURA F., MÄNTYLÄ M.: Cut and paste based modeling with boundary features. In *SMA '93: Proceedings of the Second Symposium on Solid Modeling and Applications* (May 1993), pp. 303–312.

[SLCO*04] SORKINE O., LIPMAN Y., COHEN-OR D., ALEXA M., RÖSSL C., SEIDEL H.-P.: Laplacian surface editing. In *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry processing* (2004), Eurographics Association, pp. 179–188.

[SP86] SEDERBERG T. W., PARRY S. R.: Free-form deformation of solid geometric models. In *Proceedings of SIGGRAPH 86* (Aug. 1986), pp. 151–160.

[Tau95] TAUBIN G.: A signal processing approach to fair surface design. In *Proceedings of ACM SIGGRAPH 95* (August 1995), pp. 351–358.

[YZX*04] YU Y., ZHOU K., XU D., SHI X., BAO H., GUO B., SHUM H.-Y.: Mesh editing with poisson-based gradient field manipulation. *ACM Trans. Graph. 23*, 3 (2004), 644–651.

[ZSS97] ZORIN D., SCHRÖDER P., SWELDENS W.: Interactive multiresolution mesh editing. In *Proceedings of ACM SIGGRAPH 97* (1997), pp. 259–268.

**Appendix A:** Exponential of a 3×3 skew symmetric matrix

Let $\mathbf{h} \in \mathbb{R}^3$ be a vector and $H \in \mathbb{R}^{3\times3}$ be a skew symmetric matrix so that $H\mathbf{x} = \mathbf{h} \times \mathbf{x}, \forall \mathbf{x} \in \mathbb{R}^3$. We are interested in expressing the exponential of $H$ in terms of the coefficients of $H$, i.e. the elements of $\mathbf{h}$. The matrix exponential is computed using the series expansion

$$\exp H = I + \frac{1}{1!}H + \frac{1}{2!}H^2 + \frac{1}{3!}H^3 + \dots$$

The powers of skew symmetric matrices in three dimensions have particularly simple forms. For the square we find

$$H^2 = \begin{pmatrix} -h_2^2 - h_3^2 & h_1h_2 & h_1h_3 \\ h_1h_2 & -h_1^2 - h_3^2 & h_2h_3 \\ h_1h_3 & h_2h_3 & -h_1^2 - h_2^2 \end{pmatrix} = \mathbf{h}\mathbf{h}^T - \mathbf{h}^T\mathbf{h}\, I$$

and using this expression (together with the simple fact that $H\mathbf{h} = 0$) it follows by induction that

$$H^{2n} = (-\mathbf{h}^T\mathbf{h})^{n-1}\mathbf{h}\mathbf{h}^T + (-\mathbf{h}^T\mathbf{h})^n I$$

and

$$H^{2n-1} = (-\mathbf{h}^T\mathbf{h})^{n-1}H$$

for $n \in \mathbb{N}$. Thus, all powers of $H$ can be expressed as linear combinations of $I$, $H$, and $\mathbf{h}\mathbf{h}^T$, and, therefore,

$$\exp H = \alpha I + \beta H + \gamma \mathbf{h}\mathbf{h}^T$$

for appropriate factors $\alpha, \beta, \gamma$.

**Appendix B:** Implementation Details

For ease of re-implementation, we explicitly give the rows of the basic system matrix $\mathbf{A}$. The main complication results from the rotations, which are linearized and computed from the displacements of one-rings.

We focus on one vertex $\mathbf{v}_0 = (v_{0_x}, v_{0_y}, v_{0_z})$ and its Laplacian $\delta_0 = (\delta_{0_x}, \delta_{0_y}, \delta_{0_z})$, yielding three rows in the system matrix. The transformation $T$ adjusting $\delta_0$ minimizes the squared distances between corresponding vertices $\mathbf{V} = (\mathbf{v}_1, \mathbf{v}_2, \dots)$ in the one-ring of $\mathbf{v}$:

$$T = \begin{pmatrix} s & -h_3 & h_2 \\ h_3 & s & -h_1 \\ -h_2 & h_1 & s \end{pmatrix} \tag{19}$$

The coefficients are linear expression in the displaced coordinates $\mathbf{V}'$ (see [SLCO*04] for details on how to derive the coefficients)

$$\begin{aligned} s &= \sum_i s_{i_x} v'_{i_x} + s_{i_y} v'_{i_y} + s_{i_z} v'_{i_z} \\ &= \mathbf{s}_x \mathbf{v}'_x + \mathbf{s}_y \mathbf{v}'_y + \mathbf{s}_z \mathbf{v}'_z, \end{aligned} \tag{20}$$

where the abbreviations $\mathbf{v}'_{\{x,y,z\}}$ are the rows of $\mathbf{V}'$. Similar computations lead to the linear expressions for $h_1, h_2, h_3$ and coefficient vectors $\mathbf{h}$.

Now we can plug these expressions into the matrix $T$ and multiply with $\delta_0$ to find

$$T\delta_0 = \begin{pmatrix} \sum_{k\in\{x,y,z\}} \left(\delta_{0_x}\mathbf{s}_k - \delta_{0_y}\mathbf{h}_{3_k} + \delta_{0_z}\mathbf{h}_{2_k}\right)\mathbf{v}'_k \\ \sum_{k\in\{x,y,z\}} \left(\delta_{0_x}\mathbf{h}_{3_k} + \delta_{0_y}\mathbf{s}_k - \delta_{0_z}\mathbf{h}_{1_k}\right)\mathbf{v}'_k \\ \sum_{k\in\{x,y,z\}} \left(-\delta_{0_x}\mathbf{h}_{2_k} + \delta_{0_y}\mathbf{h}_{1_k} + \delta_{0_z}\mathbf{s}_k\right)\mathbf{v}'_k \end{pmatrix} \tag{21}$$

The constraint $T\delta_0 = \delta'_0 = \mathbf{v}'_0 - \sum_i w_i \mathbf{v}'_i$ results in three rows

of the system $AV = b$ of the form

$$\begin{aligned}
\sum_{k\in\{x,y,z\}} \left(\delta_{0_x}\mathbf{s}_k - \delta_{0_y}\mathbf{h}_{3_k} + \delta_{0_z}\mathbf{h}_{2_k})\mathbf{v}'_k\right) - \mathbf{v}'_{0_x}(1, w_1, w_2, \ldots) = 0 \\
\sum_{k\in\{x,y,z\}} \left(\delta_{0_x}\mathbf{s}_k - \delta_{0_y}\mathbf{h}_{3_k} + \delta_{0_z}\mathbf{h}_{2_k})\mathbf{v}'_k\right) - \mathbf{v}'_{0_y}(1, w_1, w_2, \ldots) = 0 \\
\sum_{k\in\{x,y,z\}} \left(\delta_{0_x}\mathbf{s}_k - \delta_{0_y}\mathbf{h}_{3_k} + \delta_{0_z}\mathbf{h}_{2_k})\mathbf{v}'_k\right) - \mathbf{v}'_{0_z}(1, w_1, w_2, \ldots) = 0
\end{aligned} \tag{22}$$

or explicitly for, e.g., $x$:

$$\begin{aligned}
&\mathbf{v}'_x\left((1, w_1, w_2, \ldots) - \delta_{0_x}\mathbf{s}_x - \delta_{0_y}\mathbf{h}_{3_x} + \delta_{0_z}\mathbf{h}_{2_x}\right) + \\
&\mathbf{v}'_y\left(\delta_{0_x}\mathbf{s}_y - \delta_{0_y}\mathbf{h}_{3_y} + \delta_{0_z}\mathbf{h}_{2_y}\right) + \\
&\mathbf{v}'_z\left(\delta_{0_x}\mathbf{s}_z - \delta_{0_y}\mathbf{h}_{3_z} + \delta_{0_z}\mathbf{h}_{2_z}\right) = 0.
\end{aligned} \tag{23}$$

We see that the basic system matrix essentially contains three block copies of the Laplace matrix on the main diagonal, one for each coordinate direction. The additional coefficients in the off-diagonal blocks link the coordinate directions to accommodate rotations.

# Volumetric Sculpting From Implicit Representations to Virtual Clay

Marie-Paule Cani

GRAVIR lab (IMAG-INRIA) and INP Grenoble

**Abstract**

*Making virtual modeling as easy and intuitive as real-clay manipulation is still an unsolved problem. Many artists still consider clay modeling as a simpler, more intuitive way to design complex shapes than numerical shape modeling. Providing a virtual clay metaphor would however facilitate shape design: it would enable copy/paste and undo operations, avoid unwanted drying of the clay and enable edition at any scale. Implicit surfaces, which provide a smooth volumetric representation and a straightforward modeling of topological changes are one of the most attractive solutions towards this goal.*

*This chapter reviews some of the recent advances on interactive sculpting with implicit surfaces. Our 3D sculpturing metaphor represents the sculpted shape as the iso-surface of a scalar field sampled in a grid. The user can deposit some material in space and iteratively refines it using tools to add or remove material, paint or smooth it. A mechanism for allowing local shape deformation, such as printing the shape of a tool is provided, as well as a simple solution to offer haptic feedback while editing the shape. The extension to multi-resolution sculpting, allowing interactive shape design with no restriction in the scale of geometric details is then presented. Lastly, we describe a real-time virtual clay model that mimics real clay by combining the previous shape representation with a layered physically-based model, enabling both local and global deformations. Although not claiming physical accuracy, this model exhibits several important features of clay, namely plasticity, mass conservation and surface tension. Edition through direct manipulation using video tracking of the user's hands is finally discussed.*

## 1. Introduction

Compared with standard tools for numerical shape modeling, real clay remains a very simple and intuitive way to create complex shapes: even children use clay at school. Many artists prefer expressing themselves with real materials instead of using a computer. In many modeling systems, the user cannot focus his attention on the shape being modeled but has to understand its mathematical, internal representation and perform indirect editing, e.g. by interacting with a control mesh instead of directly pushing or pulling the surface. In addition, making holes or connecting separated parts is not always possible.

If one could get the benefits of real clay in a computer-based modeling system, one could have the best of both worlds: virtual clay would neither dry nor crack; it could be mutated from softer to dryer states as needed; the artist could pause at any time and return to work without worrying about material changes in the interim. Furthermore, gravity would no longer be a problem, so shapes than cannot be made easily with real material would become possible. Finally, the advantages of any computer-based modeling tool would apply: the artist would be able to work at any scale and use any size of tool, simplifying the production of fine details as well as global features, and virtual modeling would allow copy/paste, undo, etc., as well as more clay-specific ideas such as temporarily removing a part of the model to ease the editing of hard-to-reach areas.

This chapter explores volumetric approaches aimed at making virtual modeling as easy and intuitive as real-clay manipulation. In this context, shape representation using implicit surfaces is very well suited. Handling topological changes is not the only advan-

tage of these surfaces: they also ensure a *correct* definition of a closed surface which always possesses a well defined interior and exterior.

Standard modeling with implicit surfaces is performed using control primitives called *skeletons* that generate a scalar field from which an iso-surface is extracted. These scalar fields can then be combined in various ways, the most basic one being summation. A drawback of this constructive approach is that the cost of field evaluation grows with the number of primitives. If used in a sculpting system in which each user action results in the creation of a new primitive, the field evaluation would quickly become prohibitive and forbid interactivity. Most previous work on interactive volumetric sculpting with implicit surfaces thus take a different approach: the field function is directly stored in the form of discrete values sampled on a 3D grid. Skeleton-based primitives can still be used for representing the user-controlled tools that modify this field.

After a quick review of related work, this chapter details a full solution for enabling quick shape prototyping with no limitation in space or scale of the modeled shape. We also cover the generation of haptic feedback. We finally describe an hybrid model that combined the previous representation with a layered physically-based model for capturing local and global deformations of real clay. More details on these models can be found in [FCG00, FCG02, BFCG04, DC04a, DC04b].

## 2. Related work

A full review of modeling techniques using implicit surfaces is beyond the scope of this tutorial. The interested reader will find a good overview in [BBB*97].

### 2.1. Volumetric sculpting

Interactive modeling based on discrete scalar field representation was first introduced in 1991 by T. Galyean and J. Hughes [GH91]. The field was stored on a regular 3d grid (*voxmap*). The tool used to edit the field was also discretized and particular attention was paid to prevent aliasing when the discrete tool was re-sampled into the field grid. Available tool actions included adding or removing *material*, and smoothing the surface through a convolution applied to the 3D field.

In 1995, S. Wang and A. Kaufman [WK95] extended the interaction to carving using tools deduced from a pre-generated $20^3$ volume raster or sawing (extruding) via curves drawn onto the screen.

The following year, R. Avila and L. Sobierajski [AS96a] used a force feedback articulated arm to command the tool in a similar context. The very rapid update rate required limited the tool size to 3-5 voxels.

### 2.2. Multi-resolution editing

In 1998, J. Bærentzen [Bær98] proposed an *octree-based* volume sculpting system. Used to accelerate ray-casting rendering, the octree was unfortunately static: the subdivision was limited to, and always reached a fixed *leaf* level, yielding a regular sampling solution very close to the grid used in [GH91]. Dynamic *leaf-node* management to preserve memory in regions of low details or to increase resolution in highly detailed regions was left as a future work.

A. Raviv and G. Elber [RE00] proposed a different hierarchical approach based on scalar tensor product of uniform trivariate B-Spline function. A collection of B-Spline patches with arbitrary position, orientation and size was used to represent the scalar field. The user could create patches and select an active patch to edit it with a tool modifying its scalar coefficients. An additional octree structure was used to sample the collection of patches and conduct a Marching Cubes to extract the iso-surface. The octree resolution was guided by the underlying patches size.

S. Frisken [FPRJ00, PF01] presented a resolution adaptive volumetric approach, named ADF which stands for Adaptively Sampled Distance Field. The basic idea was to use the euclidean distance to a given surface and adaptively sample it into a discrete scalar field. Field recomputations due to surface editing were either performed by starting at the bottom of the hierarchy and then performing iterative simplifications (bottom-up strategy) or by refining the discrete field where necessary (top-down strategy). In order to maintain interactivity, the update was performed in priority at the neighborhood of the surface. Then distance modifications propagated during idle moments.

### 2.3. Towards virtual clay

The volumetric models we just reviewed were restricted to simple operations such as adding material or carving it, but contrary to real clay, didi not allow local and global, volume preserving deformations. Cellular automata were used to mimic the deformations of clay [ATTY99, DAB03]. These models allow free-form modeling with volume conservation and topological changes. However, the user can't perform large-scale deformations such as bending the limbs of a clay model.

Physically-based simulation is another option to

model clay, viewed as a material that lies between plastic solids and viscous fluids. However, plastic-solids require the addition of specific mechanisms for handling topological changes [MQW01]. Non-realtime, particle-based models were used to represent viscous material [Ton91] somewhat similar to clay. Realtime performances were achieved using Eulerian fluid simulations [Sta99], but this approach requires the fluid to occupy the entire space and thus fails to describe clay, which is bounded by a time-varying surface.

## 3. Sculpting with implicit surfaces

This section describes the implementation of a mono-resolution sculpting system based on the simple idea of a discrete field function stored in a 3D grid. The sculpted object is an iso-surface of this field. We first discuss different kinds of tools and actions and then detail the data structures enabling efficient field storage with no limitation of the shape extend in space. A snapshot of the resulting sculpting system, first presented in [FCG00], is depicted in Figure 1.

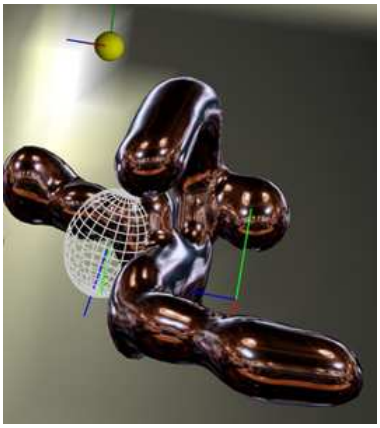


**Figure 1:** *Sample snapshot of our sculpting application. The object being modeled is environment mapped so that the user can better appreciate its shape. The sculpting tool is displayed in wireframe. The yellow spheres represent the lights that the user can move around while sculpting.*

In the remainder of this chapter, we take the convention that the field value corresponds to the *density of virtual clay*: the value is zero where there is no material and increases to a given maximal value inside the sculpted object. The surface of the latter is defined as an iso-surface of the field function. The user edits it, e.g. uses tools to add, remove or move material, by locally modifying the field function (stored as mentioned earlier, as a set of discrete values sampled in space).

### 3.1. Sculpting tools

The easiest way to set up tools for editing a scalar field is to define them as primitives generating scalar fields as well. A tool will act by adding or removing its field contribution to the discrete field function that defines the sculpted object. More precisely a tool is defined by:

- a **contribution**, i.e. a field function with local support, attached to the tool's local frame. The tool's bounding box bounds the region in space where the tool's contribution is non-zero. The tool's shape used for display is an iso-surface of the contribution.
- an **action**, that defines the way the tool's field is to be combined with the object's field (which may be zero or not in the region where the tool is applied).

#### Tool's contribution

We use two kinds of tools: analytical implicit primitives and discrete tools defined through sculpting using our application. In both cases, the tool's contribution is usually positive inside the tool and smoothly decreases to vanish at the border of a limited region of influence. This enables local control of the sculpted shape and save computational time.

The analytical primitives we have implemented are spheres or ellipsoids. We use Wyvill's field function [WMW86] to generate an isotropic (spherical) field around the tool center. We obtain general ellipsoids by scaling the tool along its three axes.

We also propose freeform tool shapes that are generated within our application. The shape displayed corresponds to the iso-surface, which is the same as the one visualized during its design process. The tool can also be scaled along the three axes of its local frame coordinate. Since applying such a discrete tool requires the evaluation of its field between its samples' location (see below), we define a continuous field for the tool using tri-linear interpolation.

#### Standard tool actions

The tool's actions listed below are similar to those presented in [GH91, AS96a]:

- deposit material, i.e. **add** the tool's contribution to the (possibly) existing field values that define the sculpted object.
- carve material, either smoothly by subtracting the tool's contribution to the object's field or unsmoothly by setting all field values under the tool's region of influence to zero.
- paint material by changing the color attributes stored together with the object's field value. Again, this can be done either smoothly or not, depending

on the way the tool's contribution is taken into account to modify or remove the previously existing color values.
- smooth the shape being modeled by applying a low-pass filtering of the the object's field function over the tool's region of influence.

**Local deformation tool**

Our aim here is to produce an intuitive local deformation, similar to the one a rigid tool would cause when interacting with clay, while avoiding the computational cost and stability problems of a physical simulation of the material displacements. Our method is inspired from an approach developed for standard skeleton-based implicit surfaces, which consists in applying a negative field to compress the object in the area where another object penetrates it, while creating a bulge by adding a positive field to imitate material displacement around the contact region [CD97].

We set the tool contribution to negative values inside the tool's iso-surface, in order to erase the material inside the tool; then the contribution becomes positive immediately outside the tool, where some material is to be added in order to compensate the loss of volume (combining these two actions imitates material displacement that occurs when a real tool collides with a block of clay). Finally the contribution vanishes to zero as we reach the border of the tools region of influence. In practice, the contribution is defined by combining a standard tools contribution (decreasing with the distance to the tools surface) with an analytical, smooth deformation function (refer to [FCG00] for details). Local deformation tools and the deformations they produced are depicted in Figures 2 and Figures 4.

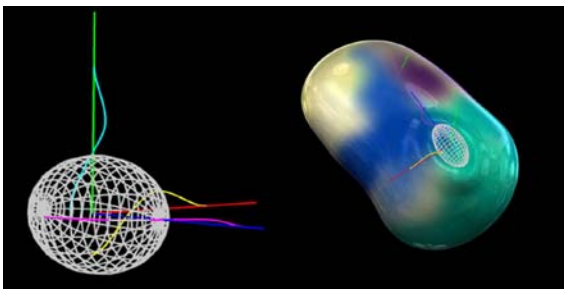


**Figure 2:** *(left) Local deformation tool (right) Imprint made by shifting an ellipsoidal tool onto an object.*

### 3.2. Representation of the discrete scalar field

Let us now discuss the data structures needed for efficiently representing the spatial samples of the field that defines the sculpted object. A straightforward solution consists in using a predefined, regular 3D grid. This is however very limitative: the grid then encloses the model, limiting its extension in space; moreover, it wastes memory by storing irrelevant sample points where no field is defined. We rather use dynamic data structures that only store the relevant voxels (intuitively, those where a field value is defined) of a virtual, infinite grid.

We call the regularly spaced points that sample the field `Vertices`. Each of them stores a field value between an arbitrary `minVal` and `maxVal` (arbitrarily set to 0 and 3 in our implementation, the iso-value at which the surface is defined being 1.5), a color and some cached data, such as the field gradient and the point location (this avoids its recomputation from the virtual grid indices). Each `Vertex` with a field value higher than `minVal` is stored in the `VertexTree`. Others are removed from the structure and deleted. Values above `maxVal` are clamped to `maxVal`. When requesting the value of a `Vertex` not defined, the returned value is `minVal`.

The regular space sampling we use divides space in cubical elements we call `Cells`. Each `Cell` having at least one `Vertex` defined is stored in a `CellsTree`. A `Cell` is made up of:

- eight pointers to its `Vertices`, one of which at least being non-null.
- an index deduced from the value of its eight `Vertices` relative to the iso-value, which encodes the `Cell`/iso-surface intersection configuration. This is a classical decomposition step from the Marching-Cells algorithm (see [Blo87, LC87, Blo88]).
- twelve pointers to edges.

The `Cells` that intersect the iso-surface (depending on their index value) are also inserted into another structure which we call `crossList`. To optimize surface display, `Edge` structures are created to compute and store an intersection of a Cells edge with the iso-surface. `Edges` are stored in an `EdgeTree`.

We tried two different implementations for the above dynamic data-structures (the VertexTree, the CellsTree, the crossList and the EdgeTree): balanced binary search trees and hash-tables. Our tests were in favor of the hash-tables implementation (see details in [FCG00]).

### 3.3. Applying a tool: data structures update

When a tool is applied, we have to flush its modification in the `VertexTree`. To this end, we first compute the axis-aligned bounding box surrounding the local tool bounding box. Then, we walk through this box by:

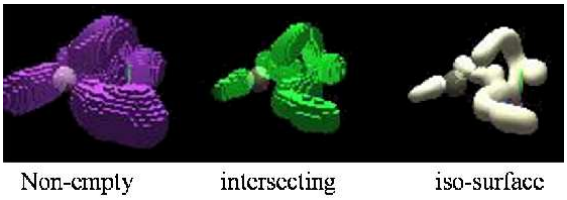



**Figure 3:** *Data structures for the field representation. Left: The Cell-Tree. Middle: the cross-list. Right: the sculpted surface*

1. transforming from world to local (tool) coordinate only the two extremal points of the box ($P_{min}$ and $P_{max}$) and the three displacement vectors (that move from one `Vertex` to the next in each axis direction).
2. starting from the $P_{min}$ point, we reach the next point simply by adding its displacement vector to its current location, and similarly adding its counterpart displacement vector to its counterpart location in local (tool) frame coordinate.

Note that any scaling can be applied to the tool by applying the inverse scaling to the local location we just obtained.

For each `Vertex` examined during this walkthrough, we distinguish three cases:

1. the `Vertex` is in the world bounding box, but outside of the local bounding box. It can be very quickly rejected, since the bounding box containment is a very rapid test in the local frame coordinate. We call these `Vertices` the **visited** `Vertices`.
2. the `Vertex` is inside the local bounding box, but outside the tool's influence (i.e. the tool's field has a null contribution at this point). To identify this case, we must compute the tool's field value for that point; we call them the **computed** `Vertices` (meaning that we computed the tool's field, but finally the `Vertex` wasn't modified).
3. the last category concerns the `Vertices` whose values were effectively modified. We call them the **dirtied** `Vertices` because they have to be updated (cleaned).

All the **dirtied** `Vertices` are inserted into a temporary tree called `modified`.

Each time a redisplay is needed, we successively extract (pop) every `Vertex` from the `modified` tree. For each `Vertex`, we then update the eight `Cells` that share it. We use a timestamp-mechanism to avoid multiple `Cell` examinations. Examining a `Cell` consists in computing its index, i.e. a bitmask deduced from its `Vertices` values relative to iso. If the `Cell` doesn't cross the iso-surface, we're finished with it. If it crosses the iso-surface, its index corresponds to a given surface crossing configuration stored in a pre-computed table (this is a standard step of the Marching Cubes process). This configuration tells us which `Edges` of the `Cell` are intersected. The corresponding `Edges` are then updated.

Creating or updating an `Edge` consists in (re-)computing the field gradients of its two `Vertices` (using for instance a central difference scheme). Then, the intersection point is obtained by linearly interpolating the `Vertex` attributes (such as the location, gradient and color) weighted by the corresponding potential field value stored. The interpolated gradient serves as surface normal.

### 3.4. Handling undo and redo

One key feature to encourage creative explorations is to allow multiple successive tries: the user can experiment whatever he desires without any consequence because he can always return to an earlier configuration.

We achieve the undo/redo process via temporary *undo-files*: each time a tool is applied, we dump all the modified `Vertices` into a new *undo-file*. In our implementation, dumping a `Vertex` corresponds to:

1. writing its indices in the virtual grid (i.e. the triplet $(i, j, k)$ relative to its current origin and step size.
2. writing its previous value and attributes (color only in our case, the other attributes such as the location and gradient are simply caches, and can be computed).
3. writing its new value and color after modification.

### 3.5. Results

Figure 4 shows an inprint inside a sculpted surface made by a local deformation tool sculpted within our application.

An example of complex object created with our sculpting system is depicted in Figure 5. Creating this shape required a number of trials and errors. It took three hours before the user was satisfied with the result. The main practical difficulty for the user was deciding, using a simple 2D display on a screen, whether the active tool was located in front of the sculpted surface or if it was intersecting it (which was desired for locally inflating or carving the surface). Several times during the edition, some material was added above the sculpture by mistake, so the undo mechanism proved very useful. The haptic feedback discussed next brings an effective solution to this positionning problem.

**Figure 4:** *Local deformation tool created by a tool sculpted using our system.*

**Figure 5:** *This sculpture was created using our volumetric sculpting system in about three hours, using a mere 2D mouse and 2D display on a screen.*

## 4. Force feedback

Like every artistic process, virtual sculpture requires a strong interaction between the artist and his artwork. Feeling the material being modeled enforces the metaphor of sculpting and the immersion of the user, making the creative activity easier. The need for haptic feedback is even stronger when the user visualizes his 3D sculpture on a standard screen: without force feedback, correctly positioning an editing tool with respect to the sculpture is difficult, since it may require changing the viewpoint several times to check the tool's position.

Fortunately, the incorporation of force feedback in a virtual sculpture system does not need to follow the strict constraints of physical accuracy. Indeed, there is no strong need for tactile realism in virtual sculpture, since the only aim is to increase the artist's ability to be creative. This freedom allows the use of expressive haptic rendering, enhancing certain aspects of the models being displayed.

This section proposes an effective solution to the incorporation of expressive haptic feedback in the volumetric sculpting system we just described, together with a simple solution for reducing the instability problems during the interaction. As our results show, our new haptic rendering improves interactivity and immersion, thus making the sculpting system far easier to use. This work was first described in [BFCG04]

### 4.1. Computing haptic forces

The haptic rendering was done with a *Phantom desktop* device, which is a 6DOF articulated arm able to render 3D force feedback [MS94]. Figure 3 shows the use of the *Phantom desktop* to model a character.

**Figure 6:** *A user modeling a character with the virtual sculpture sofware using 3D glasses a Phantom desktop device.*

The advantage of having a volumetric representation for defining the surface and its gradient is that interesting local information is available to compute force feedback. As Avila [AS96b, Avi98] showed, there is no need to make complex computations to calculate plausible forces. Our forces express in a simple way pseudo-physical properties: volumetric viscosity and surface contact.

#### Viscosity

Equation (1) shows how a friction force can be computed. This force tends to resist the movement proportionally to the material density and to the speed of the motion.

$$\vec{f_v} = -\alpha f_{v_0} \frac{V}{V_0} \dot{\vec{p}} \qquad (1)$$

The parameters in equation (1) are: $\alpha$, a positive constant dimensionally equivalent to the inverse of a speed; $f_{v_0}$, the friction intensity on the surface; $V_0$, the value of the potential defining the isosurface. $\dot{\vec{p}}$ is the speed at the point $\vec{p}$. $V$ is the value of the scalar

field function at the same point and $\vec{f_v}$ is the resulting volumetric viscosity force for this point and speed.

This force grows with the density of matter and the speed of the tool and is directed in the opposite direction of the movement. This reaction makes the user feel the volumetric property of his artwork by the resistance it opposes to the movement but it doesn't give any clue about the surface.

**Contact**

Equation 2 shows how the surface can be expressed in term of force feedback. This force is normal to the surface and grows rapidly when the tool enters the isosurface. The intensity of the force is clamped to ensure the safety of the simulation.

$$\vec{f_c} = -f_{c_0} \frac{\vec{grad}(V)}{\|\vec{grad}(V)\|} \left(\frac{V}{V_0}\right)^e \quad (2)$$

This force is locally equivalent to a spring model with stiffness $e$ if we consider the field function $V$ as a distance to the isosurface. This haptic feedback gives the user the ability to touch his artwork by feeling contact with the isosurface.

### 4.2. Expressive haptic rendering

With those two forces expressing volumetric and surfacic properties of the sculpture, it's possible to give the user a good feeling of his work [HQK98]. We extend this technique by using different combinations of the forces according to the user's intentions.

Changing the haptic representation of the object being manipulated according to user actions provides an expressive force feedback. This rendering adapts the simulation of the reality to the action of the artist, providing different feedback for the same object. This variability makes it non-realistic but enhances the interactive experience.

We achieve the goal of reinforcing the impact and the usability of the simulation by making it less realistic, in the same way non-photorealistic picture does for visual rendering. This is our notion of "non-tactorealistic" or "expressive" haptic rendering.

**Forces combination**

We found that the surfacic force is very useful when positioning the tool on the sculpture but can be disturbing when the user edits his work. If the tool can't enter inside the sculpture, carving an existing model is difficult. By attenuating the surfacic force when the user modifies his sculpture and enforcing the volumetric rendering, we reinforce the feeling of manipulating matter, not only a surface.

Thus, two combinations of the forces are used depending on the interaction mode of the user. Equation (3) is used when the user is passive and equation (4) when he's applying a tool. When the user is passive, the surfacic force dominates and when he's active, the volumetric force takes over, which can be expressed by: $\alpha_p > \beta_p$ and $\alpha_a < \beta_a$. The variation of each relative contribution is expressed by: $\alpha_p > \alpha_a$ and $\beta_p < \beta_a$.

$$\vec{f} = \alpha_p \vec{f_c} + \beta_p \vec{f_v} \quad (3)$$

$$or \ \alpha_a \vec{f_c} + \beta_a \vec{f_v} \quad (4)$$

The transition between the parameters is done smoothly to avoid discontinuities in the resulting force by using the equation (5) where $p$ varies continuously from 0 when the user is passive, to 1 after he has started to apply the tool, and from 1 to 0 for the opposite transition.

$$\vec{f} = (\alpha_p + p(\alpha_a - \alpha_p))\vec{f_c} + (\beta_p + p(\beta_a - \beta_p))\vec{f_v} \quad (5)$$

### 4.3. Stabilization of the haptic feedback

If the update of the force at $1kHz$ rate is not reached, there is a potential source of vibration in the system. This requirement is not an issue with our system because of the simplicity of the forces. However, a haptic simulation can't be stable in every condition because of the user being involved in the loop [GC96].

The original solution presented here is a particular case of virtual coupling introduced in [CSB95] without using complex linear circuit theory as in [AH99].

**Origin of the vibrations**

By its definition resulting of a gradient, the surfacic force tends to repulse the tool in an area where the magnitude of the force is smaller. The lag introduced by the user in its reaction makes him resist to a strong force when the tool is already outside the active area. Then, he doesn't meet a resistance and reenters the repulsing area. The repetition of this sequence causes the unexpected vibrations.

Filtering the force to make it vary smoothly is not a good solution because it doesn't guarantee that the position, resulting of the concomitant action of the user and the haptic feedback, will never jerk. Our solution is to filter the position coming from the device and to use this filtered position that can't vibrate, to compute the force feedback. As a side effect, this force is naturally smooth.

**Filtered position**

To avoid vibration, a damped position is computed using a low-pass filter that cuts the high spatial frequencies of the real position of the device. This filter is just an exponential damping having a time constant adapted to the vibration we want to cut. Equation (6) gives the definition of the damped position $\vec{p_d}$ in function of the real one $\vec{p_r}$, $\tau$ being the time constant of the filter.

$$\begin{aligned} \vec{\delta p} &= \vec{p_r} - \vec{p_d}(t-1) \\ \vec{p_d}(t) &\leftarrow \vec{p_d}(t-1) + \tau \vec{\delta p} \end{aligned} \tag{6}$$

Using this damped position eliminates the vibrations well. However, cutting the high frequencies of the movement introduce a lag that is noticeable in high amplitude movements. We can then use the fact that those movements, even at high speed, are not vibration but express the user's intention to really move to another area. The vibrations are then characterized by high frequencies and low amplitude.

So we compute (see equation (7)) a confidence $\gamma$ varying between 0 and 1 in the damped position depending on the distance between the real position and the damped one. If the two positions are close, meaning there is a potential vibration of low amplitude or that the tool doesn't move, the confidence in the damped position is 1; if the position is far away, the confidence tends to 0. The distant constant $\lambda$ characterizes the amplitude of movement we want to cut.

$$\gamma = \frac{1}{\|\vec{\delta p}\|/\lambda + 1} \tag{7}$$

A filtered position resulting from a combination of the real and damped one is then computed using this confidence. Equation (8) shows this filtered position $\vec{p_f}$ as a linear combination of $\vec{p_r}$ and $\vec{p_d}$.

$$\vec{p_f} = \gamma \vec{p_d} + (1-\gamma)\vec{p_r} \tag{8}$$

The continuous variation between the real and damped position makes it unnoticeable to the user and the surfacic force resulting can't be discontinuous.

**Spatial coherence**

An interesting property of the filtered position can be deduced from the precedent relations. Equation (8) directly implies equation (9); from (6) we can deduce (10) and then (11) can be deduced from (7).

Finally, we can deduce that the distance between the filtered position and the real one $\|\vec{p_f} - \vec{p_r}\|$ is always smaller than $\lambda$, the distance characterizing the confidence factor (equation (12)).

$$\begin{aligned} \|\vec{p_f} - \vec{p_r}\| &= \gamma \, \|\vec{p_d} - \vec{p_r}\| && (9) \\ &= \gamma \, \|\vec{\delta p}\| && (10) \\ &= \frac{\|\vec{\delta p}\|}{\|\vec{\delta p}\|/\lambda + 1} && (11) \\ &\leq \lambda && (12) \end{aligned}$$

This property ensures a spatial coherence between the real position and the filtered one used to display the tool and to compute the forces by guaranteeing they will never be distant from more than $\lambda$. This distance being of the same order than the magnitude of the vibrations, it's rather small and the user can't even notice the offset between the two. The guarantee expressed above ensures a good immersion of the user needed to make possible artistic work.

**Figure 7:** *This sculpture was achieved with haptic feedback within less than 1 hour, to compare to the bust in Figure 5 which took three hours to be created.*

## 5. Multi-resolution sculpting

This section presents an extension of the previous 3D sculpture system that enables interaction with a

sculpted object at any modeling scale, without having to be concerned with the underlying mathematical representations. It allows the user to model both fine and coarse features while maintaining interactive updates and display rates.

The modeled surface is still an iso-surface of a scalar-field. The new idea is to store the field in a hierarchical grid that dynamically subdivides or undivides itself according to the size of the tool being used. The extent of the structure, in terms of space and resolution, is fully dynamic; it is driven by the actions of the user and has no size limitation of maximum depth. This system allows for precise, interactive direct modeling through the addition and removal of material. As the underlying scalar field representation is completely hidden from the user, the use of the system is intuitive and gives the feeling of direct interaction with the sculpted surface. This work was first presented in [FCG02]

### 5.1. Hierarchical scalar field

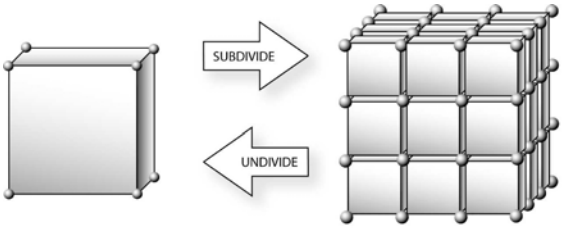



**Figure 8:** *Subdivision principle. Several choices are possible such as replacing the left Cell by the subdivided Cells on the right or referencing the subdivided Cells as children of the left Cell (i.e. enriching the left Cell), duplicating or not the Vertex elements that have the same position, ...*

We extend the idea of dynamic structures to store the field function dscribed in section 3.2 by providing a mechanism for locally refining the sampling rate. There are several ways to do so, as illustrated in Figure 8. In order to allow a multiresolution representation, the left cell isn't replaced by the sub-cells set, but is rather enriched with it: the sub-cells are its *successors* or *children*. As we do not want to restrict the user to any resolution limit (as well as we don't restrict the extent of the model in space), we allow the dynamic creation and deletion of successors sets. This precludes the use of classical octree storage optimizations. We rather reduce the structure overcost by allowing a direct jump to much finer resolutions. In practice, we recursively subdivide space by a constant factor $n$ in each dimension ($n = 3$ in Figure 8), leading to an *ntree* structure.

The next point to discuss is whether to:
(1) express the samples at a finer level, $k + 1$ as a *delta* contribution over the average or median value that would be stored at the coarser level $k$;
(2) store directly the field value in each sample, thus using simple subsampling for coarser levels.

Solution 1 appears more elegant, as it looks like a wavelet decomposition of the 3D scalar field. However, it yields the extra cost of maintaining the hierarchy coherency. Coarser levels would need to be updated when detailed modifications are conducted on smaller levels, in order to recompute the average or median field's values.

Solution 1 also suggests that large changes on the low-resolution levels could effortlessly be reflected on the higher ones: storing some min-max information along the hierarchy would allow rapid pruning of the volume parts that become completely outside or inside of the modeled shape. However, the hierarchy exploration from the root node to the leaves (which is also requested in solution 2) cannot be avoided, since the surface representation has to be updated.

With the subsampling approach of solution 2, there is no need to compute the interpolated values from the coarser levels: the field value at a vertex is directly given. Moreover it allows **no duplication of** the *Vertex* nodes between resolutions. In contrast, solution 1 would force the eight Vertices of the *Cell* at the left of Figure 8 to be distinct from their counterparts in the sub-cells on the right because the sub-cell values define a *delta* contribution over them. Lastly, solution 2 offers a kind of *vertical independence* over the hierarchy: each level is completely independent from its ancestors, and can thus be updated independently. Therefore, we have adopted solution 2.

When subdividing the grid (i.e. during *Cell* creation), we pay special care to share the *Vertex* nodes among common faces or edges between the adjacent *Cells* of the same level. Once these shared structures are wired, their forthcoming updates won't cost more than a time-stamp comparison to prevent useless computations.

### 5.2. Tool Guided Adaptive Subdivision

#### Applying a tool

The refinement of the hierarchical structure we just defined is tailored by the resolution of the tools the user applies during editing. Applying a small tool or a tool with sharp features will result in a local refinement of the structure. Since the user can then switch

to a coarser tool, special attention has to be paid to the efficiency of updates. In order to give an interactive feedback whatever the tool's size, tools are applied in an adaptive way, the grid being always updated from coarse to fine levels. This maintains interactive rates even for large tool-sizes. A dynamic Level-Of-Detail (LOD) mechanism ensures that the surface of the object is also displayed at interactive rates regardless of the zoom value: surface elements, generated and stored at each level of resolution, are displayed depending on their size on the screen. The system may switch to a coarser surface display during user actions, thus always insuring interactive visual feedback.

More precisely, applying the tool involves two important steps:
(1). we must ensure that the tool is correctly sampled, i.e. the cell resolution of the field representation must be small enough to capture the tool's features.
(2). for each covered vertex, we have to combine its current field value with the tool's contribution at that point, depending on the predefined tools action:

**ALGORITHM: Applying a Tool to a Cell**

```
Cell::apply(Tool t, Action a) {
    foreach Vertex v do { a.update(t,v); }
    if (I have no children) { checkSubdivide(t); }
    if (I have children) {
        foreach Cell child do { child.apply(t,a); }
    }
}
```

Now, how do we know if we need to subdivide a given cell (*checkSubdivide* test in the algorithm above)? Let's suppose that the tool has an ellipsoidal shape. We would like to obtain something like Figure 9, where the sampling rate increases (i.e. the cell size decreases) in regions where the tool has sharp features.

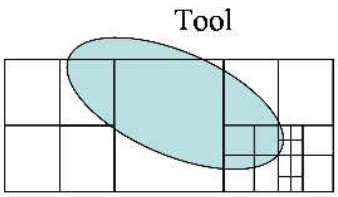



**Figure 9:** *Sampling an ellipsoidal tool. (a). Only 2 consecutive levels. (b). All the levels hierarchically created.*

First, we query the tool attributes for requirements on a minimal *security* cell-size to reach, in order not to miss any of its features. This information may be constant over the tools influence region, or locally computed. For example, if the tool field is stored as a hierarchy of cells, we use the size of the leaf cell as the minimal size to reach.

Once we have reached this minimal size, the field could still be ill-sampled. For example, if we use a eraser action, we might create field discontinuities, even with spherical tools. Our choice here is to try to estimate the *discrepancy* of the field. If the cell's discrepancy is higher than a given tolerance, we go on subdividing. Pragmatically, we use this strategy only on cells that are crossing the surface, as this is our region of interest. Moreover, using this strategy in regions where no surface exists could disturb the surface when it reaches these regions; as the details existing only in the field (and consequently hidden from the user) would suddenly become visible on the surface being created.

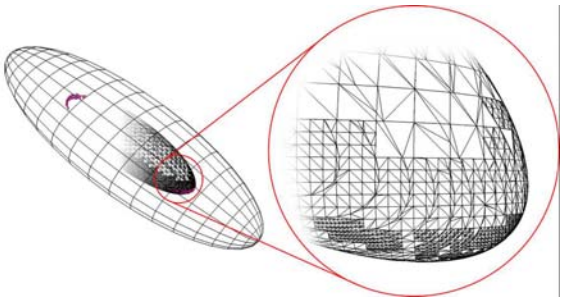


**Figure 10:** *Sampling an ellipsoidal tool: the large ellipsoid is the tool, and the small one inside it is the surface created. The figure shows the maximum resolution reached in highly curved areas.*

As stated above, using a subtractive tool can cause discontinuities in the scalar field so the subdivision process might never end. Here again we rely on the tool to query a *maximum depth* to reach. In fact, this is formulated as a smallest cell size not to overpass, which we call the *maximum resolution*. It could, exactly as the *minimum resolution* above, be locally adapted inside the tool's region of influence. At the moment, our ellipsoidal tools only expresses it as a constant factor, depending on the tool's scale used. This yields:

**ALGORITHM : Testing a cell for subdivision**

```
Cell::checkSubdivide(Tool t) {
    if (size > t.getMinResolution() ) {
        subdivide();
    } else {
```

```
        if (size < t.getMaxResolution() ) {
            if (estimateDiscrepancy() > acceptableDispcrepancy) {
                subdivide();
            }
        }
    }
}
```

We do not have *a priori* knowledge of the field's profile before we reach the bottom level of subdivision. Thus we cannot stop the subdivision to conduct any simplification ("undivide" of Figure 8) at any particular level, as we cannot guarantee that the finer level will not add surface details. As a result, we conduct a separate simplification pass over the whole sampled field at idle moments of the interaction. The simplification strategy we are currently using is rather drastic, as it destroys every *Cell* that does not (and whose children do not) cross the surface.

Updating the coarser levels first, as depicted in the **apply** algorithm, is crucial to provide an interactive visual feedback. This means careful initialization of the newly created *Vertex* set. The creation of the new vertices for the sub-cells requires interpolation of the field value **prior** to the current tool application, so that the tool's modifications can properly be added.

### 5.3. Priority Queue Based Field Update

The recursive approach for performing field updates, described in Section 5.2, isn't suitable for an interactive update. Actually, it walks along the hierarchy depth first, thus missing the requirement to completely update coarser levels before considering finer ones. Next, we explain how to get this feature, which is essential for achieving interacive display.

#### From coarser to finer levels

We use a priority queue sorted on the level addressed and on the tool/action concerned to ensure an update from coarser to finer levels. The tools/actions must be sequentially applied, but we should update the coarser levels first. Thus we perform a straightforward priority evaluation based on these two criteria.

The *Apply* procedure outlined in 5.2 is not altered much. It still updates its internal vertices and subdivides if needed. Then, instead of recursively calling apply on the existing children, it simply inserts a new element made up of the same *ToolCopy* and the next level.

#### Emptying the queue

To empty the priority queue we need to find all the cells of a given size (or level) that are *intersected* by the *ToolCopy* (which is much like an image of the *Tool* at the moment its application was posted). Without any additional structure, this would mean recursively walking through the cells hierarchy from the root-cell until we reach the cells having the desired size. To the cost of walking from the root-cell we must add the extra-cost of the intersection test with the tool for each cells of the intermediate levels.

To avoid these useless computations, we use a simple *cell-queue* with basic constant-time operations (pushback and popfront) to temporarily store the cells intersected by the tool from one level to the next. Cell-queues are indexed by a *ToolCopy* and the size of the cells it contains. They can be directly inserted/handled inside the manager priority queue, whose elements are then the cell-queues.
The *Apply* procedure of 5.2 is again slightly modified: it receives a cell-queue as an extra parameter. Children cells that intersect the tool are appended to this queue.

Another benefit of these cell-queues is that they allow **interruption of** the processing of a given level if any coarser level is inserted inside the manager. The interrupted cell-queue is simply re-inserted in the manager priority queue, and is properly handled from where it was suspended when the working task returns to it.

### 5.4. Surface creation and display

The surface of the sculpted object is still generated using a Marching Cubes algorithm. If a given cell, at any resolution, crosses the iso-value, we associate a *Surface Element* to it. This structure stores the Marching Cubes configuration index (an integer) and at most twelve pointers to some *Surface Points*, i.e. intersections of the iso-surface with the current Cell's edges. As a result, we obtain many approximations (*Level Of Detail*) of the iso-surface at each level of the cells hierarchy.

Additionally, the *Surface Element* is used to estimate the surface *discrepancy* introduced in Section 5.2. We need a quantity that indicates the *flatness* of the extracted surface. We decide to exploit the normals extracted at the surface points. If the normals are all pointing in a similar direction, the surface will be well represented by our linear approximation. On the contrary, if they have very different directions, our linear approximation is poor and the sampling rate should be increased to better match the underlying iso-surface. We use a straightforward estimator that

computes a kind of standard deviation of the surface normals.

The refinement process guided by the discrepancy estimator enables correct sampling of the field. However, at the leaf level of the hierarchy we obtain far too many triangles for any current graphic hardware to display interactively.

To address this problem, we compute for each cell an estimated projected size on the screen. It is estimated from the cell's size and the distance of the cell's center to the screen projection plane. Using this *projected size*, we can stop the hierarchy exploration when the projection of the current cell becomes too small. For example, if the projected size of a cell is smaller than a pixel, the triangles contained inside its children will be smaller, so we avoid visiting them and rather draw the surface element of the current cell.

This mechanism gives control over the number of displayed cells at each frame and dynamically selects a LOD dependent on the distance to the projection plane. We automatically adjust the *minimum projected size* from frame to frame to maintain a given framerate during user interaction. At the end of each frame, we measure the time spent from the previous frame end and we use the difference with the desired *display time* to weight the *growing factor* of the *minimum projected size*. Pragmatically, we used the third power of this difference to minimize its influence when the display time is near its goal, and emphasize it when it's far from it, keeping its sign. When the user is idle, the limit projected size is progressively reduced close to zero. So the fully detailed geometry can be rendered if the user waits sufficiently long; which will allow a non-interactive but accurate display during the session.

Contrary to other multiresolution iso-surface constructions, we pay no attention to the cracks that appear between adjacent cells of different sizes. A first reason for this choice is that the surface is always changing during the sculpting process. Another reason comes from the fact that we dynamically select, at each frame, where to stop in the cells hierarchy display. Reconnecting surface elements would force us to always track the neighboring cells. This would largely slow down the display rate, which is especially true in our unconstrained hierarchy (adjacent cells could be distant from more than one level of resolution). Moreover, as long as a sufficiently large number of polygons is displayed, the cracks can remain hardly visible (see Figure 11), it is thus *a posteriori* not worth the effort. An offline global polygonization is computed when the sculpted object needs to be exported.

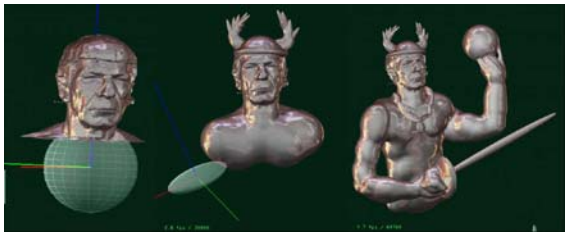


**Figure 11:** *Three steps of a character's modelling through the editing of an imported polygonal mesh. This example illustrate multi-resolution sculpting, since large tools were used for creating a smooth body for the character while very small ones were needed to create the wings of the helmet and the chain.*

## 6. Towards virtual clay

Although the previously described sculpting systems offer real-time interaction and a number of interesting features, the user action is mostly restricted to carving and adding material, since only a very restrictive kind of local deformation was defined. This fails to fully provide the intuitive interface for modeling we are looking for, since several some of the essential features of real clay — its ability to be globally bent or twisted, its constant volume during global and local deformations, its surface tension — are not simulated.

This section presents a volumetric, real-time virtual clay model which can be both sculpted by adding and removing material and deformed through the interaction with rigid tools. Either global or local, the deformations mimic the effects of the tools on real clay, due to plasticity, mass conservation, and surface tension. Our method enables the user to specify the local properties of the clay such as color and fluidity and allows the simultaneous use of an arbitrary number of tools. These contributions make this virtual clay model ready for direct hand manipulation, as be discussed in conclusion. This work was first presented in [DC04a, DC04b]

In this section, the clay surface is defined as the iso-surface at 0.5 of a scalar field modeling the clay density. Field values are stored in a 3D grid and clamped between 0 (an empty cell) and 1 (a cell full of clay).

### 6.1. A layered model for virtual clay

We are seeking for a model that, in addition to classical carving or addition of material, is able to capture local and global deformations expressed through clay displacement from a grid cell to another.

Rather than trying to be physically accurate, we use a layered physically-based model to simulate the

desired features of clay in real-time. The layers we use are:

1. **Large scale deformations:** This fist layer allows the user to bend or twists parts of the sculpted model using several rigid tools. The deformations are plastic: the clay will not come back to its initial state after the deformation is applied.
2. **Volume conservation:** The second layer prevents volume variation by iteratively poring clay in excess (i.e. clay in cells which density value exceeds 1 or which are occupied by a tool) into neighbouring cells. This results into intuitive folds and local bulges when the user deforms of presses the clay.
3. **Surface tension:** The third layer mimics surface tension by moving clay in cells where the density value is below 0.5 towards the surface of the sculpted model. As a result, clay does not spread into non-visible low-density regions, and the object remains compact anytime.

During simulation, the three layers are emulated in turn, over the same virtual clay representation (the 3D grid storing the density field). This yields a real-time model that exhibits the desired properties and is thus intuitive to use, as our results show. Figure 12 gives a schematic representation of the three layers.

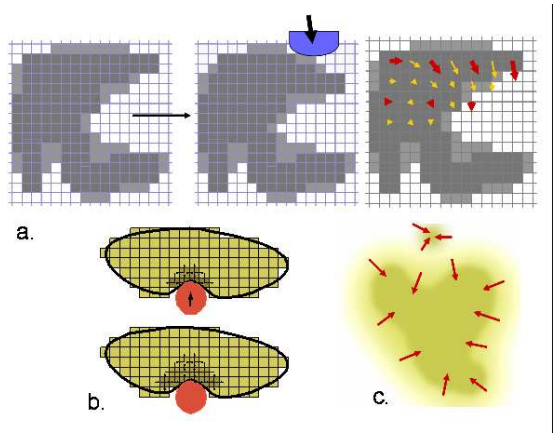



**Figure 12:** *A layered model for virtual clay. (a) large-scale deformations modeling plasticity. (b) local deformations insuring constant volume. (c) Surface tension avoiding the spreading of clay over space.*

### 6.2. Large scale deformation layer

We are seeking for the large scale, plastic deformation produced by the action of user-controled tools. Contrary to most physically-based models used in Computer animation, there is no need to use a dynamic model here: getting a static "equilibrium shape" after each user's action is sufficient.

This layer computes the displacement $\delta$ to apply to clay material lying in a given grid cell as a linear combination of the displacements dictated by the user-controled tools.

#### Combining the actions of multiple tools

Being able to interact by simultaneously using an arbitrary number of tools is an essential feature of our model: since we are trying to create a material close to real clay, sculpting with several tools, one of which may freeze a region of the clay to keep it still will be much more effective that using a single tool. It will even be mandatory to globally bend or twist the clay, operations we perform using our ten fingers in the real world.

Since the clay is a viscous fluid, the displacement of a user-controlled tool basically moves the clay around it the same way. The difficulty is to model the relative influence of the different tools on parts of the clay located in between.

The idea is to define regions of influence for each tool tool, in the spirit of voronoi regions but with a smooth transition between them. To achieve this, we compute as follows weight coefficients $k_i$ for each tool, that are used to compute the displacement $\delta$ of the clay in a cell as:

$$k_i = \frac{1 - \frac{d_i - min_j(d_j)}{min_j(d_{ij})}}{2} \quad \textbf{and} \quad \delta = \frac{\sum_i k_i \delta_i}{\sum_i k_i} \qquad (13)$$

where $j$ refers to all the other involved tools, $d_1$ is the pseudo-distance from the current cell t a given tool, and $d_{ij}$ is the pseudo-distance between two tools. We use a pseudo-distance instead of the Euclidian distance since the clay can be folded, so parts that are close in the 3D space can be far away inside the material.

More precisely, the pseudo-distance models the propagation of the quantity of movement inside a semi-fluid material. It can be seen as the length of the path, inside the object, along which the motion is transmitted. The longer this path is or the smaller the clay density is along it, the smaller the generated motion is. The pseudo-distance is computed through a propagation scheme that starts from a tool and propagates in the clay until it reaches its border or cells covered by other tools. For each non-empty cell $c_i$:

$$d_i = min_{neighbours}(d_j) + \frac{1}{\rho_i} \qquad (14)$$

where $\rho_i$ is the density of clay in $c_i$. This results into Voronoi-like regions of influence, with a continuously varying effect of a tool's motion between them.

**Rotating tools**

Up to this point, we only considered that tools motion would be translations when they are in contact with the clay. However, the user may also rotate tools, expecting to produce rotations or twists in the clay.

The motion of a solid rotating object cannot be described by a simple displacement vector. Instead, a point A rigidly linked to the tool moves according to displacement field:

$$\delta_A = \delta_O + OA \times \omega \tag{15}$$

where $\delta_O$ is the translation of point $O$ (the center of the tool) and $\omega$, the screw of the tool. To take into account the rotation of the tool, we simply replace the previous $\delta_i$ in equation (13) by the $\delta_A$ for cell $i$ in equation (15). This way, more general deformations can be generated. For instance, a bar of clay can be twisted by simply turning a tool at one end of the bar.

### 6.3. Mass-conservation layer

#### 6.3.1. Principles

The mass-conservation layer of the simulation aims at enforcing volume conservation. It also models local matter displacements near the surface of the object due to the tool's action. It will result in prints when the user pushes the tool on the object, in folds, etc. Of course, none of these effects can be produced by the previous layer. Indeed, the clay needs to locally move laterally and then even in the opposite direction from the tool to create bumps and folds around it.

The idea behind this layer is quite simple: if, in a cell, the density is greater than the maximum allowed value, 1, the excess is distributed into the six closest cells. When those cells are not full, the process terminates. If they have an excess of matter, they will distribute it among their own closest cells, and so on. Matter will move from cells to cells and finally reach the object's border, where it will find some room to remain. We will see that the object inflates in those areas.

We found the ideas behind this layer in fluid mechanics. When the medium sees locally an excess of pressure (i.e. an excess of matter), we get motions of the fluid from the areas with high pressures to areas with lower ones, until a uniform pressure is obtained. The main difference is that we only consider excess with regard to the maximum density and do not compare it to the surrounding values. This way, our clay remains solid, and doesn't tend to occupy the whole space.

#### 6.3.2. Interaction with tools

Now we need to see how tools can interact with our mass-conservation layer. We want the tool to push the clay in front of it when the user presses the tool against the object. The interaction is quite straightforward: where we have a tool, there's no more room for matter. The cells covered by the tool cannot contain clay anymore, so all the clay in those cells is in excess. We use the process we just described for moving this matter.

Rather than using purely rigid tools, we limit aliasing artifacts by defining them using a density function. The tool's density decreases near its edges. When the tool occupies eighty percent of a cell (i.e. its density value in the cell is 0.8), there is room for twenty percent of clay. Thus the carved object will have the same roughness as the tool. It is possible, too, to use a previously sculpted piece of clay as a tool. We thus let the user design his own complex tools, for example to be able to make prints or bas reliefs.

A small problem remains. If we simply move matter inside the tool to all close cells, some clay can go through the whole tool and exit on the other side. We thus add one more rule for interacting with tools: clay inside tools can only move outwards. For each cell occupied by the tool, we define allowed and forbidden directions among the six possible directions to neary cells. This way, tools really push matter in front of them, and no clay goes through the center of the tool.

For efficiency reasons, we precompute those allowed directions when we design a tool. This is done by looking for the closest direction to the surface of the tool. We *could* simply use the (discrete) gradient of the tool's field function. But this will not work for tools sculpted within our system, since we clamped the field value to 1 inside the tool. We need a second field function, with no clamping value this time, so that the gradient can be meaningfully computed anywhere. If we have only a field function already clamped to 1 to describe the tool, we have to build this second field function.

This can be done by using a propagation scheme starting from the edges of the tool, and going inside. We use the same algorithm we described in the large-scale displacement algorithm to compute the influence of the tools, except we got rid of the $1/density$ term. This way, we have everywhere an estimation of the distance to the surface of the tool, and the gradient points towards the outer part of the tool. This computation is performed each time we convert a piece of clay into a new tool.

Moving clay in one direction is allowed if this direction makes an angle with the direction of the gradient

under a given threshold. We choose to use a 60 degree angle. We normalize gradient, and we compare its components to 0.5 and -0.5 to decide whether motion along the $x-$, $y-$, and $z$-axes should be allowed.

### 6.4. Surface tension

After several deformations using the two layers above, the matter tends to become less and less compact. Clay pushed by the tools can indeed be dispersed around the object, and the transition from inside (density equal to 1) to outside (void cells) gets slower and slower. One of the problems with cells with low densities is that the user does not see them, so strange effects can arise if a tool pushes these small quantities of clay in front of it: clay popping from nowhere when density, due to action of the tool, rises to the threshold; innacurate changes of the surface location, etc. Moreover, since matter in cells of low density is no longer visible, the object's volume will seem to decrease, even if matter does not really disappear.

The surface-tension layer tries to resolve and avoid these problems. It keeps the gradient of density near the surface of the clay to an acceptable value. Matter in cells with very low densities is moved to nearby cells with higher densities. We look for every cell with a value below a threshold. At each such cell, we compute the gradient of the field function by using finite differences with nearby cells. If the length of the gradient is below another threshold (which correspond to the gradient we would like to have near the surface of the object), we move clay from the cell with a low density to closest cells with higher ones. This way, the object remains compact. The layer can be seen as adding a surface tension effect for a fluid.

While the previous layer prevents contraction of our clay, this layer tries to avoid expansion. This will separate the object in two different compact parts when the user tries to stretch it too far. Indeed, if you try to cut an object in two, the area between the two pieces will have a decrease of density. When the middle area density falls below our threshold, the matter is divided between the two parts, and the object is eventually cut in half.

Even with surface tension, some very small pieces of matter may separate from the main block of clay, like crumbs from a piece of toast; these are sets of a few neighboring cells with above-threshold densities. This is still physically correct, and the user should not be surprised, since these crumbs are visible. But because they can be distracting for the user, we get rid of them as soon as possible by removing them from the working space. If we want to preserve the volume of the object, we can put the matter removed this way back in the closest cell with high density, as if the crumb had been eaten up by the clay.

### 6.5. Local properties of clay

Adding local properties for clay, such as color or a locally varying fluidity parameter is made straightforwards by our volumetric representation: we just store the extra parameter values in each non-empty grid cell.. The main concern is how to adequately attach the local properties to the clay when it moves and deforms.

#### Updating local properties

A local property should be linked to the clay material in a cell rather than to its specific position in space. We thus have to update cell parameter values each time some clay moves due to the action of one of the layers. Although clay motion is modeled by increasing the density value in a destination cell while decreasing it in a source cell, the values of local properties only have to be updated in the destination cell, since the material's nature in the source cell remains the same.

Let $\rho_i$ represent the amount of clay being transported to the destination cell and $\nu_i$ be the vector of associated local properties. Let $\rho_j$ and $\nu_j$ respectively be the quantity of clay and its parameters already stored in the destination cell. Then the natural choice for computing the new values of local properties associated to the quantity of clay $\rho_i + \rho_j$ in the destination cell is the weighted average:

$$\nu_{destination} = \frac{\rho_j \nu_j + \rho_i \nu_i}{\rho_j + \rho_i} \qquad (16)$$

For instance, in the case of fluidity, $\nu$ represents the proportion of water in the clay. The new proportion is indeed the weighted average of the previous values.

#### Deforming non-homogeneous clay

Clay with locally varying fluidity is modelled by setting an extra parameter in each cell in order to store the proportion of water contained. In areas where the density of clay is low or where fluidity is high, the movement should propagate less from a cell to another. To achieve this, we still compute the pseudo-distance from a tool using the previous scheme, but we replace equation 14 by:

$$d_i = min_{neighbours}(d_j) + \frac{1}{(1 - f_i)\rho_i} \qquad (17)$$

so that the "distance" increases more quickly where density $\rho_i$ is low and fluidity $f$ is high.

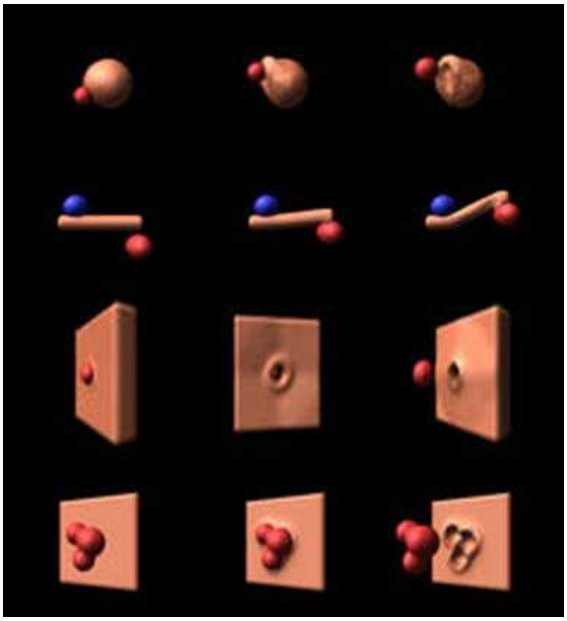


**Figure 13:** *Snapshops of the virtual clay sculpting system.*

### 6.6. Results

Figure 13 shows some snapshots or our virtual clay sculpting system. It shows that the model exhibits both global and local deformations and demonstrates its ability to convey topological changes (with the tool used to make a hole) while preserving a constant volume for the clay. For this first examples, only two tools were used, one of which keeping the whole or a part of the clay frozen to a fixed position.

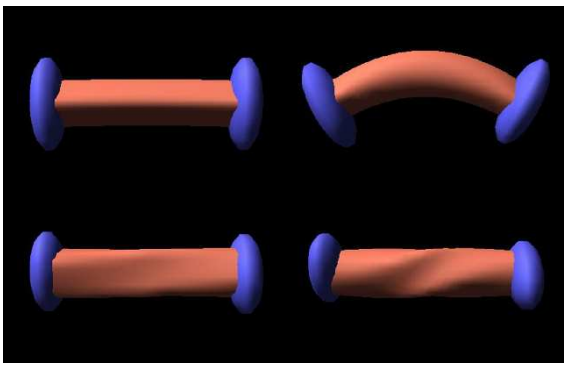


**Figure 14:** *Globally bending and twisting a piece of clay.*

Figure 14 shows the ability of the model to take into acound both translations and rotational motion of the tools in contact with the clay.

Figure 15 illustrates the fact that the clay model can be deformed by the combined motion of an arbitrary number of user-controlled tools, in the manner of the user's fingers of both hands used to deform real clay.

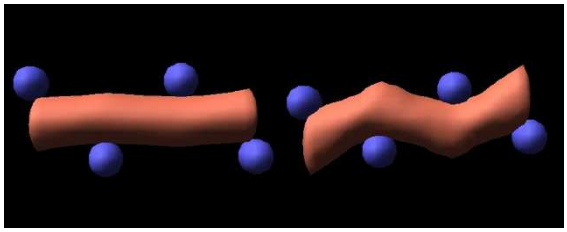


**Figure 15:** *Deforming a piece of virtual clay through the simultaneous action of four tools.*

## 7. Discussion and conclusion

We have presented volumetric sculpture systems that provides direct interaction with the model – an isosurface of a scalar field, making the internal representation of the sculpted surface totally transparent to the user, These systems allow shape editing at interactive rates.

One should however note that the closer we get towards a *virtual clay* model, the more attention we have to pay to user interaction: while the first sculpting systems which only allowed adding and removing material, was well suited to sculpting with a single tool (controled for instance via a 3D mouse or an haptic device), sculpting a complex shape with the last model we presented, much closer to real clay, cannot easily be done without defining an appropriate interface: in the real world, the user needs both hands to bend, twist or locally deform a piece of clay.

This last model is thus only a first step towards a very challenging long-term goal: enabling an artist to use his hands for modeling virtual clay, as he would do with a real material. Of course, an appropriate interface is needed to capture hands and fingers motion. The most obvious solution would be to use a glove, possibly with an exoskeleton for force feedback. We're currently studying a less invasive approach, using a vision-based interface. The motion of the user's hands, filmed by a few video cameras, will be used as an input to control virtual hands serving as multiple tools. However we have to come out with a real-time solution to hand tracking, wich is still an opened problem.

Haptic interaction proved to be a great aid in a sculpting process, since the user can "feel" the model.

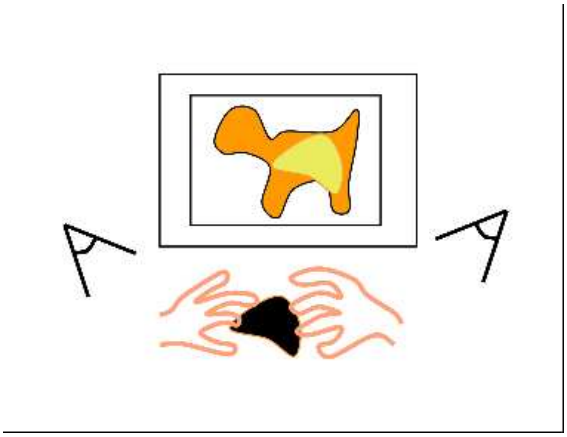


**Figure 16:** *Our concept for direct hand manipulation for virtul clay.*

In the first system we presented, it helped the user decide more easily if he was adding material onto or in front of the surface. However, haptic interaction through a force feedback device is still very far from the sense of touch a designer feels when he sculpts with real clay.

A long term goal would be to incorporate haptic feedback in the direct-hand manipulation interface we are seeking for the last model. Since we consider the use of haptic gloves as intrusive, we are thinking about rather using a "real-object interface": The user would manipulate a real deformable object (e.g. a ball full of sand), serving as an avatar for all or for a part of the sculpture (see Figure 16). His hand gestures would be captured by cameras and the reconstructed gestures be used to deform the virtual sculpture. We believe that such a interface providing a real sense of touch, even not exactly correlated with the display, would be a good advance towards making virtual sculpture as intuitive as the manipulation of real clay.

### Acknowledgments

Many thanks to Eric Ferley, Jean-Dominique Gascuel, Renaud Blanch and Guillaume Dewaele for their work on the volumetric sculpting systems presented in this chapter.

### References

[AH99] Adams R. J., Hannaford B.: Stable haptic interaction with virtual environments. *IEEE Transactions on Robotics and Automation 15*, 3 (1999), 465–474.

[AS96a] Avila R., Sobierajski L.: A haptic interaction method for volume visualization. *Computer Graphics* (Oct. 1996), 197–204. Proceedings of Visualization'96.

[AS96b] Avila R. S., Sobierajski L. M.: A haptic interaction method for volume visualization. In *IEEE Visualization '96* (1996), IEEE, pp. 197–204.

[ATTY99] Arata H., Takai Y., Takai N. K., Yamamoto T.: Free-form shape modeling by 3d cellular automata. In *International Conference on Shape Modeling and Applications* (1999), pp. 242–247.

[Avi98] Avila R. S.: Volume haptics. *Computer Graphics* (1998), 103–123. SIGGRAPH'98 Course Notes #01.

[Bær98] Bærentzen A.: Octree-based volume sculpting. *Presented at IEEE Visualization '98* (1998). `www.gk.dtu.dk/Andreas/publications.html`.

[BBB*97] Bloomenthal J., Bajaj C., Blinn J., Cani M.-P., Rockwood A., Wyvill B., Wyvill G.: *Introduction to Implicit Surfaces*. Morgan Kaufman, 1997.

[BFCG04] Blanch R., Ferley E., Cani M.-P., Gascuel J.-D.: *Non-Realistic Haptic Feedback for Virtual Sculpture*. Tech. Rep. RR-5090, INRIA, U.R. Rhone-Alpes, january 2004. Projet EVASION, theme 3.

[Blo87] Bloomenthal J.: Polygonization of implicit surfaces. *Xerox Technical Report CSL-87-2* (May 1987), 1–19.

[Blo88] Bloomenthal J.: Polygonization of implicit surfaces. *Computer Aided Geometric Design 5* (1988), 341–355.

[CD97] Cani M.-P., Desbrun M.: Animation of deformable models using implicit surfaces. *IEEE Transactions on Visualization and Computer Graphics 3*, 1 (Mar. 1997), 39–50. Published under the name Marie-Paule Cani-Gascuel.

[CSB95] Colgate J. E., Stanley M. C., Brown J. M.: Issues in the haptic display of tool use, 1995. IROS'95.

[DAB03] Druon S., A.Crosnier, Brigandat L.: Efficient cellular automata for 2d / 3d free-form modeling. *WSCG 11* (Feb. 2003).

[DC04a] Dewaele G., Cani M.-P.: Interactive global and local deformations for virtual clay. *Graphical Models 66* (sep 2004), 352–369. A preliminary version of this paper appeared in the proceedings of Pacifics Graphics'2003.

[DC04b] Dewaele G., Cani M.-P.: Virtual clay for direct hand manipulation. In *Eurographics '04 (short papers)* (2004).

[FCG00] Ferley E., Cani M.-P., Gascuel J.-D.: Practical volumetric sculpting. *the Visual Computer 16*, 8 (dec 2000), 469–480. A preliminary version of this paper appeared in Implicit Surfaces'99, Bordeaux, France, sept 1999.

[FCG02] Ferley E., Cani M.-P., Gascuel J.-D.: Resolution adaptive volume sculpting. *Graphical Models (GMOD) 63* (march 2002), 459–478. Special Issue on Volume Modelling.

[FPRJ00] Frisken S. F., Perry R. N., Rockwood A. P., Jones T. R.: Adaptively sampled distance fields: A general representation of shape for computer graphics. *Proceedings of SIGGRAPH 2000* (July 2000), 249–254. ISBN 1-58113-208-5.

[GC96] Gillespie R. B., Cutkosky M. R.: Stable user-specific haptic rendering of the virtual wall, 1996. Proceedings of the 1996 ASME International Mechanical Engineering Congress and Exhibition, DSC-Vol. 58.

[GH91] Galyean T., Hughes J.: Sculpting: An interactive volumetric modeling technique. *Computer Graphics 25*, 4 (July 1991), 267–274. Proceedings of SIGGRAPH'91 (Las Vegas, Nevada, July 1991).

[HQK98] Huang C., Qu H., Kaufman A.: Volume rendering with haptic interaction. In *Proceedings of the Third PHANTOM Users Group Workshop* (1998), vol. 3, pp. 14–18.

[LC87] Lorensen W., Cline H.: Marching cubes: a high resolution 3d surface construction algorithm. *Computer Graphics* (July 1987), 163–169. Proceedings of SIGGRAPH'87 (Anaheim).

[MQW01] McDonnell K. T., Qin H., Wlodarczyk R. A.: Virtual clay: A real-time sculpting system with haptic toolkits. *2001 ACM Symposium on Interactive 3D Graphics* (March 2001), 179–190. ISBN 1-58113-292-1.

[MS94] Massie T. H., Salisbury J. K.: The phantom haptic interface : A device for probing virtual objects, 1994. Proceedings of ASME'94.

[PF01] Perry R. N., Frisken S. F.: Kizamu: A system for sculpting digital characters. *Proceedings of SIGGRAPH 2001* (2001), 47–56.

[RE00] Raviv A., Elber G.: Three-dimensional freeform sculpting via zero sets of scalar trivariate functions. *Computer-Aided Design 32*, 8-9 (August 2000), 513–526. ISSN 0010-4485.

[Sta99] Stam J.: Stable fluids. In *Proceedings of SIGGRAPH 99* (Aug. 1999), Computer Graphics Proceedings, Annual Conference Series, pp. 121–128.

[Ton91] Tonnesen D.: Modeling liquids and solids using thermal particles. In *Graphics Interface'91* (Calgary, AL, June 1991), pp. 255–262.

[WK95] Wang S. W., Kaufman A. E.: Volume sculpting. *1995 Symposium on Interactive 3D Graphics* (April 1995), 151–156. ISBN 0-89791-736-7.

[WMW86] Wyvill B., McPheeters C., Wyvill G.: Data structure for soft objects. *Visual Computer 4*, 2 (Aug. 1986), 227–234.

# Sweepers & Swirling-Sweepers

Alexis Angelidis

University of Otago, New Zealand

**Abstract**

*Sweepers and swirling-sweepers are operations for modeling by space deformation. The artist describes a deformation as paths through which tools are moved. The movement of a tool causes a deformation of the working shape along the path of the tool. This is in accordance with a clay modeling metaphor, easy to understand and predict. It is desirable that deformations for modeling are 'foldover-free', that is parts of deformed space cannot overlap so that the deformations are reversible. Both sweepers and swirling-sweepers satisfy this criteria. In addition, swirling-sweepers preserve the shape's volume.*

## 1. Introduction

In Computer Graphics, in the context of interactive shape modeling, a common characteristic of popular techniques is the possibility for the artist to operate on a shape by modifying directly the shape's mathematical description. But with the constant increase of computing power, it is realistic and more effective to insert some interface between the artist and the mathematics describing a shape.

Space deformation is a family of techniques that permits describing operations on a shape independently from that shape's description. With this separation, new shape descriptions can easily benefit from existing space deformation, and further development can be carried in parallel. While space deformation has been used for solving a wide range of problems in Computer Graphics, they are missing a framework specific to interactive shape modeling. Sweepers is a framework for defining shape operations, in which the basis of operations is simply gesture.

Shapes produced with sweepers are coherent because sweepers are foldover-free: there is no ambiguity as to which points of space belong to the shape. A non foldover-free deformation would produce a self intersection of the shape, which cannot be cured with any space deformation. Sweepers were introduced in [AWC04], and are presented in Section 4.

In addition to *gesture as the basis of creation* and *shape coherency*, another concept familiar to most users is the *preservation of material*. A shape modeling technique that preserves volume would take even more advantage of user a priori knowledge of shapes. Swirling-sweepers is a technique that preserves a shape's volume independently from its description. In conjunction with any other sweeper operation, the volume of a shape can be increased, decreased or preserved. Swirling-sweepers were introduced in [ACWK04] and are presented in Section 5.

Sweeper and swirling-sweeper operations are independent from any shape description, since they are deformation of space. In order to visualize their effect on a shape, we propose in Section 6 a shape description which is suitable for the task of interactive shape modeling, and animation to some extent [AW04].

## 2. Modeling with deformation

A shape is the result of repeated deformation of the space in which the initial shape is embedded. A convenient formalism can be used for specifying any modeling operation by deformation: the *modeling equation* gives the final shape $S(t_n)$ as a function the initial shape $S(t_0)$:

$$S(t_n) = \{ \bigOmega_{i=0}^{n-1} f_{t_i \mapsto t_{i+1}}(\mathrm{p}) | \mathrm{p} \in S(t_0) \} \quad (1)$$

$$\text{where } \bigOmega_{i=0}^{n-1} f_{k_i \mapsto k_{i+1}}(\mathrm{p}) = f_{k_{n-1} \mapsto k_n} \circ \cdots \circ f_{k_0 \mapsto k_1}(\mathrm{p})$$

The operator $\Omega$ expresses the finite repeated composition of functions. Each function $f_{t_i \mapsto t_{i+1}} : \mathbb{R}^3 \mapsto \mathbb{R}^3$ is a deformation that transforms every point p of space at time $t_i$ into a point of space at time $t_{i+1}$. Sections 3, 4 and 5 will focus on defining functions $f_{t_i \mapsto t_{i+1}}$.

**Normal Deformation:** Computing accurate normals to the surface is very important, since the normals' level of quality will dramatically affect the visual quality of the shape. Let us recall that in order to compute the new normals, the previous normals are multiplied by the co-matrix[†] of the Jacobian [Bar84]. The Jacobian of $f$ at p is the matrix $J(f,\mathrm{p}) = (\frac{\partial f}{\partial x}(\mathrm{p}), \frac{\partial f}{\partial y}(\mathrm{p}), \frac{\partial f}{\partial z}(\mathrm{p}))$. Let us also recall that the following expression is a convenient way to compute the co-matrix of $J = (\vec{\mathrm{j}}_x, \vec{\mathrm{j}}_y, \vec{\mathrm{j}}_z)$, where the vectors $\vec{\mathrm{j}}_x$, $\vec{\mathrm{j}}_y$ and $\vec{\mathrm{j}}_z$ are column vectors:

$$J^C = \left(\vec{\mathrm{j}}_y \times \vec{\mathrm{j}}_z, \vec{\mathrm{j}}_z \times \vec{\mathrm{j}}_x, \vec{\mathrm{j}}_x \times \vec{\mathrm{j}}_y\right) \tag{2}$$

## 3. Related work

This section overviews several existing space deformation techniques, organized in three groups: axial deformations, lattice-based deformations and tool-based deformations. For the sake of clarity, we present existing space deformations aligned with the axes $\vec{\mathrm{e}}_x$, $\vec{\mathrm{e}}_y$ and $\vec{\mathrm{e}}_z$ and within the unit cube $[0,1]^3$, whenever possible. But a mere change of coordinates enables the artist to place the deformation anywhere in space. Note that affine transformations are the simplest case of space deformations.

### 3.1. Axial space deformations

Axial space deformations are a subset of space deformations whose control-points are geometrically connected along a curve. The curve may be initially straight or bent. To compare existing deformation techniques from the same point of view, we use $\vec{\mathrm{e}}_z$ as the common axis of deformation, which leads to slight reformulation in a few cases.

#### 3.1.1. Global and local deformations of solid primitives

A. Barr defines space tapering, twisting and bending via matrices whose components are functions of one space coordinate [Bar84]. We denote $(x,y,z)^\top$ the coordinates of a point. We show in Figures 1, 2, and 3 the effects of these operations, and we give their formula in the form of $4\times 4$ homogeneous matrices to be applied to the coordinates of every point in space to be deformed.

**3.1.1.1. Tapering operation:** The function $r$ is monotonic in an interval, and is constant outside that interval.

$$\begin{pmatrix} r(z) & 0 & 0 & 0 \\ 0 & r(z) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

**Figure 1:** *Taper deformation of a super-ellipsoid shape. A description of the shape can be found in [Gla89].*

[†] Matrix of the co-factors

**3.1.1.2. Twisting operation:** The function $\theta$ is monotonic in an interval, and is constant outside that interval.

$$\begin{pmatrix} \cos(\theta(z)) & -\sin(\theta(z)) & 0 & 0 \\ \sin(\theta(z)) & \cos(\theta(z)) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

**Figure 2:** *Twist deformation of a super-ellipsoid.*

**3.1.1.3. Bending operation:** This operation bends space along the axis $y$, in the $0 < z$ half-space. The desired radius of curvature is specified with $\rho$. The angle corresponding to $\rho$ is $\theta = \hat{z}/\rho$. The value of $\hat{z}$ is the value of $z$, clamped in the interval $[0, z_{\max}]$.

$$\begin{pmatrix} \cos\theta & 0 & \sin\theta & \rho - \rho\cos\theta - \hat{z}\sin\theta \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & \rho\sin\theta - \hat{z}\cos\theta \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

**Figure 3:** *Bend deformation of a super-ellipsoid.*

A. Barr observes that rendering the deformed shape with rays of light is equivalent to rendering the undeformed shape with curves of light. The curves of light are obtained by applying the inverse of the deformation to the rays. Because the deformation he proposes are not local, the portions of the rays to deform can be quite large.

#### 3.1.2. A generic implementation of axial procedural deformation techniques

C. Blanc extends A. Barr's work to mold, shear and pinch deformations [Bla94]. Her transformations use a function of one or two components. She calls this function the *shape function*. Examples are shown in Figures 4, 5, and 6.

$$\begin{pmatrix} r(z) & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

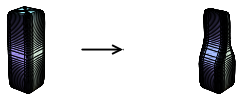


**Figure 4:** *Pinch deformation of a super-ellipsoid.*

$$\begin{pmatrix} r(\tan^{-1}(x,y)) & 0 & 0 & 0 \\ 0 & r(\tan^{-1}(x,y)) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

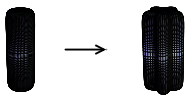


**Figure 5:** *Mold deformation of a super-ellipsoid.*

$$\begin{pmatrix} 1 & 0 & 0 & s(z) \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

**Figure 6:** *Shear deformation of a super-ellipsoid.*

### 3.1.3. A generalized de Casteljau approach to 3d free-form deformation

Y.K. Chang and A.P. Rockwood propose a polynomial deformation that efficiently achieves "Barr"-like deformations and more [CR94], using a Bézier curve with coordinate sets defined along $\vec{e}_z$ at the curve's control knots $(z_0, z_1 \ldots, z_n) \in [0,1]^{n+1}$. A reference straight segment, $z \in [0,1]$, is deformed by specification of coordinate sets $(c_i, \vec{u}_i, \vec{v}_i, \vec{w}_i)$ along that segment. The shape follows the deformation of the segment, as shown in Figure 7.

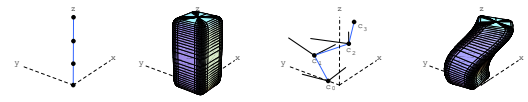



**Figure 7:** *Example of the deformation of Y.K. Chang and A.P.Rockwood applied to a super-ellipsoid. There is no need to define a pair of handles for the end control point.*

To compute the image q of a point p of the original shape, the matrix transforming a point to a local coordinate set is needed:

$$M_i = \begin{pmatrix} u_{i,x} & v_{i,x} & w_{i,x} & c_{i,x} \\ u_{i,y} & v_{i,y} & w_{i,y} & c_{i,y} \\ u_{i,z} & v_{i,z} & w_{i,z} & c_{i,z} \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (3)$$

where $\vec{w}_i = c_{i+1} - c_i$ , and $\vec{u}_i$, $\vec{v}_i$ are the handles.

Using this matrix, the deformation of a point is obtained recursively with the de Casteljau algorithm for evaluating a Bézier curve:

$$f_i^j(p) = (1 - p_z) f_i^{j-1}(p) + p_z f_{i+1}^{j-1}(p) \quad (4)$$

$$\text{where } f_i^0(p) = M_i \cdot p$$

The original generalized de Casteljau algorithm presented by Y.K. Chang and A.P. Rockwood is a recursion on affine transformations rather than on points. They remark that their recursion simplifies to the classic de Casteljau algorithm when the affine transformations are degenerate, and use the degenerate case in all their examples. As we show in Figure 8, this method is capable of performing "Barr"-like deformations and more.

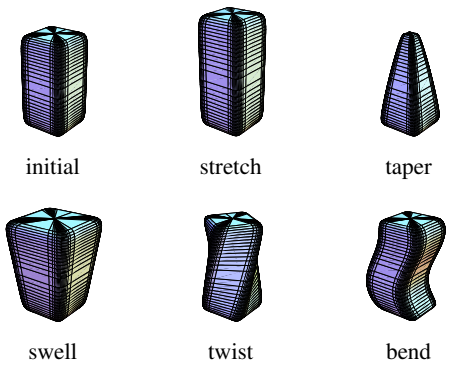



**Figure 8:** *Deformation of a super-ellipsoid.*

### 3.1.4. Axial deformation

The limitation of the methods presented so far is the initial rectilinear axis. If the shape is initially excessively bent, the manipulation of an initially straight control axis will not induce a predictable behavior of the shape. F. Lazarus et al. develop an extension of axial-based deformations using an initially curved axis [LCJ94]. Let us define a parametric curve $c(u)$. A point p in space is attached to local coordinates along the curve. The origin of this local coordinate system is $c(u_p)$, the closest point to p on the curve, and the axes are those of an extended Frenet frame that discards vanishing points [Blo90]. To find the closest point to p on curves, B. Crespin proposes an efficient algorithm based on subdivision [Cre99]. The axes are computed by propagating along the curve a frame defined at one extremity of the curve. The axes consist of three vectors: a tangent $\vec{t}(u)$, a normal $\vec{n}(u)$ and a binormal $\vec{b}(u)$. The propagated frame is computed as follows:

- the unit tangent at the origin is given by the equation of the curve:
  $\vec{t}(0) = \frac{dc(0)}{du} / \| \frac{dc(0)}{du} \|$.
- the normal and binormal are given by the Frenet frame, or can be any pair of unit vectors such that the initial frame is orthonormal.

To compute the next frame, a rotation matrix is needed. The purpose of this matrix is to minimize torsion along the curve. Numerous constructions of the rotation matrix require a sim-

ple formula:

$$R = \begin{pmatrix} a_{xx}+\theta & a_{xy}+b_z & a_{zx}-b_y \\ a_{xy}-b_z & a_{yy}+\theta & a_{yz}+b_x \\ a_{zx}+b_y & a_{yz}-b_x & a_{zz}+\theta \end{pmatrix} \quad (5)$$

$$\text{where} \quad (a_x,a_y,a_z)^\top = \frac{\vec{t}(u_i)\times\vec{t}(u_{i+1})}{\|\vec{t}(u_i)\times\vec{t}(u_{i+1})\|} \qquad \alpha = 1-\theta \quad (6)$$
$$\theta = \vec{t}(u_i)\cdot\vec{t}(u_{i+1}) \qquad \beta = \sqrt{1-\theta^2}$$

$$\begin{array}{lll} a_{xx} = \alpha a_x^2 & a_{xy} = \alpha a_x a_y & b_x = \beta a_x \\ a_{yy} = \alpha a_y^2 & a_{yz} = \alpha a_y a_z & b_y = \beta a_y \\ a_{zz} = \alpha a_z^2 & a_{zx} = \alpha a_z a_x & b_z = \beta a_z \end{array} \quad (7)$$

Given a frame at parameter $u_i$, the next axes of a frame at $u_{i+1}$ are computed as follows:

- the tangent is defined by the equation of the curve: $\vec{t}(u_{i+1}) = \frac{dc(u_{i+1})}{du} / \| \frac{dc(u_{i+1})}{du} \|$.
- the normal is given by the rotation of the previous normal: $\vec{n}(u_{i+1}) = R \cdot \vec{t}(u_i)$.
- the binormal is given by a cross product: $\vec{b}(u_{i+1}) = \vec{t}(u_i) \times \vec{n}(u_i)$.

The choice of the size of the step, $u_{i+1} - u_i$, depends on the trade-off between accuracy and speed. B. Crespin extends the axial deformation to surface deformation [Cre99].

### 3.1.5. Wires: a geometric deformation technique

K. Singh and E. Fiume introduce *wires*, a technique which can easily achieve a very rich set of deformations with curves as control features [SF98]. Their technique is inspired by armatures used by sculptors.

A wire is defined by a quadruple $(R, W, s, r)$: the reference curve R, the wire curve W, a scaling factor $s$ that controls bulging around the curve, and a radius of influence $r$. The set of reference curves describes the armature embedded in the initial shape, while the set of wire curves defines the new pose of the armature.

On a curve C, let $p_C$ denote the parameter value for which $C(p_C)$ is the closest point to p. Let us also denote $C'(p_C)$ the tangent vector at that parameter value.

The reference curve, R, generates a scalar field $F : \mathbb{R}^3 \mapsto [0,1]$. The function F which decreases with the distance to R, is equal to 1 along the curve and equals 0 outside a neighborhood of radius $r$. The algorithm to compute the image q of a point p influenced by a single deformation consists of three steps, illustrated in Figure 9:

- Scaling step. The scaling factor is modulated with F. The image of a point p after scaling is: $p_s = R(p_R) + (p - R(p_R))(1 + sF(p))$, where $p_R$ denotes the parameter value for which $R(p_R)$ is the closest to p.
- Rotation step. Let $\theta$ be the angle between the tangents $R'(p_R)$ and $W'(p_R)$. The point $p_s$ is rotated around axis $R'(p_R) \times W'(p_R)$ about center $R(p_R)$ by the modulated angle $\theta\, F(p)$. This results in point $p_r$
- Translation step. Finally, a translation is modulated to produce the image $p_{def} = p_r + (W(p_R) - R(p_R))$.

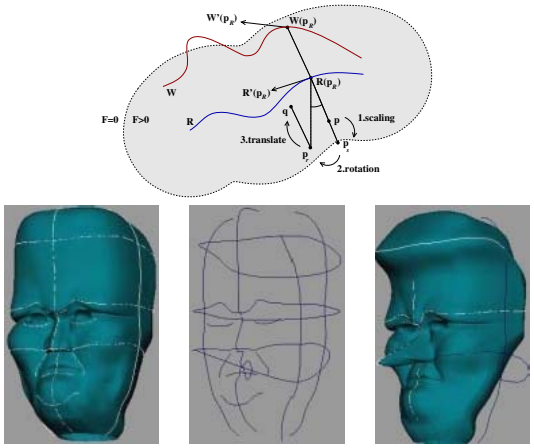



**Figure 9:** *Top: deformation of a point by a single wire. Bottom: deformation of a shape with multiple wires (courtesy of [SF98]).*

They propose different blending methods in the case when a point is subject to multiple wires. These methods work by taking weighted combinations of the individually deformed point. Let us denote $p_i$ the deformation of p by wire $i$. Let $\Delta p_i = p_i - p$. The simplest deformation is:

$$p_{def} = p + \frac{\sum_{i=1}^{n} \Delta p_i \|\Delta p_i\|^m}{\sum_{i=1}^{n} \|\Delta p_i\|^m}$$

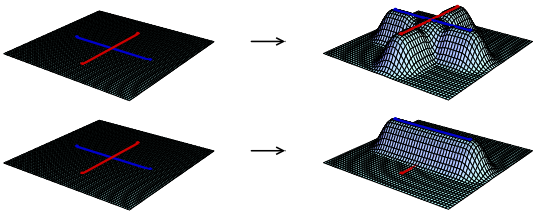



**Figure 10:** *Blending weights based on summed displacement magnitudes. This blending is not free from artifacts: notice the creases around the intersection in the upper-right figure.*

The scalar $m$ is defined by the artist. This expression is not defined when $m$ is negative and $\|\Delta p_i\|$ is zero. To fix this, they suggest to omit the wires for which this is the case. Their second solution is to use another blending defined for both positive and negative values of $m$:

$$\mathrm{p}_{def} = \mathrm{p} + \frac{\sum_{i=1}^{n} \Delta\mathrm{p}_i \prod_{j\neq i} \|\Delta\mathrm{p}_j\|^{|m|}}{\sum_{i=1}^{n} \prod_{j\neq i} \|\Delta\mathrm{p}_j\|^{|m|}}$$

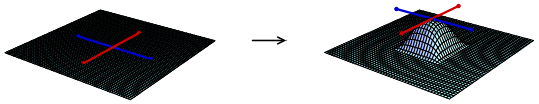



**Figure 11:** *Blending weights based on multiplied displacement magnitudes. The deformation is defined at the intersection of the reference curves.*

In order to use unmoved wires as anchors that hold the surface, they use $\mathrm{F}_i(\mathrm{p})$ instead of $\Delta\mathrm{p}_i$ as a measure of proximity:

$$\mathrm{p}_{def} = \mathrm{p} + \frac{\sum_{i=1}^{n} \Delta\mathrm{p}_i \mathrm{F}_i(\mathrm{p})^m}{\sum_{i=1}^{n} \mathrm{F}_i(\mathrm{p})^m}$$

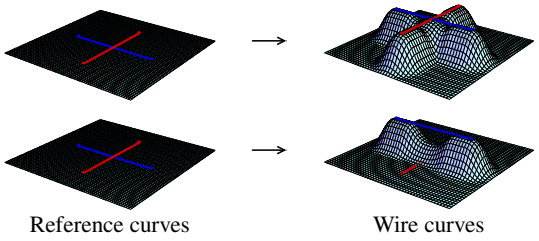



**Figure 12:** *Blending weights based on influence function. The unmoved wire holds space still. This blending is not free from artifacts: notice the creases around the intersection in the upper-right figure.*

Other capabilities of wires can be found in the original paper [SF98]. Note that the expensive part of the algorithm is computing the distance from each curve to each deformed surface point.

#### 3.1.6. Blendeforming: ray traceable localized foldover-free space deformation

As explained in the introduction, there are practical reasons for which a space deformation should be foldover-free. D. Mason and G. Wyvill introduce blendeforming [MW01]. A deformation is specified by moving a point or the control points of a curve along a constrained direction. Space follows the deformation of these control features in a predictable manner.

They define the blendeforming deformation as a bundle of non-intersecting streamlines. The streamlines are parallel, and described by a pair of functions: $b_{\mathrm{x,y}} : \mathbb{R}^2 \mapsto [-d_{\max}, d_{\max}]$ and $b_{\mathrm{z}} : [0,1] \mapsto [0,1]$. Function $b_{\mathrm{x,y}}$ controls the amount of deformation for each individual $z$-streamlines, and the choice of function $b_{\mathrm{z}}$ affects the maximum compression of space along the streamlines. The deformation of point $\mathrm{p} = (x,y,z)^\top$ is

$$\mathrm{p}_{def} = (x,y,z_{def})^\top \qquad (8)$$
$$\text{where } z_{def} = z + b_{\mathrm{x,y}}(x,y)\, b_{\mathrm{z}}(z)$$

It is the definition of $b_{\mathrm{z}}$ together with a corresponding threshold $d_{\max}$ that prevents foldovers, as shown in Figure 13. The following function is a possible choice for $b_{\mathrm{z}}(z)$, used in the example:

$$b_{\mathrm{z}}(z) = \begin{cases} 16z^2(1-z)^2 & \text{if } z \in [0,1] \\ 0 & \text{otherwise} \end{cases} \qquad (9)$$
$$\text{with } d_{\max} = \frac{3\sqrt{3}}{16} \simeq 0.324$$

Functions permitting larger values for $d_{\max}$ can be found in the original paper. Since $b_{\mathrm{x,y}}$ is independent of $z$, any function with values in $[-d_{\max}, d_{\max}]$ can be used for it, regardless of the slope. Because the amplitude of the effect of a blendeforming function is bounded by the $d_{\max}$ threshold, it is obvious that modeling an entire shape uniquely with blendeforming functions can be rather tedious. In the original paper, the authors also propose bending blendeforming functions, defined in cylindrical coordinates.

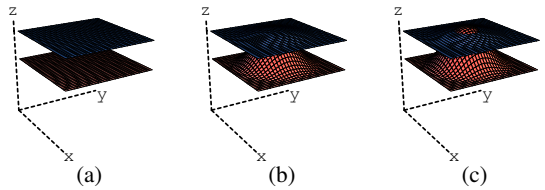



**Figure 13:** *(a) Initial scene: two parallel planes. (b) Blendeforming, with $b_{\mathrm{x,y}}(x,y) = (x^2 - x + y^2 - y - 1/2)^2$. The value of $d_{\max}$ guarantees that the two planes will never intersect. (c) With $d_{\max} < d$, foldover occurs: the lower plane intersects the higher plane.*

### 3.2. Lattice-based space deformations

The limitation of axial-based or surface-based space deformation is the arrangement of the controls along a curve or on a surface. Note that this statement is untrue only for wires, which permits the blending of the controls [SF98]. Lattice-based space deformations are techniques that allow control points to be connected along the three dimensions of space. There are two ways of understanding lattice-based deformation, related to the manner in which the artist expresses the deformation. Let us denote the space deformation function by $f$.

In the first interpretation of lattice-based deformations, the artist provides pairs of points: a source point and a destination point, $(\mathrm{p}_i, \mathrm{q}_i)$. The deformation $f$ will interpolate or approximate the pairs in this way $f(\mathrm{p}_i) = f_{\mathrm{p}}(\mathrm{p}_i) \approx \mathrm{q}_i$. The function $f_{\mathrm{p}}$ is a position field. A position field does not have any physical equivalent to which the artist or scientist can relate, and requires a certain amount of imagination to be visualized.

In the second interpretation of lattice-based deformations, the artist provides a source point and a displacement of that

point, $(\mathrm{p}_i, \vec{\mathrm{v}}_i)$. The deformation $f$ will interpolate or approximate the pairs in this way $f(\mathrm{p}_i) = \mathrm{p}_i + f_{\vec{\mathrm{v}}}(\mathrm{p}_i) \approx \mathrm{p}_i + \vec{\mathrm{v}}_i$. The function $f_{\vec{\mathrm{v}}}$ is a vector field. There is a convenient physical analogy to a vector field. Vector fields are used in fluid mechanics to describe the motion of fluids or to describe fields in electromagnetics [Rut90, Gri]. This analogy is of great help for explaining and creating new space deformations.

While the effect of using either a position field or a vector field is equivalent, the vector field also gives more insight in the process of deforming space: in lattice-based space deformations, the path that brings the source point onto the desired target point is a straight translation using a vector. In this section on lattice-based space deformation, we will therefore consider the construction of a vector field rather than a position field whenever possible.

#### 3.2.1. Free-form deformation of solid geometric models

The effect of Free-Form Deformation (FFD) on a shape is to embed this shape in a piece of flexible plastic. The shape deforms along with the plastic that surrounds it [SP86].

The idea behind FFD is to interpolate or approximate vectors defined in a 3d regular lattice. The vectors are then used to translate space. In their original paper, T. Sederberg and S. Parry propose to use the trivariate Bernstein polynomial as a smoothing filter. Let us denote by $\vec{\mathrm{v}}_{ijk}$ the $(l+1) \times (m+1) \times (n+1)$ *control vectors* defined by the artist. The smoothed vector field is a mapping $\mathrm{p} \in [0,1]^3 \mapsto \mathbb{R}^3$.

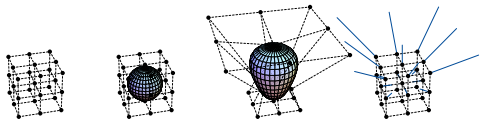
$$\vec{\mathrm{v}}(\mathrm{p}) = \sum_{i=0}^{l} \binom{i}{l} (1-x)^{l-i} x^i \left( \sum_{j=0}^{m} \binom{j}{m} (1-y)^{m-j} y^j \left( \sum_{k=0}^{n} \binom{k}{n} (1-z)^{n-k} z^k \right) \right) \vec{\mathrm{v}}_{ijk} \tag{10}$$

Then the deformation of a point is a translation of that point

$$\mathrm{p}_{def} = \mathrm{p} + \vec{\mathrm{v}}(\mathrm{p}) \tag{11}$$

In order for the deformation to be continuous across the faces of the FFD cube, the boundary vectors should be set to zero. A drawback of using the Bernstein polynomial is that a control vector $\vec{\mathrm{v}}_{ijk}$ has a non-local effect on the deformation. Hence updating the modification of a control vector requires updating the entire portions of shape within the lattice. For this reason, J. Griessmair and W. Purgathofer propose to use B-Splines [GP89].

In commercial software, the popular way to let the artist specify the control vectors is to let him move the control points of the lattice, as shown in Figure 14(c). A drawback often cited about this interface is the visual self occlusion of the control points. This problem increases with the increase in resolution of the lattice. Another drawback is the manipulation of control points, which requires high skills in spacial apprehension from the artist. Clearly, practical FFD manipulation through control-points can only be done with reasonably small lattices.



(a) (b) (c) (d)

**Figure 14:** *FFD. (a) Lattice of size $3^3$. (b) Initial shape. (c) The popular interaction with an FFD lattice consists of displacing the control points. (d) The discrete vectors.*

#### 3.2.2. Extended free-form deformation (EFFD)

Due to the practical limit of the size of the FFD-lattice, the major restriction of an FFD is strongly related to the arrangement of control-points in parallelepipeds. The parallelepipeds are also called *cells*. To provide the artist with more control, S. Coquillart proposes a technique with non-parallelepipedic and arbitrarily connected *cells*. The technique is called Extended Free-Form Deformation (EFFD) [Coq90].

To model with EFFD, the artist first builds a lattice by placing the extended cells anywhere in space, and then manipulates the cells to deform the shape. An extended cell is a small FFD of size $4^4$. The transformation from the cell's *local coordinates* $\mathrm{s} = (u,v,w)^\top$ to world coordinates is:

$$\mathrm{p}(\mathrm{s}) = \sum_{i=0}^{3} \binom{i}{3} (1-u)^{3-i} u^i \left( \sum_{j=0}^{3} \binom{j}{3} (1-v)^{3-j} v^j \left( \sum_{k=0}^{3} \binom{k}{3} (1-w)^{3-k} w^k \right) \right) \mathrm{p}_{ijk} \tag{12}$$

The eight corners $\mathrm{p}_{ijk \in \{0,3\}^3}$ of a cell are freely defined by the artist. The position of the remaining $4^4 - 8$ are constrained by the connection between cells, so that continuity is maintained across boundaries. This is done when the artist connects the cells. Because the lattice is initially deformed, finding a point's coordinates s in a cell is not straightforward. The local coordinates of a point p in a cell are found by solving Equation (12) in s using a numerical iteration. This can be unstable in some cases, although the authors report they did not encounter such cases in practice. Once s is found, the translation to apply to p is found by substituting in Equation (12) the control points $\mathrm{p}_{ijk}$ with the control vectors $\vec{\mathrm{v}}_{ijk}$. Note that specifying the control points, the cells and the control vectors is rather tedious, and results shown in the paper consist essentially of imprints.

#### 3.2.3. Free-form deformations with lattices of arbitrary topology (SFFD)

R.A. MacCracken and K.I. Joy have established a method that allows the user to define lattices of arbitrary shape and topology [MJ96]. The method is more stable than EFFD since it does not rely on a numerical iteration technique.

Their method is based on subdivision lattices. We will refer to it as SFFD, for subdivision FFD. The user defines a

control lattice, $L$: a set of vertices, edges, faces and cells. A set of refinement rules are repeatedly applied to $L$, creating a sequence of increasingly finer lattices $\{L_1, L_2, \dots L_l\}$. The union of cells define the deformable space. After the first subdivision, all cells can be classified into cells of different type: type-$n$ cells, $n \geq 3$. See [MJ96] for the rules.

Although there is no available trivariate parameterization of the subdivision lattice, the correspondence between world coordinates and lattice coordinates is possible thanks to the subdivision procedure. The location of a vertex embedded in the deformable space is found by identifying which cell contains it. Then, for a type-3 cell, trilinear parameterization is used. For a type-$n$ cell, the cell is partitioned in $4n$ tetrahedra, in which the vertex takes a trilinear parameterization. Each point is tagged with its position in its cell.

Once a point's location is found in the lattice, finding the point's new location is straightforward. When the artist displaces the control points, the point's new coordinates are traced through the subdivision of the deformed lattice.

#### 3.2.4. Direct manipulation of free-form deformations (DMFFD)

The manipulation of individual control points makes FFD and EFFD tedious methods to use. Two groups of researchers, P. Borrel and D. Bechmann, and W.M. Hsu et al. propose a similar way of doing direct manipulation of FFD control points (DMFFD) [BB91, HHK92]. The artist specifies translations $\vec{\mathrm{v}}_i$ at points $\mathrm{p}_i$ in the form $(\mathrm{p}_i, \vec{\mathrm{v}}_i)$. The DMFFD algorithm finds control vectors that satisfy, if possible, the artist's desire. Let us define a single input vector $\vec{\mathrm{v}}$ at point p. The FFD Equation (10) must satisfy

$$\vec{\mathrm{v}} = \mathrm{B}(\mathrm{p})(\vec{\mathrm{v}}_{ijk}) \tag{13}$$

Let $\nu = (3(l+1)(m+1)(n+1))$. The matrix B is the $3 \times \nu$ matrix of the Bernstein coefficients, which are functions of point p. Note that their method is independent of the chosen filter: instead of the Bernstein polynomials, W.M. Hsu et al. use B-Splines and remark that Bernstein polynomials can be used. P. Borrel and D. Bechmann on the other hand found that using simple polynomials works just as well as B-Splines. The size of the vector of control vectors $(\vec{\mathrm{v}}_{ijk})$ is $3(l+1)(m+1)(n+1)$. When the artist specifies $\mu$ pairs $(\mathrm{p}_i, \vec{\mathrm{v}}_i)$, the FFD Equation (10) must satisfy a larger set of equations:

$$\begin{pmatrix} \vec{\mathrm{v}}_1 \\ \vdots \\ \vec{\mathrm{v}}_\mu \end{pmatrix} = \mathbf{B} \cdot \begin{pmatrix} \vec{\mathrm{v}}_{ijk} \\ \vdots \\ \vec{\mathrm{v}}_{ijk} \end{pmatrix} \quad \text{where } \mathbf{B} = \begin{pmatrix} \mathrm{B}(\mathrm{p}_1) \\ \vdots \\ \mathrm{B}(\mathrm{p}_\mu) \end{pmatrix} \tag{14}$$

This set of equations can either be overdetermined or under determined. In either case, the matrix $\mathbf{B}$ cannot be inverted in order to find the $\vec{\mathrm{v}}_{ijk}$. The authors use the Moore-Penrose pseudo-inverse, $\mathbf{B}^+$. If the inverse of $\mathbf{B}^\top \cdot \mathbf{B}$ exists, then

$$\mathbf{B}^+ = (\mathbf{B}^\top \cdot \mathbf{B})^{-1} \cdot \mathbf{B}^\top \tag{15}$$

It is however preferable to compute the Moore-Penrose pseudo-inverse using singular value decomposition (SVD). The $\mu \times \nu$ matrix $\mathbf{B}$ can be written

$$\mathbf{B} = U \cdot D \cdot V^\top \tag{16}$$

where $U$ is an $\mu \times \mu$ orthogonal matrix, $V$ is an $\nu \times \nu$ orthogonal matrix and $D$ is an $\mu \times \nu$ diagonal matrix with real, non-negative elements in descending order.

$$\mathbf{B}^+ = V \cdot D^{-1} \cdot U^\top \tag{17}$$

Here, the diagonal terms of $D^{-1}$ are simply the inverse of the diagonal terms of $D$.

The size of the basis, or, equivalently the number of control points, has a direct effect on the locality of the deformation around the selected point. In their approach, P. Borrel and D. Bechmann pursue the reasoning even further, and define a technique suitable for $n$-dimensional objects [BB91]. In the context of shape animation, i.e. in $\mathbb{R}^4$ with time as the fourth dimension, the Bernstein, B-Splines or simple polynomials are inappropriate. They propose to use a basis that does not change the initial time, $t_0$, and final time, $t_f$, of an object:

$$\mathrm{B}_t(\mathrm{p},t) \quad = \\ \Big((t-t_0)(t-t_f)\ ,\ (t-t_0)(t-t_f)t\ ,\ (t-t_0)(t-t_f)t^2\ ,\ \dots\Big)^\top$$

#### 3.2.5. Simple constrained deformations for geometric modeling and interactive design (scodef)

In simple constrained deformations (scodef), P. Borrel and A. Rappoport propose to use DMFFD with radial basis functions (RBF) [BR94]. The artist defines constraint triplets $(\mathrm{p}_i, \vec{\mathrm{v}}_i, r_i)$: a point, a vector that defines the desired image of the point, and a radius of influence. Let $\phi_i(\mathrm{p})$ denote the scalar function $\phi(\frac{\|\mathrm{p}-\mathrm{p}_i\|}{r_i})$ for short. The motivation of using RBF is to keep the deformation local, in the union of spheres of radius $r_i$ around the points $\mathrm{p}_i$. A naive vector field would be:

$$\vec{\mathrm{v}}(\mathrm{p}) = \sum_{i=1}^{n} \vec{\mathrm{v}}_i \phi_i(\mathrm{p}) \tag{18}$$

Unless the points $\mathrm{p}_i$ are far apart enough, Equation (18) will not necessarily satisfy the artist's input $\vec{\mathrm{v}}(\mathrm{p}_i) = \vec{\mathrm{v}}_i$ if the functions $\phi_i$ overlap. However, this can be made possible by substituting the vectors $\vec{\mathrm{v}}_i$ with suitable vectors $\vec{\mathrm{w}}_i$.

$$\vec{\mathrm{v}}(\mathrm{p}) = \sum_{i=1}^{n} \vec{\mathrm{w}}_i \phi_i(p) \tag{19}$$

These vectors $\vec{\mathrm{w}}_i$ can be found by solving a set of $3n$ equations:

$$\vec{\mathrm{v}}_i = (\vec{\mathrm{w}}_1 \dots \vec{\mathrm{w}}_n) \cdot \begin{pmatrix} \phi_1(\mathrm{p}_i) \\ \vdots \\ \phi_n(\mathrm{p}_i) \end{pmatrix} \text{ where } i \in [1,n] \tag{20}$$

Let us take the transpose, and arrange the $n$ equations in rows. The following equation is the equivalent of Equation (14), but with radial basis functions:

$$\begin{pmatrix} \vec{v}_1^{\top} \\ \vdots \\ \vec{v}_n^{\top} \end{pmatrix} = \begin{pmatrix} \phi_1(p_1) & \dots & \phi_n(p_1) \\ \vdots & & \\ \phi_1(p_n) & \dots & \phi_n(p_n) \end{pmatrix} \cdot \begin{pmatrix} \vec{w}_1^{\top} \\ \vdots \\ \vec{w}_n^{\top} \end{pmatrix} \quad (21)$$

where $i \in [1,n]$

Let $\mathbf{\Phi}$ be the $n \times n$ square matrix of Equation (21). This matrix takes the role of $\mathbf{B}$ in Equation (14). Since $\mathbf{\Phi}$ can be singular, the authors also use its pseudo-inverse $\mathbf{\Phi}^{+}$ to find the vectors $\vec{w}_i$.

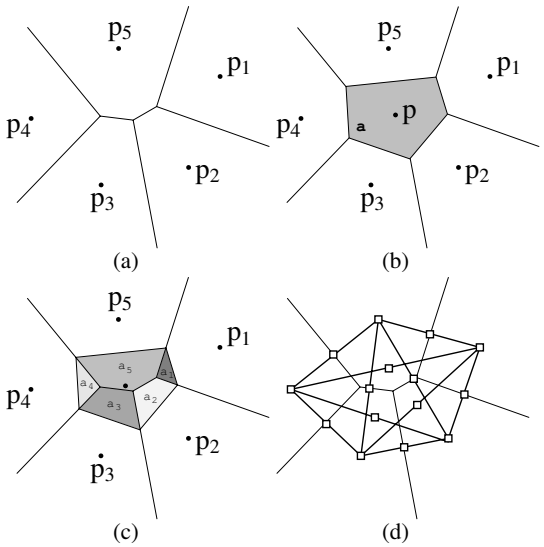



**Figure 15:** *2D illustration of the Sibson coordinates (a) Voronoï cells of the control points. (b) Voronoï diagram is updated after the insertion of point* p. *(c) The areas stolen by the point* p *from its natural neighbors give the Sibson coordinates $a_i/a$. (d) Local control net, with Bézier abscissa.*

### 3.2.6. Dirichlet free-form deformation (DFFD)

With DFFD, L. Moccozet and N. Magnenat-Thalmann propose a technique that builds the cells of a lattice automatically [MMT97], relieving the artist from a tedious task. The lattice cells are the cells of a Voronoï diagram of the control points, shown in Figure 15. The location of a point within a cell is neatly captured by the Sibson coordinates. The naive deformation of a point p is given by interpolating vectors defined at the control points with the Sibson coordinate.

$$\mathrm{p} \mathrel{+}= \sum_{i=1}^{n} \frac{a_i}{a} \vec{v}_i \quad (22)$$

Where $a_i$ is the volume of cell $i$ stolen by p, and $a$ is the volume of the cell of p. This interpolation is only $C^0$. They use a method developed by G. Farin [Far90] to define a continuous parameterization on top of the Sibson coordinates. The interpolation is made of four steps:

- build the local control net
- build *Bézier abscissa*
- define *Bézier ordinates* such that the interpolant is $C^1$
- evaluate the multivariate Bernstein polynomial using Sibson coordinates.

### 3.2.7. Preventing self-intersection under free-form deformation

In FFD, EFFD and DMFFD, if the magnitude of a control-vector is too high, the deformation may produce a self-intersection of the shape's surface (see a self-intersection in Figure 13). Once the shape's surface self-intersects, there is no space deformation that can remove the self-intersection. The appearance of this surface incoherency is the result of a space foldover: the deformation function is a surjection of $\mathbb{R}^3$ onto $\mathbb{R}^3$, not a bijection. J. Gain and N. Dodgson present foldover detection tests for DMFFD deformations that are based on uniform B-Splines [GD01]. They argue that a necessary and sufficient test is too time consuming, and present an alternative sufficient test. Let us define $q_{ijk}$, the deformed control points of the lattice. If the determinants of all the following $3 \times 3$ matrices are all positive, there is no foldover.

$$\phi_{ijk} = s \ \det\left(q_{i\pm 1jk} - q_{ijk}, q_{ij\pm 1k} - q_{ijk}, q_{ijk\pm 1} - q_{ijk}\right) \quad (23)$$

where the sign $s$ is 1 if $(i \pm 1, j \pm 1, k \pm 1)$ are clockwise, else $-1$.

The idea underlying the test is that the determinant of three column vectors is the volume of the parallelepiped defined by these vectors. A negative volume detects a possible singularity in the deformation. A technique for efficiently testing several determinants at once can be found in the original paper.

This test can then be used to repair the DMFFD. Let us define $(p_i, \vec{v}_i)$, the pairs of points and vectors defining the DMFFD. If a foldover is detected, the DMFFD is recursively

split into two parts: $(p_i, \vec{v}_i/2)$ and $(p_i + \vec{v}_i/2, \vec{v}_i/2)$. The procedure eventually converges, and the series of DMFFDs obtained are foldover-free and can be applied safely to the shape.

### 3.3. Tool-based space deformations

Lattice-based techniques are capable of building a wide range of vector fields. But when dealing with a problem in animation, modeling or visualization, a technique tailored for that specific problem will be more suitable. This section is about techniques that focus on a particular unresolved problem of space deformation, and solve it in an original way.

#### 3.3.1. Interactive space deformation with hardware assisted rendering

Y. Kurzion and R. Yagel present *ray deflectors* [KY97]. The authors are interested in rendering the shape by deforming the rays, as opposed to directly deforming the shape. To deform the rays, one needs the inverse of the deformation that the artist intends to apply to the shape. Rather than defining a deformation and then trying to find its inverse, the authors directly define deformations by their inverse. Their tool can translate, rotate and scale space contained in a sphere, locally and smoothly. Moreover they define a discontinuous deformation that allows the artist to cut space, and change a shape's topology. A tool is defined within a ball of radius $r$ around a center c. Let $\rho$ be the distance from the center of the deflector c and a point p.

$$\rho = \|p - c\| \tag{24}$$

**3.3.1.1. Translate deflector:** To define a translate deflector, the artist has to provide a translation vector, $\vec{t}$. The effect of the translate deflector will be to transform the center point, c, into $c + \vec{t}$.

$$f_T(p) = \begin{cases} p - \vec{t}(1 - \frac{\rho^2}{r^2})^2 & \text{if } \rho < r \\ p & \text{otherwise} \end{cases} \tag{25}$$

where $\theta \in \mathbb{R}$

**3.3.1.2. Rotate deflector:** To define a rotate deflector, the artist has to provide an angle of rotation, $\theta$, and a vector, $\vec{n}$, about which the rotation will be done. The reader can find the expression of a rotation matrix, $R_{\theta', \vec{n}, c}$, in Appendix **??**. Let us call $\theta'$ an angle of rotation that varies in space:

$$\theta' = -\theta(1 - \frac{\rho^2}{r^2})^4$$

$$f_R(p) = \begin{cases} R_{\theta', \vec{n}, c} \cdot p & \text{if } \rho < r \\ p & \text{otherwise} \end{cases} \tag{26}$$

where $\|\vec{t}\| \in [0, \frac{3\sqrt{3}r}{8}]$

**3.3.1.3. Scale deflector:** To define a scale deflector, the artist has to provide a scale factor $s$. The scale deflector acts like a magnifying glass.

$$f_S(p) = \begin{cases} p - (p - c)(1 - \frac{\rho^2}{r^2})^4 s & \text{if } \rho < r \\ p & \text{otherwise} \end{cases} \tag{27}$$

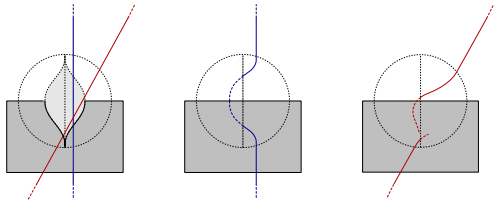
where $s \in [-1, 1]$

**3.3.1.4. Discontinuous deflector:** To define a discontinuous deflector, the artist has to provide a translation vector, $\vec{t}$. The deflector is split into two halves, on each side of a plane going through c and perpendicular to $\vec{t}$. In the half pointed at by $\vec{t}$, the discontinuous deflector will transform c, into $c + \vec{t}$, while in the other half, the discontinuous deflector will transform c, into $c - \vec{t}$. The effect will be to cut space. The deformation applied to the rays is:

$$f_D(p) = \begin{cases} p - \vec{t}(1 - \frac{\rho^2}{r^2})^2 & \text{if } \rho < r \text{ and } 0 < (p - c) \cdot \vec{t} \\ p + \vec{t}(1 - \frac{\rho^2}{r^2})^2 & \text{if } \rho < r \text{ and } (p - c) \cdot \vec{t} < 0 \\ p & \text{otherwise} \end{cases} \tag{28}$$

where $\theta \in \mathbb{R}$

Since this deformation is discontinuous on the disk separating the two halves of the deformation, a ray crossing that disk will be cut in two, as we show in Figure 16(c). Thus a shape intersection algorithm will have to march along the ray from the two sides of the ray, until each curve crosses the separating disk. This deformation assumes that the shape's representation has an inside and outside test. Note that other authors have extended FFD for dealing with discontinuities [SE04].



(a) (b) (c)

**Figure 16:** *(a) Discontinuous deflector as observed by the artist. Two arbitrary rays are shown. (b) Simple case, where the ray of light crosses only one hemisphere. (c) When the ray of light changes hemisphere, the curve of light is subject to a discontinuity.*

#### 3.3.2. Geometric deformation by merging a 3D object with a simple shape

P. Decaudin proposes a technique that allows the artist to model a shape by iteratively adding the volume of simple 3D shapes [Dec96]. His method is a metaphor of clay sculpture by addition of lumps of definite size and shape. His deformation function is a closed-form, as opposed to a numerical method that would explicitly control the volume [HML99].

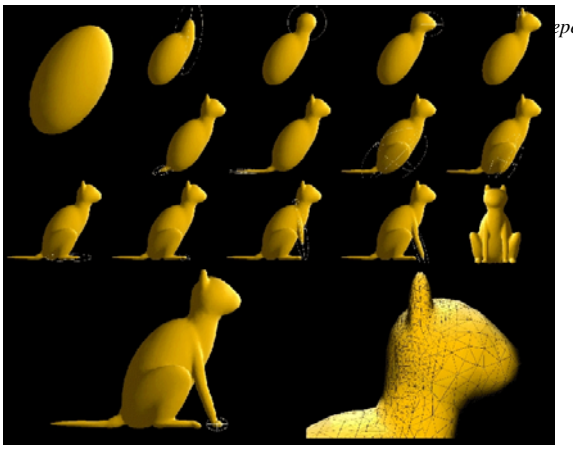


**Figure 17:** *Steps of the modeling of a cat (courtesy of [Dec96]).*

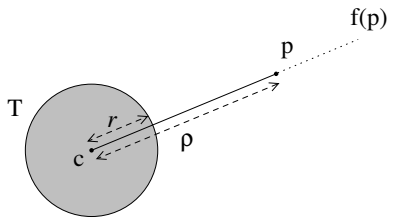



**Figure 18:** *The insertion of a tool at center* c *affects the position of point* p. *See the deformation in Equations (29).*

Loosely speaking, this technique inflates space by blowing up a tool in space through a hole. This will compress space around the point in a way that preserves the outside volume. Hence if the tool is inserted inside the shape, the tool's volume will be added to the shape's volume. On the other hand, if the tool is inserted outside the shape, the shape will be deformed but its volume will remain constant. This is illustrated for the 2D case in Figure 19. A restriction on the tool is to be star-convex with respect to its center c . The deformation function is[‡] (see Figure 18):

$$f_{3D}(\mathrm{p}) = \mathrm{c} + \sqrt[3]{\rho(\mathrm{p})^3 + r(\mathrm{p})^3}\,\vec{\mathrm{n}} \qquad (29)$$

- $\rho(\mathrm{p})$ is the magnitude of the vector $\vec{\mathrm{u}} = \mathrm{p} - \mathrm{c}$.
- $r(\mathrm{p})$ is the distance between c and the intersection of the tool with the half-line $(\mathrm{c}, \vec{\mathrm{u}})$.
- $\vec{\mathrm{n}} = \vec{\mathrm{u}}/\|\vec{\mathrm{u}}\|$ is the unit vector pointing from c to p.

If the tool was not a star-convex in c, then $r(\mathrm{p})$ would be ambiguous. The deformation is foldover-free. It is continuous everywhere except at the center c. The effect of the deformation converges quickly to the identity with the increasing distance from c. The deformation can be considered local, and is smooth everywhere except at c. An example in $3D$ is shown in Figure 17. A feature of this space deformation which is rare, is that it has an exact yet simple inverse in the space outside the tool:

$$f_{3D}^{-1}(\mathrm{p}) = \mathrm{c} + \sqrt[3]{\rho(\mathrm{p})^3 - r(\mathrm{p})^3}\,\vec{\mathrm{n}} \qquad (30)$$

[‡] The 2D case is obtained by replacing 3 with 2.

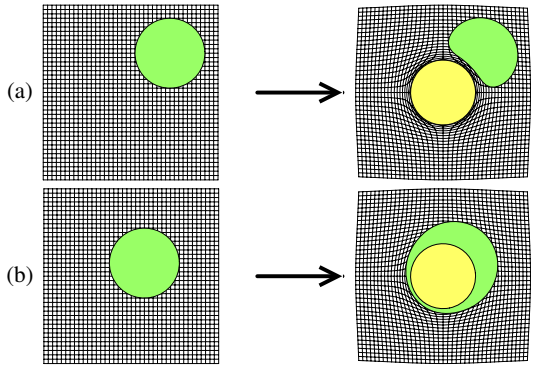



**Figure 19:** *(a) Deformation of a shape (green) by blowing up a tool (yellow) outside the shape. The shape's area is preserved. (b) Deformation of a shape by blowing up a tool inside the shape. The shape's area is increased by that of the tool.*

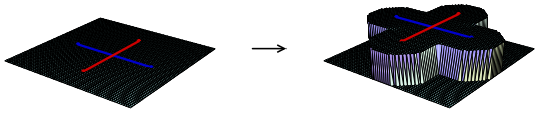
### 3.3.3. Implicit free-form deformations (IFFD)

B. Crespin introduces Implicit Free-Form Deformations (IFFD) [Cre99]. Note that though it is called implicit, the deformation is explicit. IFFD is rather a technique inspired by implicit surfaces, a vast branch of computer graphics whose presentation is beyond the scope of this manuscript [BBB*97]. The field $\phi \in [0,1]$ generated by a skeleton modulates a transformation, $M$, of points. The deformation of point p with a single function is:

$$f(\mathrm{p}) = \mathrm{p} + \phi(\mathrm{p})(M \cdot \mathrm{p} - \mathrm{p}) \qquad (31)$$

He proposes two ways to combine many deformations simultaneously. Let use denote $\mathrm{p}_i$ the transformation of p with deformation $f_i$. The first blending is shown in Figure 20. For $M$, we have used a translation matrix.

$$p_{def} = p + \frac{\sum_{i=1}^{n}(p_i - p)\phi_i(p)}{\sum_{i=1}^{n}\phi_i(p)}$$

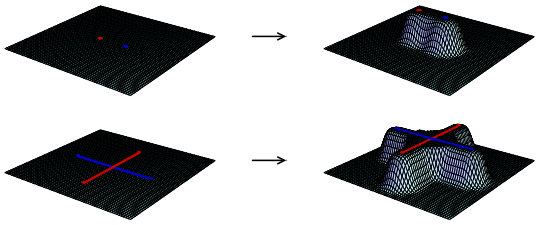


Reference segments Translated segments

**Figure 20:** *Blending weights based on summed displacement magnitudes. The deformation is only defined where the amounts $\phi$ are not zero, and is discontinuous at the interface $\sum_i \phi_i = 0$. This blending is useful when the deformed shape is entirely contained within the field.*

The second blending attempts to solve the continuity issue, but requires the definition of supplementary profile functions, $\gamma_i$. The purpose of the index $i$ is to assign individual profiles to skeletons.

$$p_{def} = p + \frac{\sum_{i=1}^{n}(p_i - p)\phi_i(p)\gamma_i(p)}{\sum_{i=1}^{n}\phi_i(p)}$$



Reference segments Translated segments

**Figure 21:** *Blending weights based on displacement magnitudes and profile functions. For control points, the technique works well. For segments, there is a discontinuity near their intersection.*

In order to produce Figure 21, the following $\gamma_i$ function was used:

$$\gamma_i(p) = \begin{cases} 1-(1-\sigma^2)^2 & \text{if } \sigma \in [0,1], \\ & \text{where } \sigma = \sum_{i=1}^{n}\phi_i(p) \\ 1 & \text{otherwise} \end{cases} \quad (32)$$

### 3.3.4. Twister

I. Llamas et al propose a method called twister in which a twist transformation of points is weighted with a scalar function [LKG*03], i.e. in a similar way to IFFD but with a transformation restrained to a twist. With this restriction, they propose to weight single twists along the trajectory of transformation rather than weighting the displacement. They define a twist by transforming an orthonormal coordinate system $(o, \vec{u}, \vec{v}, \vec{w})$ into $(o', \vec{u}', \vec{v}', \vec{w}')$. The axis of the twist is defined by a direction $\vec{d}$ and point $a$ on the axis, while the angle of rotation around the axis is $\alpha$ and the translation factor along the axis is $d$:

$$\begin{aligned} \vec{d} &= \frac{\vec{g}}{\|\vec{g}\|} \\ \text{where } \vec{g} &= (\vec{u}'-\vec{u})\times(\vec{v}'-\vec{v}) + \\ &\quad (\vec{v}'-\vec{v})\times(\vec{w}'-\vec{w}) + \\ &\quad (\vec{w}'-\vec{w})\times(\vec{u}'-\vec{u}) \\ \alpha &= 2\arcsin\left(\frac{\|\vec{u}'-\vec{u}\|}{2\|\vec{d}\times\vec{u}\|}\right) \\ d &= \vec{d}\cdot(o'-o) \\ a &= \frac{o+o'-d\vec{d}}{2} + \frac{\vec{d}\times(o-o')}{2\tan(\alpha/2)} \end{aligned} \quad (33)$$

Their procedure for deforming a point p with a twist parameterized in $t$ is:

1. Bring p into local coordinates: translate by $-\vec{a}$ and then rotate by a rotation that maps $\vec{d}$ onto $\vec{z}$.
2. Apply the twist in local coordinates: translate by $t\ d$ along $\vec{z}$ and rotate by $t\ \alpha$ around $\vec{z}$
3. Finally bring p back into world coordinates: rotate by a rotation that maps $\vec{z}$ onto $\vec{d}$ and translate by $\vec{a}$

To weight the twist, they propose to use a piecewise scalar function:

$$t(p) = \cos^2(\pi\|p-o\|/2r) \quad (34)$$

For operations that require simultaneous twists, they propose simply to add the displacement of the weighted twist. Details for defining a two-point constraint can be found in the paper.

### 3.3.5. Scalar-field guided adaptive shape deformation and animation (SFD)

J. Hua and H. Qin create a technique called SFD [HQ04]. They define a deformation by attaching space to the level-sets of an animated scalar field. The artist is offered three different techniques for animating a scalar field. Since there are many ways of attaching a point to a level-set of a scalar field, the authors choose the way that keeps the shape as rigid as possible.

They define $\phi(t, p(t))$, the scalar field which is animated in time, $t$. Since a moving point, $p(t)$, is attached to a level-set of the scalar field, the value of $\phi$ at p is constant in time:

$$\frac{d\phi}{dt} = 0 \quad (35)$$

The square of Equation (35) gives a constraint:

$$\left(\frac{d\phi}{dt}\right)^2 = 0 \quad (36)$$

There are several ways of attaching a point to a level set while the scalar field is moving. The simplest way would be to make a point follow the shortest path, found when the magnitude of the point's speed, $\|\vec{v}(t)\|$, is minimized. Another possibility, chosen by the authors, is to minimize the variation of velocity, so that the deformation is as rigid as possible. Instead of using the divergence of the speed to measure rigidity, they use an estimate by averaging the variation

of speed between that point's speed, $\vec{v}$, and its neighbors' speed, $\vec{v}_k$:

$$(\nabla \cdot \vec{v})^2 \approx \frac{1}{k} \sum_k (\vec{v} - \vec{v}_k)^2 \tag{37}$$

Since this is a constrained optimization problem [Wei04], there exists a Lagrange multiplier $\lambda$ such that:

$$\frac{d}{d\vec{v}} \left( \frac{d}{dt} \phi(t, p(t)) \right)^2 + \lambda \frac{d}{d\vec{v}} (\nabla \cdot \vec{v})^2 = \vec{0} \tag{38}$$

According to the authors, $\lambda$ is an experimental constant, used to balance the flow constraint and speed variation constraint. Its value ranges between 0.05 and 0.25. We rearrange this equation and expand the derivative of $\phi$ with the chain rule:

$$\frac{d}{d\vec{v}} \left( (\nabla \phi \cdot \vec{v} + \frac{\partial \phi}{\partial t})^2 + \lambda (\nabla \cdot \vec{v})^2 \right) = \vec{0} \tag{39}$$

Let us define $\hat{\vec{v}}$, the average of the velocity of all the adjacent neighbors connected with edges to point p. If we substitute $(\nabla \cdot \vec{v})^2$ for its approximate given by Equation (37), and then apply the derivative with respect to $\vec{v}$, we obtain:

$$(\nabla \phi \cdot \vec{v} + \frac{\partial \phi}{\partial t}) \nabla \phi + \lambda (\vec{v} - \hat{\vec{v}}) = \vec{0} \tag{40}$$

By solving the system of Equation (40), the updated position is:

$$\vec{v} = \hat{\vec{v}} - \frac{\hat{\vec{v}} \cdot \nabla \phi + \frac{\partial \phi}{\partial t}}{\lambda + (\nabla \phi)^2} \nabla \phi \tag{41}$$

The algorithm deforms a set of vertices in $n$ sub-steps. If $n$ is set to one, the deformation takes one step:

**for** i = 1 to n **do**
  **for all** $p_k$ in the list of vertices to update **do**
    Update the scalar field $\phi(t + \Delta t, p_k)$.
    Deduce $\frac{\partial \phi}{\partial t} = \frac{\phi(t+\Delta t, p_k) - \phi(t, p_k)}{\Delta t}$
    Calculate $\nabla \phi$, possibly with finite differences.
    Compute $\hat{\vec{v}}$ according to neighbors' velocities.
    Deduce $\vec{v}$ according to Equation (41).
    Update vertex positions with $p_k(t + \Delta t) = p_k(t) + \vec{v} \frac{\Delta t}{n}$
    Improve surface representation using a mesh refinement and simplification strategy.
    **if** $\phi(t + \Delta t, p_k(t + \Delta t)) \approx \phi(t, p_k(t))$ **then**
      remove $p_k$ from the list of vertices to update.
    **end if**
  **end for**
**end for**

In the first step, since all the speeds are zero, we suggest that they could be initialized with:

$$\vec{v} = -\frac{\frac{\partial \phi}{\partial t}}{\lambda + (\nabla \phi)^2} \nabla \phi \tag{42}$$

Firstly, this technique is not a very versatile space deformation technique since it requires an explicit surface in order to compute the divergence of the speed. Secondly, the advantage of a large set of possible SFD shape operations (as large as the set of possible animated scalar fields) is at the cost of making the artist's task rather tedious: specifying the animated field does not permit quick and repeated operations on the shape. Also, results show the editing of imported shapes rather than shapes entirely modeled from scratch.

### 3.4. Limitations

The large number of space deformation techniques can lead quickly to the naive conclusion that in any shape modeling by deformation scenario, the limitation of a technique may be simply circumvented by using another technique. This reasoning presents several flaws. Firstly, from the point of view of a programmer, the amount of effort required to implement a space deformation Swiss-army knife for shape modeling would be considerable. Secondly, from the point of view of an artist, choosing quickly the most appropriate space deformation would require a vast amount of knowledge of the underlying mathematics of many techniques, which is a skill that should not be required. Thirdly, from a researcher's point of view, all space deformation techniques are not necessarily designed for the specific purpose of shape modeling, and there are surely efficient ways of dealing with specific problems. We will discuss this last point in the remainder of this section, i.e. we will overview the suitability of individual space deformation techniques for the purpose of interactive shape modeling.

Firstly, the subset of space deformations, whose effect on a shape is not local, makes these techniques unsuitable for the task of modeling shapes, since an artist's operation on a visible portion of the shape will have effects on portions that are further away [Bar84, Bla94, CR94, LCJ94].

Secondly, a large number of space deformation techniques requires the artist to specify a rather large number of control parameters [SP86, Coq90, MJ96, MMT97, HML99, HQ04]. We believe that increasing the number of parameters does not increase the amount of control by an artist, but rather it makes the task longer and more tedious. Many techniques illustrate their capabilities on imported models, that were either digitized or pre-modeled with conventional modeling techniques with a few exceptions [Dec96, HHK92, LKG*03]. We believe that the absence of a model entirely developed in one piece with a single technique is some evidence that the technique is tedious to use for the dedicated purpose of modeling shapes.

Finally, many space deformation techniques do not prevent a surface from self-intersecting after deformation, aside from a few exceptions [Dec96, MW01, GD01]. A self-intersecting surface is a rather annoying situation in modeling with deformation, since it is impossible for a space deformation to remove a previously introduced self-intersection. Thus we believe that space deformation operations for shape modeling should satisfy all the following criteria:

- Its effective span should be controllable.
- Its input parameters should be reduced to their strict minimum: a gesture.
- It should be predictable, in accordance with a metaphor.
- It should be foldover-free, and even revertible.
- It should be sufficiently fast for existing computing devices.

To our knowledge, the literature does not contain techniques satisfying all the above criteria. Rather than defining a set of unrelated techniques, we will specify a framework in which we will define deformation operations that satisfy the above. We will illustrate the modeling capabilities of our framework with techniques and examples.

## 4. Modeling with gesture

In the following, the input that defines transformations is a gesture, obtained with a mouse or hand tracking device. For the sake of simplicity, we will denote by $f$ the function $f_{t_i \mapsto t_{i+1}}$.

### 4.1. Naive deformation

A simple space deformation can be defined with a transformation (translation, rotation, scale, etc.) whose effect is spatially weighted. Thus two entities suffice:

- a transformation: $4 \times 4$ matrix $M$, defined by a gesture (mouse move, tracked hand)
- the amount of transformation at $\mathrm{p} \in \mathbb{R}^3$: scalar field $\varphi(\mathrm{p}) \in [0,1]$, defined for instance using the distance to a shape. We call *tool* the field.

The most straightforward way of weighting $M$ with $\varphi$ is to weight the displacement induced by $M$ at p:

$$\dot{f}(\mathrm{p}) = \mathrm{p} + \varphi(\mathrm{p})\,(M \cdot \mathrm{p} - \mathrm{p}) \tag{43}$$

This weighting of $M$ produces however poor results in several cases, including when $M$ is a rotation matrix. To compute fractions of a transformation, we rather use the formalism of M. Alexa [Ale02], i.e. the multiplication operator $\odot$ which behaves essentially like $\cdot$ for scalars (see Appendix A). Note that although we use Alexa's operator, we do not necessarily evaluate it numerically as proposed in his paper, since some cases reduce to more efficient and elegant closed-form formulas, as we will show. Thus the transformation $M$ can be weighted with $\varphi$ as follows:

$$\ddot{f}(\mathrm{p}) = (\varphi(\mathrm{p}) \odot M) \cdot \mathrm{p} \tag{44}$$

The deformation $\ddot{f}$ is however naive, since it can create a foldover. For example, if $M$ is a translation of large magnitude, it can map points within the support of $\varphi$ onto points outside from the support of $\varphi$, thus folding space onto itself as shown in Figure 22(left).

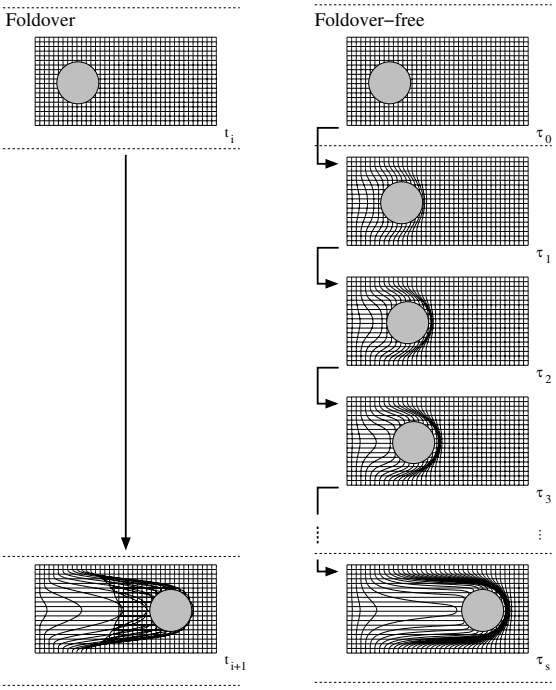



**Figure 22:** *2D illustration of our solution to foldovers. Left: the deformation maps space onto itself. Right: the deformation is decomposed into small foldover-free steps.*

### 4.2. Defining simple tools

To define a tool, a smooth function $\mu$ can be composed to the distance to a shape. We chose to use the following $C^2$ piecewise polynomial, in which $\lambda$ controls the size of the influence of the tool:

$$\mu_\lambda(d) = \begin{cases} 0 & \text{if} \quad \lambda \le d \\ 1 + (\frac{d}{\lambda})^3 (\frac{d}{\lambda}(15 - 6\frac{d}{\lambda}) - 10) & \text{if} \quad d < \lambda \end{cases} \tag{45}$$

**Ball tool:** The distance to a ball has a simple expression in local coordinates:

$$d_{sphere}(\mathrm{p}) = \begin{cases} 0 & \text{if } \|M_{t_i}^{-1} \cdot \mathrm{p}\|^2 \le 1 \\ \det(M_{t_i}^{\frac{1}{3}})(\|M_{t_i}^{-1} \cdot \mathrm{p}\| - 1) & \text{otherwise} \end{cases} \tag{46}$$

If the artist wishes to apply a non-uniform scale to the sphere, it would turn into an ellipsoid, and Equation (46) would not be usable.

**Filled ellipsoid tool:** The ellipsoid is defined in local coordinates as a unit sphere, whose position in world coordinates is encoded in a possibly non-uniform matrix $M_{t_i}$. To compute the distance to a filled ellipsoid, we use the numerical method described in [Ebe01].

$$d_{\text{ellipsoid}}(\mathrm{p}) = \begin{cases} 0 & \text{if } \|M_{t_i}^{-1} \cdot \mathrm{p}\|^2 \le 1 \\ \min\limits_{\mathrm{q} \in S} \|\mathrm{p} - M_{t_i} \cdot \mathrm{q}\| & \text{otherwise,} \\ \text{where } S \text{ is the unit sphere at the origin} & \end{cases} \tag{47}$$

**Mesh tool:** It is convenient for an artist to choose or manufacture his own tools. For this purpose, we propose the possibility of *baking* pieces of clay in order to use them as tools. By baking, we mean precomputing a data structure such that the distance field can be computed efficiently. We propose one way in [Ang05] (see Figure 23). More information on distance computation can be found in [Gué01].

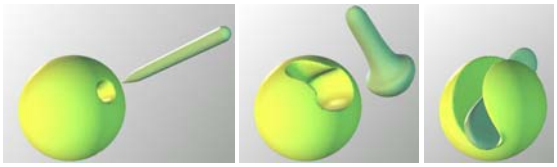


**Figure 23:** *Example of customized tools deforming a sphere.*

### 4.3. Single sweeper

As shown in Figure 22(right), if we decompose the transformation into a series of $s$ small enough transformations, and apply each of them to the result of the previous one, foldovers are avoided. The decomposition in $s$ steps for a general transformation is expressed as follows:

$$\begin{array}{rcl} f(\mathrm{p}) & = & \bigcirc_{k=0}^{s-1} f_k(\mathrm{p}) \\ \text{where } f_k(\mathrm{p}) & = & (\frac{\varphi^k(\mathrm{p})}{s} \odot M) \cdot \mathrm{p} \\ \text{and } \varphi^k(\mathrm{p}) & = & \varphi((\frac{k}{s} \odot M^{-1}) \cdot \mathrm{p}) \end{array} \quad (48)$$

The value returned by $\varphi^k$ is that of the scalar field $\varphi$ transformed by $\frac{k}{s} \odot M$, a fraction of $M$. It can be shown that there exists a finite number of steps such that the deformation is foldover-free (see [Ang05]). We propose the following as a lower bound to the required number of steps $s$:

$$\max_{\mathrm{p}} \|\nabla\varphi(\mathrm{p})\| \max_{l\in[1,8]} \|\log(M)\cdot \mathrm{p}_l\| < s \quad (49)$$

where $\mathrm{p}_{l\in[1,8]}$ are the corners of a box oustide which the function $\varphi$ equals zero.

### 4.4. Simultaneous sweepers

Applying more than one operation at the same time and the same place has applications in modeling: for instance for modeling a symmetric object, or to define a tool composed of several tools. The simultaneous manipulation of tools also allows the artist to pinch a shape. Let us consider $n$ operations, defined with $M_{i\in[1,n]}$ and $\varphi_{i\in[1,n]}$. The following is a naive way to achieve simultaneous deformations, using the formalism of M. Alexa (see Appendix A):

$$f(\mathrm{p}) = (\bigoplus_{i=1}^{n} \varphi_i(\mathrm{p}) \odot M_i) \cdot \mathrm{p} \quad (50)$$

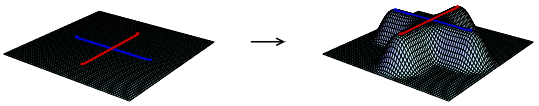
This function is naive because it adds the effect of each operation. The following expression provides a *normalized* and *smooth*[§] combination of all the transformations at any point p in space[¶]:

$$\begin{cases} \mathrm{p} & \text{if } \sum_k \varphi_k = 0 \\ \bigoplus_{i=1}^{n} \left( \left( \frac{1-\prod_k (1-\varphi_k)}{\sum_k \varphi_k} \varphi_i \right) \odot M_i \right) \cdot \mathrm{p} & \text{otherwise} \end{cases} \quad (51)$$

where:

- $\frac{1}{\sum_k \varphi_k}$ is required to produce a **normalized** combination of the transformations. This prevents for instance two translations of vector $\vec{d}$ producing translations of vector $2\vec{d}$, which would send some points far away from the tools. This unwanted behaviour was also identified by K. Singh and E. Fiume [SF98].
- $1-\prod_{k=1}^{n}(1-\varphi_k(\mathrm{p}))$ **smooths** the deformation in the entire space. Smoothness would be lost between the regions where $\sum_k \varphi_k = 0$ and $\sum_k \varphi_k \neq 0$ if we only used the normalization above.

Figure 25 shows a comparison between additive blending of Equation (50) and the correct one of Equation (51). In Figure 24, we show our blending in a scenarion similar to existing blending methods, presented in Section 3.



Reference segments Translated segments

**Figure 24:** *Blending with sweepers. The surface appears nice and smooth, as opposed to surfaces in Figures 10,11,12, 25,21 and 20.*

Equation (51) however may produces foldovers for similar reasons to the case of a single tool, with Equation (44). If we decompose it into small steps, foldovers can be avoided:

$$\begin{array}{rcl} f(\mathrm{p}) & = & \bigcirc_{k=0}^{s-1} f_k(\mathrm{p}) \\ \text{where } f_k(\mathrm{p}) & = & \begin{cases} \mathrm{p} & \text{if } \sum_j \varphi_j^k = 0 \\ \bigoplus_{i=1}^{n} \left( \left( \frac{1-\prod_j (1-\varphi_j^k)}{\sum_j \varphi_j^k} \varphi_i^k \right) \odot M_i \right) \cdot \mathrm{p} & \text{otherwise} \end{cases} \\ \text{and } \varphi_j^k(\mathrm{p}) & = & \varphi_j((\frac{k}{s} \odot M_j^{-1}) \cdot \mathrm{p}) \end{array} \quad (52)$$

Note that the value returned by $\varphi_j^k$ is that of the scalar field $\varphi_j$ transformed by $\frac{k}{s} \odot M_j$, a fraction of $M_j$. The following expression is a lower bound to the required number of steps, generalizing the single tool condition (see justification in [Ang05]):

$$\sum_j \max_{\mathrm{p}} (\|\nabla\varphi_j(\mathrm{p})\|) \max_{l\in[1,8]} \left|\left| \log M_j \cdot \mathrm{p}_{l_j} \right|\right| < s \quad (53)$$

§ as smooth as the $\varphi_i$.

¶ The operator $\bigoplus$ expresses a repetive sum: $\bigoplus_{i=1}^{n} M_i = M_1 \oplus M_2 \oplus \cdots \oplus M_n$.

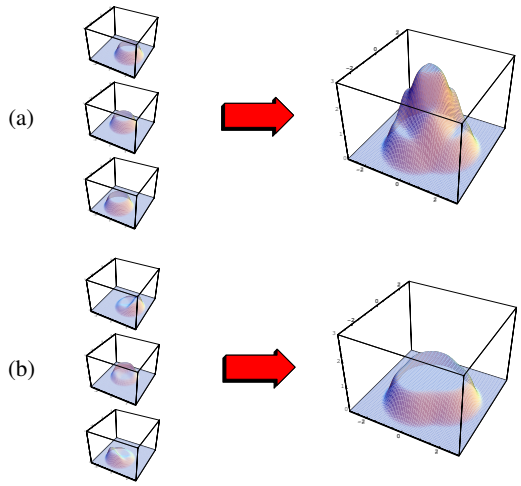



**Figure 25:** *Blending of three scalar fields. To illustrate the behaviour of our blending in this figure, we directly combine the scalar fields instead of using them to modulate a transformation. (a) Adding the scalar fields. (b) By multiplying each field with* $(1-\prod(1-\varphi_k))/\sum\varphi_k$*, the sum of the fields is normalized.*

where $p_{l_j\in[1,8]}$ are the corners of a bounding box oustide which the function $\varphi_j$ equals zero.

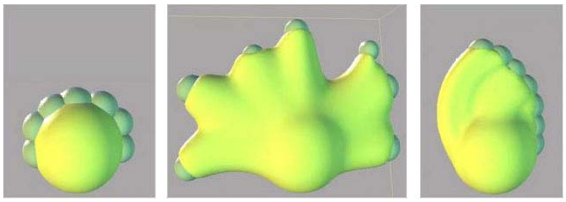


**Figure 26:** *Simultaneous sweepers*

### 4.5. Fast single sweeper

In a single tool scenario, some transformations are convenient to input by the artist: translations, non-uniform and uniform scaling and rotations. With these simple transformations, the deformations of a point is much simpler to compute, as there is a closed-form to the logarithm of simple matrices. In this section, in addition to efficient expressions for computing the number of required steps, we provide fast deformation functions for a vertex and its normal. For deforming the normal, computing the Jacobian's co-matrix is not always required: $J^C\cdot\vec{n}$ leads to much simpler expressions. Note that the normal's deformations do not preserve the normal's length. It is therefore necessary to divide the normal by its magnitude. We denote $\vec{\gamma}_k=(\gamma_x,\gamma_y,\gamma_z)^\top$ the gradient of $\varphi_k$ at p and $\vec{\gamma}$ the gradient of $\varphi$ at p

**If *M* is a translation:** The use of $\odot$ simplifiies, using translation vector $\vec{d}$. The minimum number of steps is:

$$\max_p\|\vec{\gamma}(p)\|\ \|\vec{d}\|<s \tag{54}$$

The $s$ vertex deformations are:

$$f_k(p)=p+\frac{\varphi_k(p)}{s}\vec{d} \tag{55}$$

The $s$ normal deformations are:

$$g_k(\vec{n})=\vec{n}+\frac{1}{s}(\vec{\gamma}_k\times\vec{n})\times\vec{d} \tag{56}$$

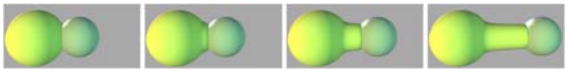


**Figure 27:** *Translation*

**If *M* is a uniform scaling operation:** Let us define the center of the scale $c$, and the scaling factor $\sigma$. The minimum number of steps is:

$$\max_p\|\vec{\gamma}(p)\|\ \sigma\log(\sigma)d_{\max}<s \tag{57}$$

where $d_{\max}$ is the largest distance between a point in the deformed area and the center c, approximated using a bounding box. The $s$ vertex deformations are:

$$f_k(p)=p+(\sigma^{\frac{\varphi_k(p)}{s}}-1)(p-c) \tag{58}$$

Let $\vec{\chi}=\frac{\log(\sigma)}{s}(p-c)$. The $s$ normal deformations are:

$$g_k(\vec{n})=\vec{n}+(\vec{\gamma}_k\times\vec{n})\times\vec{\chi} \tag{59}$$

**Figure 28:** *Scale*

**If *M* is a non-uniform scaling operation:** Let us define the center of the scale c , its direction of scale, unit vector $\vec{n}$, and its scaling factor, $\sigma$. The minimum number of steps is:

$$\max_p\|\vec{\gamma}(p)\|\ \sigma\log(\sigma)d_{\max}<s \tag{60}$$

where $d_{\max}$ is the largest distance between a point in the deformed area and the plane of normal $\vec{n}$ passing through c. The $s$ vertex deformations are:

$$f_k(p)=p+(\sigma^{\frac{\varphi_k(p)}{s}}-1)((p-c)\cdot\vec{n})\vec{n} \tag{61}$$

Let $\vec{\chi}=\frac{\log(\sigma)}{s}(p-c)$. The $s$ normal deformations are:

$$g_k(\vec{n})=\vec{n}+\sigma^{\frac{\varphi_k(p)}{s}}((\vec{v}+(\vec{v}\cdot\vec{\chi})\vec{\gamma}_k)\times\vec{n})\times\vec{v} \tag{62}$$

It is appropriate to remark here that the tool is also subject to the scale, and that the influence function $\varphi_t$ must be defined in an appropriate way, as described in Section **??**.

**If *M* is a rotation:** Let us define a rotation angle $\theta$, center of

rotation r and vector of rotation $\vec{v} = (v_x, v_y, v_z)^\top$. The minimum number of steps is:

$$\max_{p} \|\vec{\gamma}(p)\| \, \theta r_{max} < s \tag{63}$$

where $r_{max}$ is the distance between the axis of rotation and the farthest point from it, approximated using a bounding box. The $s$ vertex deformations are:

$$f_k(p) = p + (\cos\frac{\varphi_k\theta}{s} - 1)\vec{\xi} \times \vec{n} + \sin\frac{\varphi_k\theta}{s}\vec{\xi} \tag{64}$$
$$\text{where } \vec{\xi} = \vec{v} \times (p - r)$$

The $s$ normal deformations are:

$$\begin{aligned} g_k(\vec{n}) &= (\vec{n}\cdot\vec{v})\vec{v} + \vec{v} \times (\cos(h)\vec{n} \times \vec{v} - \sin(h)\vec{n}) \\ &\quad + \theta\vec{\gamma} \times (\vec{n} \times \vec{\xi} \\ &\quad + ((\cos(h) - 1)(\vec{n} \times \vec{\xi}) \cdot \vec{v} \\ &\quad + \sin(h)\vec{n} \cdot \vec{\xi})\vec{v}) \end{aligned} \tag{65}$$
$$\text{where } h = \frac{\varphi_k\theta}{s}$$

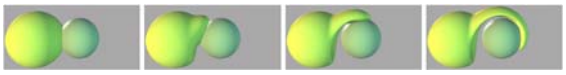


**Figure 29:** *Rotation*

### 4.6. Fast symmetric sweepers

For an operation symmetric about a plane, the transformation matrices are of the same type, thus blending them leads to simple expressions. Let us consider two tools $\varphi_0$ and $\varphi_1$. If the influence of both tools is zero at p, that is if $\varphi_0(p) = 0$ and $\varphi_1(p) = 0$, then the deformation is the identity. If the influence of one tool is zero at p, that is if $\varphi_0(p) = 0$ or $\varphi_1(p) = 0$, then the deformation equation is that of a single tool. When both influences are not zero at p, that is if $\varphi_0(p) \neq 0$ and $\varphi_1(p) \neq 0$, then the deformation induced at p by the tools' motion must be computed using Equation (51). In the rest of this section, we have simplified the blending equation for simple symmetric transformations of the same type.

**Translation:** The number of steps is:

$$\max_{p} \|\vec{\gamma}(p)\| \, (\|\vec{d}_0\| + \|\vec{d}_1\|) < s \tag{66}$$

The deformation of a point is:

$$f_k(p) = p + \frac{1}{s}(1 - \frac{\varphi_0 + \varphi_1}{\varphi_0\varphi_1})(\varphi_0\vec{d}_0 + \varphi_1\vec{d}_1) \tag{67}$$

**Rotation, scale and non-uniform scale:** The deformation of a point is:

$$f_k(p) = \exp(\frac{1}{s}(1 - \frac{\varphi_0 + \varphi_1}{\varphi_0\varphi_1})(\varphi_0 \log M_0 + \varphi_1 \log M_1)) \tag{68}$$

### 4.7. Results

Although a few simple transformations were combined (translation, uniform scale and rotation), the set of possible deformations is very high because of the arbitrary shape of the tools, and also because many tools' deformations can be blended. The shapes shown in Figure 30 were modeled in real-time in *one hour* at most, and were all made starting with a sphere.

Figures 30(a) and 30(b) show the use of the multi-tool to achieve smooth and symmetric objects. Figure 30(d) shows that sharp features can be easily modeled. Figures 30(c) and 30(i) show the advantage of foldover-free deformations, as the artist did not have to concentrate on avoiding self-intersections: our deformations do not change the topology of space and thus preserve the topology of the initial object.

## 5. Modeling with constant volume

In a non-virtual modeling context, one of the most important factors which affects the artist's technique is the amount of available material. This aspect was ignored in the previous sections. The notion of an amount of material is not only familiar to professional artists, but also to children, who may experience it with Play-Doh® at kindergarten, and to adults through everyday life experience. A shape modeling technique that preserves volume will take advantage of this, and will hopefully be genuinely intuitive to use.

### 5.1. Swirl

We define a particular case of sweeper, a *swirl*, by using a point tool c, together with a rotation of angle $\theta$ around an axis $\vec{v}$ (see Figure 31). A scalar function, $\varphi$, and a deformation are defined as before. Informally, a swirl twists space locally around axis $\vec{v}$ without compression or dilation (see proof in [ACWK04]): it preserves volume.

### 5.2. Ring of swirls

Many deformations of the above kind can be naively combined to create a more complex deformation:

$$f(p) = \left(\bigoplus_{i=0}^{n-1}(\varphi_i(p) \odot M_i)\right) \cdot p \tag{69}$$

It is important here to remark that the above blending is *not* the blending formula of simultaneous tools defined in Equation (51), and only uses simple weights. The reason for using the above simple blending equation as opposed to Equation (51) is that the latter modulates the amount of individual transformations locally, and attempting to control the volume with it would be inappropriate. We provide a convenient way for the artist to input $n$ rotations, by specification of a single translation $\vec{t}$. Let us consider $n$ points, $c_i$, on the circle of center h, and radius $r$ lying in a plane perpendicular to $\vec{t}$. To these points correspond $n$ consistently-oriented

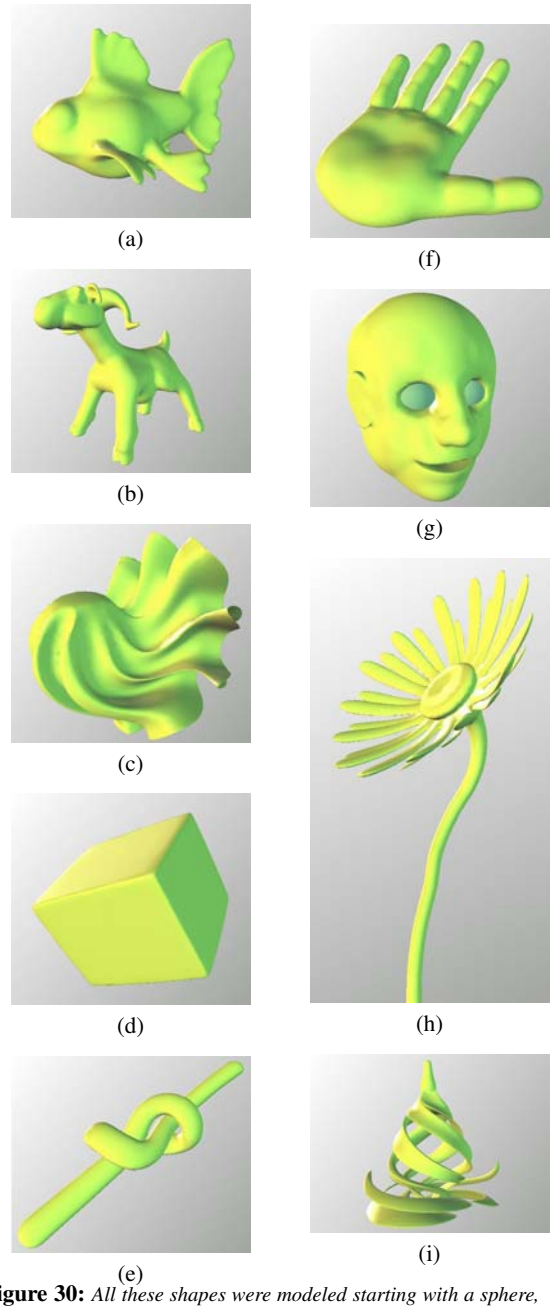



**Figure 30:** *All these shapes were modeled starting with a sphere, in at most one hour. In (c), the first modeling step was to squash the sphere into a very thin disk. In (g), eyeballs were added.*

unit tangent vectors $\vec{v}_i$ (see Figure 32). Each pair, $(c_i, \vec{v}_i)$, together with an angle, $\theta_i$, define a rotation. Along with radii of influence $\lambda_i = 2r$, we can define $n$ swirls. The radius of the circle $r$, is left to the user to choose. The following value for $\theta_i$ will transform h exactly into $h + \vec{t}$ (see justification in [ACWK04]):

$$\theta_i = \frac{2\|\vec{t}\|}{nr} \tag{70}$$

With this information, the deformation of Equation (69) is now a tool capable of transforming a point into a desired target. We show in Figure 32 the effect of the tool for different values of $n$; in practice, we use 8 swirls.

**5.2.0.1. Preserving coherency and volume** If the magnitude of the input vector $\vec{t}$ is too large, the deformation of Equation (69) will produce a self-intersecting surface, and will not preserve volume accurately. The reason for self-intersection is explained in Section 4.1. The volume is not accurately preserved because the blending operator, $\oplus$, blends the transformation matrices, and not the deformations. To correct this, it is necessary to subdivide $\vec{t}$ into smaller vectors. Thus foldovers and volume preservation are healed with the same strategy. The number of steps must be proportional to the speed and inversely proportional to the size of the tool. We use:

$$s = \max(1, \lceil 4\|\vec{t}\|/r \rceil) \tag{71}$$

As the circle sweeps space, it defines a cylinder. Thus the swirling-sweeper is made of $ns$ basic deformations. Figure 33 illustrates this decomposition applied to a shape.

### 5.3. Swirling-Sweepers

We summarize here the swirling-sweepers algorithm:

```
Input point h, translation t⃗, and radius r
Compute the number of required steps s
Compute the angle of each step, θ_i = 2||t⃗|| / nrs
for each step k from 0 to s − 1 do
  for each point p in the tool's bounding box do
    M = 0
    for each swirl i from 0 to n − 1 do
      M += φ^i_k(p) log M_{i,k}
    end for
    p = (exp M) · p
  end for
end for
```

The point $c_{ik}$ denotes the center of the $i^{\text{th}}$ swirl of the $k^{\text{th}}$ ring of swirls. For efficiency, a table of the basic-swirl centers, $c_{ik}$, and a table of the rotation matrices, $\log M_{i,k}$, are precomputed. We have a closed-form for the logarithm of the involved matrix, given in Equations (72) and (73), saving an otherwise expensive numerical approximation:

$$\begin{aligned} \vec{n} &= \theta_i \vec{v}_i \\ \vec{m} &= c_{i,k} \times \vec{n} \end{aligned} \tag{72}$$

$$\log M_{i,k} = \begin{pmatrix} 0 & -n_z & n_y & m_x \\ n_z & 0 & -n_x & m_y \\ -n_y & n_x & 0 & m_z \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{73}$$

Note that for the sake of efficiency, we handle these matrices as mere pairs of vectors, $(\vec{n},\vec{m})$. Once $M$ is computed, we use a closed-form for computing $\exp M$. Since the matrix $M$ is a weighted sum of matrices $\log M_{i,k}$, the matrix $M$ is of the form of Equation (73), and can be represented with a pair $(\vec{n}_M,\vec{m}_M)$. If $\vec{n}_M = 0$, then $\exp M$ is a translation of vector $\vec{m}_M$. Else, if the dot product $\vec{m}_M \cdot \vec{n}_M = 0$, then $\exp M$ is a rotation of center c, angle $\theta$ axis $\vec{v}$, as given by Equation (74):

$$\begin{array}{rcl} c & = & \frac{\vec{\omega}\times\vec{m}}{\|\vec{\omega}\|^2} \\ \theta & = & \|\vec{n}_M\| \\ \vec{v} & = & \vec{n}_M/\theta \end{array} \qquad (74)$$

Finally, in the remaining cases, we denote $l = \|\vec{n}_M\|$, and we use Equation (75) (see Appendix B for efficiency):

$$\exp M = I + M + \frac{1-\cos l}{l^2}M^2 + \frac{l-\sin l}{l^3}M^3 \qquad (75)$$

Symmetrical objects can be easily modeled by introducing a plane of symmetry about which the tool is reflected (see Figure 35).

### 5.4. Results

In Figure 35, we compare the shapes' volume with unit spheres on the right. The shapes volumes are respectively 101.422%, 99.993%, 101.158% and 103.633% of the initial sphere. This error is the result of accumulating smaller errors from each deformation. For instance 80 swirling-sweepers have been used to model the alien. The small errors are due to the finite number of steps, and to our choice of shape representation. The shapes shown in Figure 35 were modeled in real-time in *half an hour* at most, and were all made starting with a sphere.

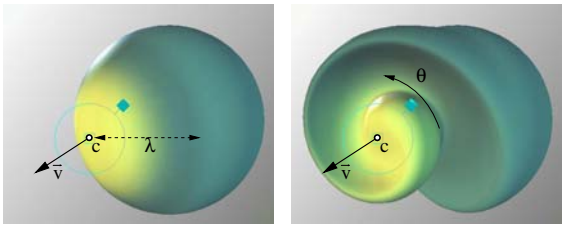



**Figure 31:** *The effect on a sphere of a swirl centered at c, with a rotation angle $\theta$ around $\vec{v}$. The two shapes have the same volume.*

## 6. A shape description

With sweepers and swirling-sweepers, shape modeling operations based on gesture can be conveniently described, while coherency and volume of the shape are maintained. By using both operation types, the artist can increased, decreased or preserved the volume of a shape (see Figure 36). Because these operation types are indepent from the shape description, several choices are available: mesh, particles, discrete grid of deformed raytracing (see [Ang05]). In the context of shape modeling, the number of deformations is possibly excessively large, and issues related to such excess have to be taken into consideration when defining a shape description. We provide in this section a shape description for interactive modeling which supports high deformation and does not break when highly stretched.

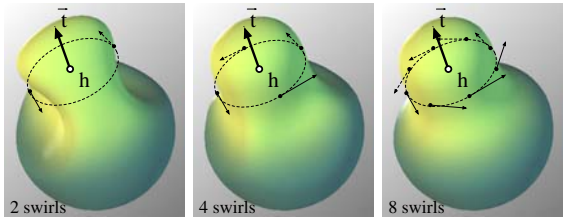



**Figure 32:** *By arranging n basic swirls in a circle, a more complex deformation is achieved. In the rightmost image: with 8 swirls, there are no visible artifacts due to the discrete number of swirls.*

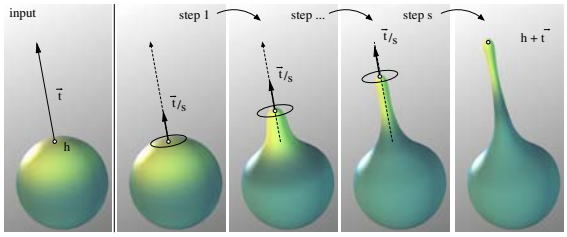



**Figure 33:** *A volume preserving deformation is obtained by decomposing a translation into circles of swirls. 3 steps have been used for this illustration. As the artist pulls the surface, the shape gets thinner. The selected point's transformation is precisely controlled.*

A simple way of representing a deformable shape is to place a set of samples on the surface of the shape: this makes the task of deforming the shape as straightforward as deforming the points on its surface. Points are discrete surface samples, and need to be somehow connected using splatting, interpolation or approximation scheme in order to display a continuous surface.

Our method uses an updated mesh, i.e. vertices connected with triangles. Connectivity provides convenient $2D$-boundary information for rendering the surface as well as surface neighborhood information, which enables the artist to define very thin membranes without having them vanish, as shown in Figure 30(c). The use of triangular "$C^0$ patches" circumvents issues related to non-regular vertices and smoothness maintenance across the boundaries that join patches. Also, current hardware handles polygons very efficiently, which is relevant to us since interactivity is among our objectives. The reader however should be aware that point-sampled geometry has recently ignited a lot of interest from researchers [PKKG03].

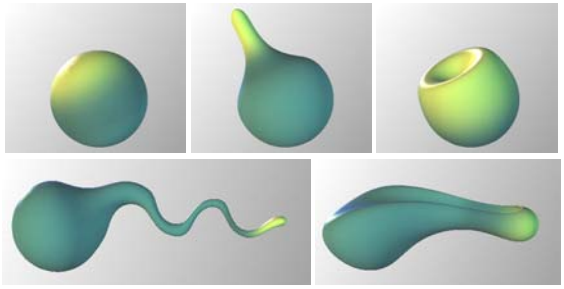


**Figure 34:** *When pushed or pulled, a sphere will inflate or deflate elsewhere.*

The possibly large number of deformations applied by an artist requires some minimum surface sampling density. Thus, the scene is initialized with a polygonal model, e.g. a sphere with a homogeneous density of nearly equilateral triangles[‖]. In order to quickly fetch the vertices to be deformed and the edges that require splitting or collapsing, these are inserted into a 3D grid. Note that this spatial limitation is not too restrictive for the artist, as our deformations allow us to translate the entire model rigidly and scale it uniformly.

To fetch the vertices that are deformed, a query is done with the tool's bounding box. Conveniently, this bounding box is also used in Eq. (49). Since the principle of our swept deformations is to subdivide the input gesture into a series of smaller ones, all the transformations applied to the vertices are bounded. To take advantage of this decomposition in steps, we apply a modified version of a more generic algorithm [GD99]. Our method requires keeping two vertices and two normals per vertex, corresponding to the previous and following state of some small step operation $f_k$. Loosely speaking, our surface-updating algorithm assumes that smooth curves run on the surface, and that the available information, namely vertices and normals, should be able to represent them. If this is not the case after deformation, then it means the surface is under-sampled. On the other hand, if an edge is well enough represented by a single sample, then it is collapsed.

Let us consider an edge $e$ defined by two vertices $(\mathrm{v}_0, \mathrm{v}_1)$ with normals $(\vec{\mathrm{n}}_0, \vec{\mathrm{n}}_1)$, and the deformed edge $e'$ defined by vertices $(\mathrm{v}'_0, \mathrm{v}'_1)$ with normals $(\vec{\mathrm{n}}'_0, \vec{\mathrm{n}}'_1)$. In addition to the conditions in [GD99] based on edge length and angle between normals, we also base the choice of splitting edge $e'$ on the error between the edge and a fictitious vertex, which belongs to a smooth curve on the surface. The fictitious vertex is used only for measuring the error, and is not a means of interpolating the vertices. If the error between the fictitious vertex and the edge is too large, the edge $e$ is split, and the new vertex and normal are deformed. On the other hand if the fictitious vertex represents the edge $e'$ well enough, then edge $e$ is collapsed, and the new vertex is deformed. We

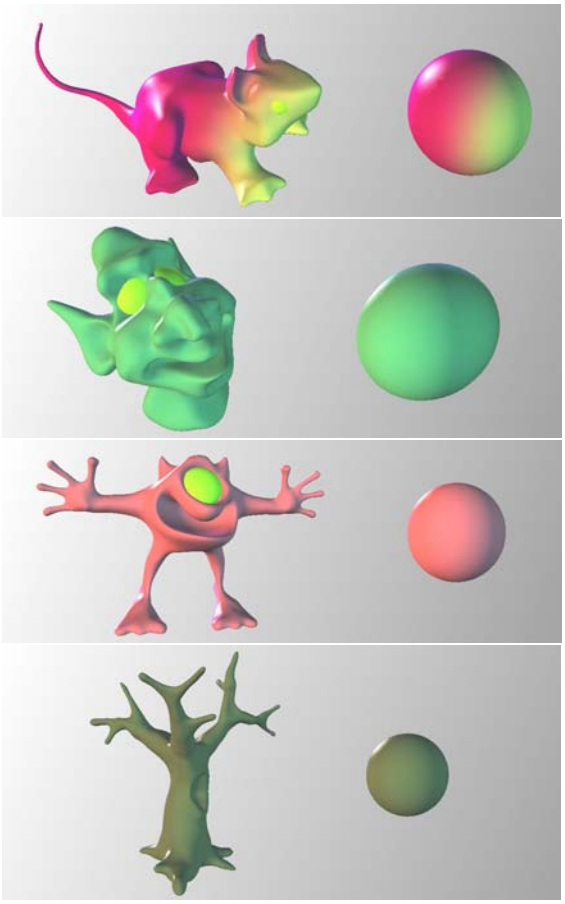


**Figure 35:** *Examples of models "sculpted" with swirling-sweepers. The mouse, the goblin, the alien and the tree have respectively 27607, 25509, 40495 and 38420 vertices. These objects were modeled in less than 30 min by one of the authors. Eyeballs have been added.*

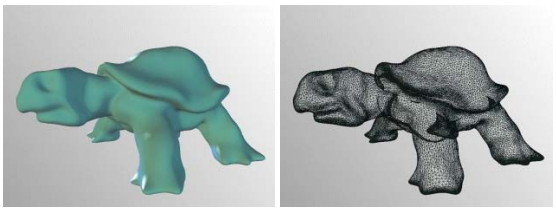


**Figure 36:** *Shape modeled with sweepers and swirling-sweepers.*

‖ A simple way to obtain an homogeneous sphere polygonization consists of starting with an icosahedron, putting all its edges longer than $h$ in a queue, splitting them and putting the pieces longer than $h$ back in the queue. Each time a split is performed, the new edges are flipped to maximize the smallest angle.

define the fictitious vertex as the mid-vertex of a $C^1$ curve, since vertices and normals only provide first order information about the surface. The following cubic polynomial curve interpolates the vertices $v'_0$ and $v'_1$ with corresponding shape tangents $\vec{t}_0$ and $\vec{t}_1$, defined below:

$$\begin{aligned} c(u) \quad = \quad & (v'_0(1+2u)+\vec{t}_0 u)(1-u)^2 + \\ & (v'_1(1+2(1-u))-\vec{t}_1(1-u))u^2 \end{aligned} \tag{76}$$

The only constraint on tangent $\vec{t}_i$ is to be perpendicular to the corresponding normal $\vec{n}_i$. The following choice defines tangents of magnitude proportional to the distance between the vertices:

$$\begin{aligned} \vec{t}_0 &= \vec{g} - \vec{g}\cdot\vec{n}'_1 \\ \vec{t}_1 &= \vec{g} - \vec{g}\cdot\vec{n}'_0 \\ \text{where} \quad \vec{g} &= v'_1 - v'_0 \end{aligned} \tag{77}$$

With the above tangents, the expression of the middle vertex simplifies:

$$c(0.5) = ( v'_0 + v'_1 + (\vec{g}\cdot\vec{n}'_0 - \vec{g}\cdot\vec{n}'_1)/4 ) \;/\; 2 \tag{78}$$

With the fictitious vertex $c(0.5)$, the tests to decide whether an edge should be split or collapsed can now be defined:

***Too-long edge:*** An edge $e'$ is too long if *at least one* of the following conditions is met:

- The edge is longer than $L_{\max}$, the size of a grid-cell. This condition keeps a minimum surface density, so that the deformation can be caught by the net of vertices if the coating thickness $\lambda_j$ is greater than $L_{\max}$.
- The distance between the fictitious vertex and the mid-vertex of $e'$ is too large (we used $L_{\max}/20$). This condition prevents the sampling from folding on itself, which would produce multiple sampling layers of the same surface.
- The angle between the normals $\vec{n}'_0$ and $\vec{n}'_1$ is larger than a constant $\theta_{\max}$. This condition keeps a minimum curvature sampling.

***Too-short edge:*** An edge $e'$ is too short if *all of* the following conditions are met:

- The edge's length is shorter than $L_{\min}$ (we used $L_{\max}/2$).
- The angle between the normals $\vec{n}'_0$ and $\vec{n}'_1$ is smaller than a constant $\theta_{\min}$.
- The distance between the fictitious vertex and the mid-vertex of $e'$ is too small (we used $L_{\min}/20$).

Also, to avoid excessively small edges, an edge is merged regardless of previous conditions if it is too small (we used $L_{\min}/20$).

We stress that the procedure for updating the mesh is applied at each small step, rather than after the user's deformation function has been applied. Because vertex displacements are bounded by the foldover-free conditions, the update of our shape description does not suffer from problems related to updating a greatly distorted triangulation. Figure 37 shows a twist on a simple U-shape. Figure 38 shows the algorithm preserving a fine triangulation only where required. Figure 39 shows the algorithm at work in a more practical situation. The procedure outline is:

```
Compute the number of steps required, s.
for each step k do
  Deform the points, and hold their previous values
  for each too-long edge do
    split the edge and deform the new point.
  end for
  for each too-short edge do
    collapse the edge and deform the new point.
  end for
end for
```

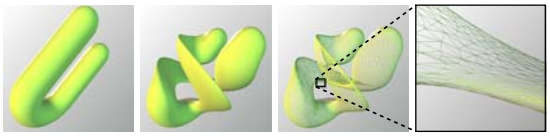


**Figure 37:** *Example of our mesh-updating algorithm on a highly twisted U-shape. The close-up shows a sharp feature, with finer elongated triangles.*

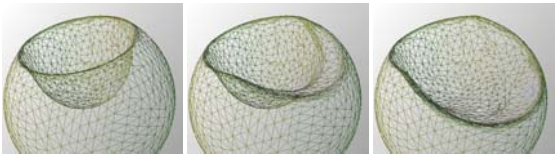


**Figure 38:** *Behaviour of our mesh-updating algorithm on an already punched sphere. The decimation acompanying the second punch simplifies the small triangle of the first punch. The tool has been removed for better visualization.*

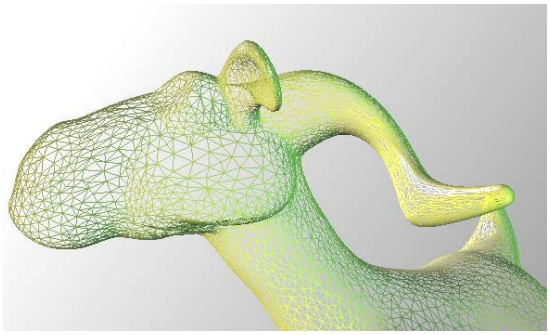


**Figure 39:** *Close-up of the goat. Notice the large triangles on the cheek and the fine ones on the ear. The initial shape is a sphere.*

***Limitation:*** With the updated mesh method, we choose to ignore the history of functions applied to the shape by the artist. Thus we "collapse" the history by freezing it in the current shape. To explain the major consequence of this, let us suppose the scene at a time $t_k$, such that the shape $S(t_k)$ is shown to the user. The next deformation produced by the artist with the mouse is function $f_{t_k \mapsto t_{k+1}}$, and all the

mesh refinements and simplifications are performed in $S(t_k)$. This is however an approximation: ideally the last operation should be concatenated to the history of deformations, and the whole series should be applied to the initial shape $S(t_0)$, i.e. $\bigcirc_{i=0}^{n} f_{t_i \mapsto t_{i+1}}$ should be applied to each new vertex. This would however become more and more time consuming as the sequence of deformations gets longer ($n$ gets larger), and the modeling software would eventually become unusable.

## 7. Conclusion

Sweepers, is a framework for defining swept deformation operation for shape modeling. It permits the description of a family of shape operations based on gesture between the artist and the mathematics describing the shape, and enables an artist to handle shapes in a more efficient way than modifying directly a shape's mathematical description. Because sweepers are foldover-free, they maintain easily a shape coherency. Swirling-sweepers is a type of swept-deformation for describing shape modeling operations that preserve implicitly the shape's volume. Subjectively, swirling-sweepers is the most effective modeling technique defined in the sweepers framework. Further work on volume-preservation outlines that there is in fact a link between swirling-sweepers and fluid mechanics [AN]. The separation of the shape's operations and the shape's description leads to the exploration of alternative ways to describe a shape's surface or volume for rendering. While our proposed method is sufficient in a wide range of situations, more research should be done in this area.

## 8. Acknowledgments

Many thanks to Marie-Paule Cani, Geoff Wyvill, Scott King and Brendan McCane for contributing to the work on sweepers and swirling-sweepers presented in this chapter.

## Appendix A: Linear Combination of Transformations

The multiplication operator $\odot$ and addition operator $\oplus$, which behave essentially like $\cdot$ and $+$ for scalars. The operator $\odot$ is defined as $\alpha \odot M = \exp(\alpha \log M)$ and the operator $\oplus$ is defined as $M \oplus N = \exp(\log M + \log N)$. The following series defines the exponential of a matrix:

$$\exp M = \mathrm{I} + M + \frac{1}{2}M^2 + \frac{1}{6}M^3 \cdots = \sum_{k=0}^{\infty} \frac{M^k}{k!} \tag{79}$$

The logarithm of a matrix is defined as an inverse of the exponential, as follows:

$$\log(\mathrm{I} - M) = -M - \frac{1}{2}M^2 - \frac{1}{3}M^3 \cdots = -\sum_{k=1}^{\infty} \frac{M^k}{k} \tag{80}$$

In a similar that repeting $+$ can be expressed with $\sum$, the repetition of $\oplus$ can be expressed as follows:

$$\bigoplus_{i=1}^{n} M_i = M_1 \oplus M_2 \oplus \cdots \oplus M_n \tag{81}$$

## Appendix B: Exponential

Applying the exponential of the matrix to a point does not require to compute the exponential of the matrix explicitly. Let us define the matrix $M$ with a pair of vectors, $(\vec{n}, \vec{m})$.

$$\begin{aligned} \exp(M) \cdot \mathrm{p} &= \mathrm{p} + (\vec{\mathrm{m}} + \vec{\mathrm{n}} \times \mathrm{p})b + (\tfrac{\vec{\mathrm{n}} \times \vec{\mathrm{m}}}{l^2} - \mathrm{p})a \\ &\quad + \vec{\mathrm{n}}((\vec{\mathrm{n}} \cdot \mathrm{p})a + (\vec{\mathrm{n}} \cdot \vec{\mathrm{m}})(1-b))\tfrac{1}{l^2} \\ \text{where } l &= \|\vec{\mathrm{n}}\| \\ a &= 1 - \cos(l) \\ b &= \tfrac{\sin(l)}{l} \end{aligned} \tag{82}$$

## References

[ACWK04] ANGELIDIS A., CANI M.-P., WYVILL G., KING S.: Swirling-sweepers: Constant-volume modeling. In *Pacific Graphics 2004* (October 2004), IEEE, pp. 10–15. Best paper award at PG04. 1, 16, 17

[Ale02] ALEXA M.: Linear combination of transformations. In *Proceedings of SIGGRAPH'02* (July 2002), vol. 21(3) of *ACM Transactions on Graphics, Annual Conference Series*, ACM, ACM Press / ACM SIGGRAPH, pp. 380–387. 13

[AN] ANGELIDIS A., NEYRET F.: Simulation of smoke based on vortex filament primitives. In *Symposium on Computer Animation'05*. http://www-evasion.imag.fr/Publications/2005/AN05. 21

[Ang05] ANGELIDIS A.: *Shape Modeling by Swept Space Deformation*. PhD thesis, University of Otago, 2005. 14, 18

[AW04] ANGELIDIS A., WYVILL G.: Animated sweepers: Keyframed swept deformations. In *CGI'04* (July 2004), IEEE, pp. 320–326. 1

[AWC04] ANGELIDIS A., WYVILL G., CANI M.-P.: Sweepers: Swept user-defined tools for modeling by deformation. In *Proceedings of Shape Modeling and Applications* (June 2004), IEEE, pp. 63–73. Best paper award at SMI04. 1

[Bar84] BARR A. H.: Global and local deformations of solid primitives. In *Proceedings of SIGGRAPH'84* (July 1984), vol. 18(3) of *Computer Graphics Proceedings, Annual Conference Series*, ACM, ACM Press / ACM SIGGRAPH, pp. 21–30. 2, 12

[BB91] BORREL P., BECHMANN D.: Deformation of n-dimensional objects. In *Proceedings of the first symposium on Solid modeling foundations and CAD/CAM applications* (1991), pp. 351–369. 7

[BBB*97] BAJAJ C., BLINN J., BLOOMENTHAL J., CANI-GASCUEL M.-P., ROCKWOOD A., WYVILL B., WYVILL G.: *Introduction to Implicit Surfaces*. Morgan-Kaufmann, 1997. 10

[Bla94] BLANC C.: A generic implementation of axial procedural deformation techniques. In *Graphics Gems* (1994), vol. 5, pp. 249–256. Academic Press. 2, 12

[Blo90] BLOOMENTHAL J.: Calculation of reference frames along a space curve. *Graphics gems* (1990), 567–571. 3

[BR94] BORREL P., RAPPOPORT A.: Simple constrained deformations for geometric modeling and interactive design. In *ACM Transactions on Graphics* (April 1994), vol. 13(2), pp. 137–155. 7

[Coq90] COQUILLART S.: Extended free-form deformation: A sculpturing tool for 3d geometric modeling. In *Proceedings of SIGGRAPH'90* (July/August 1990), vol. 24(4) of *Computer Graphics Proceedings, Annual Conference Series*, ACM, ACM Press / ACM SIGGRAPH, pp. 187–195. 6, 12

[CR94] CHANG Y.-K., ROCKWOOD A.: A generalized de Casteljau approach to 3d free-form deformation. In *Proceedings of SIGGRAPH'94* (July 1994), Computer Graphics Proceedings, Annual Conference Series, ACM, ACM Press / ACM SIGGRAPH, pp. 257–260. 3, 12

[Cre99] CRESPIN B.: Implicit free-form deformations. In *Proceedings of the Fourth International Workshop on Implicit Surfaces* (1999), pp. 17–24. 3, 4, 10

[Dec96] DECAUDIN P.: Geometric deformation by merging a 3d object with a simple shape. In *Graphics Interface* (May 1996), pp. 55–60. 9, 10, 12

[Ebe01] EBERLY D. H.: *3D Game Engine Design*. Morgan Kaufmann, 2001. 13

[Far90] FARIN G.: Surfaces over Dirichlet tessellations. *Computer Aided Geometric Design 7(1-4)* (June 1990), 281–292. 8

[GD99] GAIN J. E., DODGSON N. A.: Adaptive refinement and decimation under free-form deformation. *Eurographics'99 7*, 4 (April 1999), 13–15. 19

[GD01] GAIN J. E., DODGSON N. A.: Preventing self-intersection under free-form deformation. *IEEE Transactions on Visualization and Computer Graphics 7*, 4 (October-December 2001), 289–298. 8, 12

[Gla89] GLASSNER A. (Ed.): *An Introduction to Ray traycing*. Morgan Kaufmann, 1989. 2

[GP89] GRIESSMAIR J., PURGATHOFER W.: Deformation of solids with trivariate b-splines. In *Eurographics Conference Proceedings* (1989), Elsevier Science, pp. 137–148. 6

[Gri] GRIFFITHS D. J.: *Introduction to Electrodynamics*, third ed. Prentice Hall. 6

[Gué01] GUÉZIEC A. P.: "Meshsweeper": Dynamic point-to-polygonal-mesh distance and applications. *IEEE Transactions on Visualization and Computer Graphics 7*, 1 (January/March 2001), 47–61. 14

[HHK92] HSU W. M., HUGHES J. F., KAUFMAN H.: Direct manipulation of free-form deformations. In *Proceedings of SIGGRAPH'92* (July 1992), vol. 26(2) of *Computer Graphics Proceedings, Annual Conference Series*, ACM, ACM Press / ACM SIGGRAPH, pp. 177–184. 7, 12

[HML99] HIROTA G., MAHESHWARI R., LIN M. C.: Fast volume-preserving free form deformation using multi-level optimization. In *Proceedings of the fifth ACM symposium on Solid modeling and applications* (June 1999), ACM, pp. 234–245. 9, 12

[HQ04] HUA J., QIN H.: Scalar-field-guided adaptive shape deformation and animation. *The Visual Computer 1*, 1 (April 2004), 47–66. 11, 12

[KY97] KURZION Y., YAGEL R.: Interactive space deformation with hardware assisted rendering. *IEEE Computer Graphics and Applications 17(5)* (September/October 1997), 66–77. 9

[LCJ94] LAZARUS F., COQUILLART S., JANCÃĹNE P.: Axial deformations: an intuitive deformation technique. In *Computer-Aided Design* (1994), vol. 26, 8, pp. 607–613. 3, 12

[LKG*03] LLAMAS I., KIM B., GARGUS J., ROSSIGNAC J., SHAW C. D.: Twister: A space-warp operator for the two-handed editing of 3d shapes. In *SIGGRAPH* (August 2003), vol. 22(3) of *ACM Transactions on Graphics, Annual Conference Series*, ACM, pp. 663–668. 11, 12

[MJ96] MACCRACKEN R. A., JOY K. I.: Free-form deformations with lattices of arbitrary topology. In *Proceedings of SIGGRAPH'96* (August 1996), Computer Graphics Proceedings,

Annual Conference Series, ACM, ACM Press / ACM SIGGRAPH, pp. 181–188. 6, 7, 12

[MMT97] MOCCOZET L., MAGNENAT-THALMANN N.: Dirichlet free-form deformation and their application to hand simulation. In *Computer Animation'97* (June 1997), pp. 93–102. 8, 12

[MW01] MASON D., WYVILL G.: Blendeforming: Ray traceable localized foldover-free space deformation. In *Proceedings of Computer Graphics International (CGI)* (July 2001), pp. 183–190. 5, 12

[PKKG03] PAULY M., KEISER R., KOBBELT L. P., GROSS M.: Shape modeling with point-sampled geometry. In *Proceedings of SIGGRAPH'03* (July 2003), vol. 22(3), ACM, pp. 641–650. 18

[Rut90] RUTHERFORD A.: *Vectors, Tensors and the Basic Equations of Fluid Mechanics*. Dover, 1990. 6

[SE04] SCHEIN S., ELBER G.: Discontinuous free form deformations. In *Proceedings of Pacific Graphics* (October 2004), IEEE, pp. 227–236. 9

[SF98] SINGH K., FIUME E.: Wires: a geometric deformation technique. In *Computer graphics, Proceedings of SIGGRAPH'98* (July 1998), Computer Graphics Proceedings, Annual Conference Series, ACM, ACM Press / ACM SIGGRAPH, pp. 405–414. 4, 5, 14

[SP86] SEDERBERG T., PARRY S.: Free-form deformation of solid geometric models. In *Proceedings of SIGGRAPH'86* (August 1986), vol. 20(4) of *Computer Graphics Proceedings, Annual Conference Series*, ACM, ACM Press / ACM SIGGRAPH, pp. 151–160. 6, 12

[Wei04] WEISSTEIN E.: Lagrange multipliers. From Mathworld – A Wolfram Web Ressource `mathworld.wolfram.com/LagrangeMultipliers.html`, 2004. 12

# Modeling by Sketching

Marie-Paule Cani

GRAVIR lab (IMAG-INRIA) and INP Grenoble

**Abstract**

*This chapter shows that sketching in 2D and annotating 3D models can be very effective tools for designing free-form shapes. Easier to use than 3D sculpting systems for arbitrary users, sketching systems take advantage of the natural expressiveness of drawings and of the rapidity with which they convey the idea of a shape. Indeed, an interesting problem is the way 3D information is inferred by 2D drawings. We show that, according to the application, different good solutions can be set up for this problem. We detail three specific examples: a sketching and annotation system that uses 2D sketches to generate 3D drawings that can be viewed from any direction and enable to quickly exchange ideas at the early stage of design; A quick prototyping system for 3D shapes that allows the user to model arbitrary shapes by successively drawing parts of them under different viewing angles, an implicit representation being used for easily combining the different parts; a sketching system dedicated to a specific application: the design of virtual garments. Here, a priori knowledge specific to the application is used to infer the missing 3D information.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling Geometric algorithms; I.3.6 [Computer Graphics]: Methodology and Techniques Interaction Techniques J.5 [Computer Applications]: Arts and Humanities Fine arts

**Keywords:** drawing, sketching, annotation, modeling, interfaces, implicit representations, clothing design

## 1. Introduction

Standard modeling systems rely heavily on 3D widget-based interfaces to ease the user task of specifying 3D actions using 2D input devices. The user visualizes the task by looking at the 3D scene from several viewpoints, such as the traditional blueprint projection planes. The underlying rationale is that a 3D task is better performed using 3D tools [CSH*92]. This is true; however, these tools are generally controlled using 2D input devices, and this results in sophisticated systems with a somewhat non-intuitive interface.

Other solutions are based on user-interaction in 3D, using specific devices such as a 3D mouse, a 3D glove, or a phantom desktop. Some of these systems were presented in the chapter *Volumetric sculpting* of this tutorial. They rely on the ability of the user to sculpt a shape directly in 3D. However, few people sculpt in real life. Even though most of us used to play with modeling clay such as Play-Doh at an early age, shaping objects in 3D generally requires specialized skills. Using computers does not make the process much simpler.

In contrast, most people draw. We sketch, doodle, and scribble to keep track of our thoughts or communicate ideas to others. We consider drawing as an alternative to writing, because it is often faster and more concise to describe three-dimensional shapes and spatial relationships with two-dimensional lines than with words. Drawing tools are simple and their dexterous use constitutes a wealth of common knowledge most people have acquired since kindergarten. However, this know-how is seldom used in computer graphics for anything but two-dimensional vector or pixel-based drawing applications.

This chapter studies the use of sketching as an interface for 3D modeling. Depending on the applica-

tion, different solutions for inferring a 3D shape from a sketch can be though off. After a quick review of prior work in section 2, we cover this diversity by detailing three specific examples:

- Section 3 presents a sketching and annotation system that generates 3D illustrations to be used for exchanging ideas on 3D shapes at the early stage of design. Although no standard 3D shape reconstruction is performed, the third dimension is inferred so that the drawn object can be visualized under different viewing angles, as if a non-photorealistic rendering was computed from its surface.
- Section 4 describes a quick prototyping system that outputs 3D meshes from sketches. This system allows the user to model arbitrary shapes or any topology by iteratively sketching new parts of the current object under different viewing angles. The solution is based on the blending of convolution surfaces inferred from each sketch. This system is well adapted to the quick design of smooth, volumetric objects such as cartoon characters.
- Section 6 covers a specific application of sketching: the design of virtual clothes from their contour lines drawn over a front view of a character. Here, a priori knowledge specific to the application is used to infer the missing 3D information from a single sketch. The system outputs the surface of the virtual garments in the form of open 3D meshes, already adequately positioned over the characters model.

## 2. Related Work

Since the invention of Sketchpad by Ivan Sutherland in 1963 [Sut63], many computer graphics systems have used drawing metaphors for 3D shape modeling, most often requiring the user to draw polyhedral surfaces in wireframe [LS96, SC04].

Eggli et al. alleviate the unintuitive input by using simple drawing techniques that have an unambiguous interpretation in three dimensions [EBE95].

Departing from these approaches, SKETCH and Teddy demonstrate that gesture-based interfaces are powerful and intuitive tools for 3D model design [ZHH96, IMT99]. More recent work uses either variational implicit surfaces [KHR02] or a volumetric representation [ONNI03] for generating the surface.

In the systems we just mentioned, drawing is used for describing boundaries and shape features using strokes, but not as a way of modeling the relief of surfaces. Williams proposes to create a height field by directly painting the luminance value corresponding to a given elevation [Wil90]. Bourguignon [BCCD04] presents a modeling system which takes advantage of both 2D shape boundary and texture to generate a displacement map. The latter is either used to create new surface patches or to push or pull the vertices of an existing surfaces on which the drawing was done.

Three interesting solutions to the modeling by drawing problem are found in commercial systems. Artisan is a Maya software toolset with a common painting metaphor [Ali04], inspired from interactive texturing interfaces [HH90]. The Sculpt Polygons Tool pushes, pulls, or smoothes the surface when the user paints it with the corresponding brush. ZBrush greatly expands the mesh editing operations of Maya's Artisan [Pix04]. The user first sculpts a rough shape envelope using ZSpheres, and then refines it thanks to its subdivision surface multiresolution capabilities. Finally, SketchUp is a thorough architecture design system, inspired by SKETCH [@La04].

Sketches are not only used to create 3D geometry, but also for positioning objects with respect to each other. Igarashi [IH02] describes a sketch-based method for positioning garment patterns over a 3D body. The user cannot, however, directly sketch the desired garment, a problem which is tackled by one of the systems presented in this chapter.

## 3. Sketching at an early stage of design

This section presents an interactive sketching system, first introduced in [BCD01], that infers 3D drawings that are similar to an expressive rendering of a 3D object from 2D sketches. The central idea is that reconstructing a surface is not necessary at the early stage of design: the user is rather attempting to visualize and communicate the ideas he has in mind. He may also like to use sketching to suggest modifications on an existing 3D shape, by annotating a 3D model.

Even in 3D, we think that strokes are an excellent way to indicate the presence of a surface silhouette: several neighbouring strokes reinforce the presence of a surface in the viewer's mind, while attenuated strokes may indicate imprecise contours or even hidden parts.

To enable the user to view stroke-based sketches from multiple viewpoints, we interpret 2D silhouette strokes as curves, and use a curvature estimation scheme to infer a local surface around the original stroke. This mechanism permits efficient stroke-based rendering of the silhouette from multiple viewpoints. In addition to stroke deformations, this includes variation of intensity according to the viewing angle, since the precision of the inferred local surface decreases when we move away from the initial viewpoint. It also includes relative stroke occlusion, and additive blending of neighbouring strokes in the image.

Apart from silhouette strokes, our system also provides line strokes that represent 1D elements. These

have the ability to remain at a fixed position in 3D while still being occluded by surfaces inferred using silhouette strokes. They can be used to add 1D detail to the sketches, such as the arrow symbols in the example of annotation.

Because strokes have to be positioned in space, we present an interface for 3D stroke input. The user always draws on a 2D plane which is embedded in space. This plane is most often the screen plane, selected by changing the viewpoint. The depth location of this plane can be controlled either explicitly via the user interface or implicitly by drawing onto an existing object. The user may also draw strokes that are not in the screen plane, but that join two separate objects.

### 3.1. Drawing and rendering 3D strokes

Two kinds of strokes are used in our system: line strokes that represent 1D details, and silhouette strokes that represent the contour of a surface.

For line strokes, we use a Bzier space curve for compact representation. These strokes are rendered using GL line primitives and behave consistently with respect to occlusion.

Silhouette strokes in 3D are more involved: a silhouette smoothly deforms when the view-point changes. Contrary to line strokes, a silhouette stroke is not located at a fixed space position. It may rather be seen as a 3D curve that "slides" across the surface that generates it. Our system infers the simplest surface, i.e. the same local curvature in 3D as that observed in 2D. For this we rely on the differential geometry properties of the user-drawn stroke, generating a local surface around it. But the degree of validity of this surface decreases when the camera moves. Therefore, we decrease the intensity of the silhouette as the point of view gets farther from the initial viewpoint. This allows the user to either correct or reinforce the attenuated stroke by drawing the silhouette again from the current viewpoint.

#### Local surface estimation from 2D input

Since the inferred local surface will be based on the initial stroke curvature, the first step of our method is to compute the variations of this curvature along each 2D silhouette stroke drawn by the user.

Each 2D silhouette stroke segment is first fit to a piecewise cubic Bzier curve, each control point being associated with a given value of the parameter $u$ along the curve. For each parameter value $u$ associated with a control point $V(x(u), y(u))$, we find the center of curvature $C(\xi(u), \eta(u))$ using:

$$\xi = x - \frac{\dot{y}(\dot{x}^2 + \dot{y}^2)}{\dot{x}\ddot{y} - \dot{y}\ddot{x}} \qquad \eta = y + \frac{\dot{x}(\dot{x}^2 + \dot{y}^2)}{\dot{x}\ddot{y} - \dot{y}\ddot{x}},$$

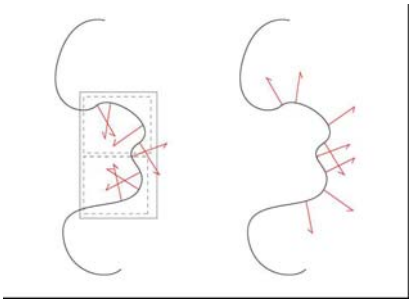


**Figure 1:** *Processing vectors of curvature. Curvature vectors are clamped using bounding boxes and their orientation is switched when necessary.*

where $\dot{x}$ and $\ddot{x}$ are first and second derivatives of $x$ in $u$. Therefore, we obtain a curvature vector between a point on curve at parameter $u$ and its associated center of curvature $C$ (see Figure 1). We will be using these curvature vectors to reconstruct local 3D surface properties. However, if the stroke is completely flat, the norm of the curvature vector (i.e. the radius of curvature) goes to infinity; the method we present next solves this problem.

In order to infer a plausible surface in all cases, we use a heuristic based on the curve's length to limit the radius of curvature. One way of looking at this process is that we are attempting to fit circles along the stroke curve. Thus, if we encounter many inflection points, the circles fitted should be smaller, and the local surface should be narrower; in contrast, if the curve has few inflection points, the local surface generated should be broader.

To achieve this, we construct axis-aligned bounding boxes of the control polygon of the curve between each pair of inflection points (see Figure 1). Inflection points can be found easily since we are dealing with a well-defined piecewise cubic Bzier curve. We discard bounding boxes which are either too small or too close to the curve extremities. If the norm of the curvature vector is larger than a certain fraction of the largest dimension of the bounding box computed previously

it is clamped to this value. We use a fraction value at most equal to $\frac{1}{2}$, which gives a length equal to the radius of a perfect circle stroke.

We also impose a consistent in/out orientation of the curve based on the orientation of the curvature vectors in the first bounding box computed, thus implicitly considering initial user input as giving correct orientation. This intuitive choice corresponds to the intent of the user most of the time. If not, a button in the UI can be used to invert all the curvature vectors along the stroke.

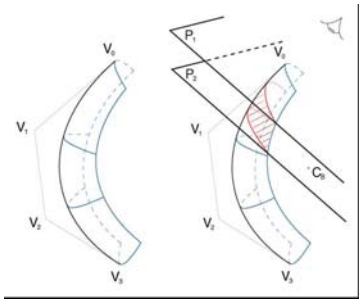



**Figure 2:** *Left: Infering a local surface from a 2D stroke. Right: Rendering a slice of this model to create a new silhouette from another viewpoint.*

From these 2D strokes we infer local surface properties, which are then used to create a 3D stroke representation. Each center of curvature embedded in the drawing plane is considered as the center of a circle in a plane perpendicular to the drawing plane and passing by the corresponding control point (see Figure 2, left). We consider an arc of $\frac{2\pi}{3}$ radians for each circle, thus defining a piecewise Bzier surface by moving each control point on its circle arc. This piecewise tensor product surface is quadratic in one dimension, corresponding to a good approximation to circle arcs, and cubic in the other, which corresponds to the stroke curve.

In practice, the inferred surface will only be used for generating a probable silhouette when the viewing angle changes slightly. If more information is needed about the 3D surface geometry, the contour will have to be redrawn by the user at another viewpoint.

### Rendering in 3D

Given a local surface estimation, our goal is to display the initial stroke from a new viewpoint. When the viewpoint changes, we expect the stroke to change its shape, as a true silhouette curve would do. We also expect its color to change, blending progressively into the background color to indicate the degree of confidence we have in this silhouette estimation. Recall that we want our system to be interactive, which imposes an additional computational constraint. In what follows, the term "local surface" corresponds to the polygonal approximation of the local surface estimation of the stroke.

The solution we adopt is to generate a fast but approximate silhouette based on the local surface generated as described above. We simply render a "slice" of the local surface that lies between two additional clipping planes, parallel to the camera plane and located in front of and behind the barycenter of the centers of curvature (see Figure 2, right). The distance between clipping planes depends on the stroke width value we have chosen. This ensures silhouette-like shape modification, with minimal computational overhead.

Initially, we render all geometry other than the silhouette strokes (for example the house in Figure 3). Therefore, the depth and color buffers are correctly filled with respect to this geometry. In the next step, we use different elements to display the silhouette strokes and to perform stroke occlusion. This was implemented using a multipass algorithm.

**Figure 3:** *A single landscaping sketch, which can also be seen as an annotation of an existing 3D model; the two different views are automatically generated by our system.*

To represent the confidence in the surface around the initial stroke we apply a "stroke texture" as an *alpha* texture to the local surface. This confidence is maximum at the initial stroke position and minimum at left and right extremities of local surface. We use a Gaussian distribution that progressively blends the stroke color into the background color for modeling this confidence function. As a result, the stroke becomes less intense as we move away from the initial viewpoint. This blending also allows two different strokes to reinforce each other by superposition, which corresponds to the behavior of traditional ink brush drawings.

In addition to occlusion by other geometry, we also need to handle occlusion by strokes. This required a slightly more sophisticated process, since we do not want local surfaces to produce hard occlusion (such as that created by a depth buffer) but rather to softly occlude using the background color.

Details on the multipass algorithm and on the different drawing styles can be found in [BCD01].

### 3.2. Interface for Drawing: Stroke Positioning

A drawing session using our system is in many ways similar to 2D drawing. A designer either starts with an empty space, or, in the case of annotation with a pre-existing 3D model.

For strokes drawn in empty space, we project onto a reference plane, parallel to camera plane and containing the world origin (it is possible to choose a fixed offset relative to this position).

Once such parts of the drawing have been created, we can use the existing entities to position the curves in space. More precisely, if at the beginning or at the end of a 2D stroke the pointer is on an existing object, we use this object to determine a new projection plane. We obtain the depth of the point selected by a simple picking. The picked object can correspond to a geometric object or to (the local surface of) a stroke. There are three possibilities:

- If only the beginning of the stroke is on an object, we project the stroke on a plane parallel to camera plane, which contains the selected point.
- If the beginning and the end of the stroke are on an object, we interpolate depth values found at the two extremities by using the parameter $u$ of the piecewise cubic Bzier curve. Each control point is projected on a plane parallel to the camera plane and located at the corresponding depth.
- If it happens that the stroke extremities are in empty space, it is projected on the same plane as the previous stroke, except if the trackball has been moved. In this case, the reference plane is used.

### 3.3. Results

**Illustration in 3D**

**Figure 4:** *Example of artistic illustration. Three views of the same sketch area are shown.*

Figure 4 shows a 3D illustration designed with our system. Most strokes are silhouette strokes. They have been rendered on a textured background so that the local surface occluder appears as a "fill" effect. Each illustration is shown from several different points of view, showing the effects of occlusion, varying stroke lightness, and silhouettes deformations.

**Annotation of a 3D scene**

Another application of 3D drawing is to use our system for annotating an existing 3D model. Figure 3 shows how the 3D model of a house is annotated to give an idea of a landscape around it. Interactive annotation (which can include using 1D strokes to draw symbols such as arrows) can also be used for educational purposes, or in brainstorming sessions of collaborative design.

**"Guided design"**

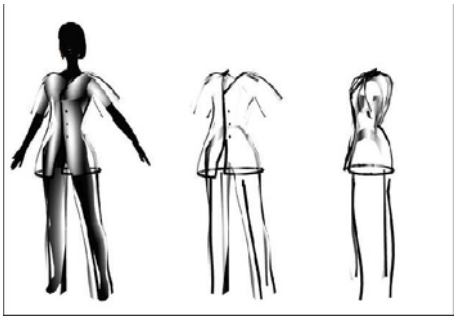


**Figure 5:** *Using a 3D model as a guide can be useful in fashion applications.*

The idea of this third application is to load a 3D model and use it as a guide. When the drawing is completed, the model is removed. A good example of this kind of application is drawing clothes for fashion. A 3D model is used to obtain the body proportions (see Figure 5).

## 4. Sketching for the quick prototyping of 3D shapes

The purpose of the system presented in this section (see also [ABCG05]) is to allow a very quick prototyping of 3D volumetric objects of any geometry and topology genius, while offering an interface as simple as pencil and paper. The modeling process iterates the following steps until modeling is complete:

1. The user changes the viewpoint and draws one or several strokes
2. The strokes are interpreted to reconstruct a 3D object part
3. This part is added to the current object (or subtracted if the current mode is carving).

### 4.1. Overview of the users input

The user draws a contour on the graphic tablet using the digital pen. As the user draws a stroke, its thickness and colour intensity vary proportionally with the

pressure on the digital pen, as to imitate the irregular density and thickness of the strokes produced by a real pen. Several strokes accumulated in the same pixel result in a darker colour for that pixel. The other end of the pen is used as an eraser. As long as the stroke has not been reconstructed, the user is free to erase and modify it. This way the user's input is allowed to be noisy and irregular, as it is naturally on paper. The Undo command is also available.

Once the contour has been completed the user presses the digital pen against the tablet. This produces the 3D reconstruction of an object part (see Figure 6). The shape is reconstructed in such a manner that the projection of the shape on the screen fits the contour that has been drawn by the user.

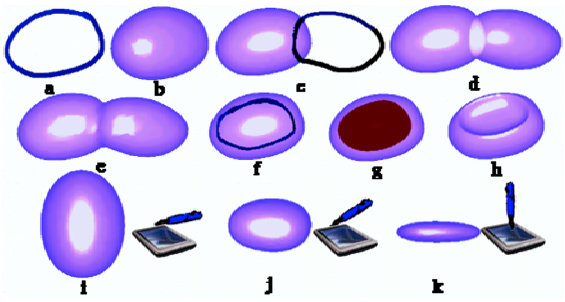



**Figure 6:** *(a), (b) Creating a part. (c),(d),(e). Adding a part to an object. (f ),(g),(h) Carving. (i),(j),(k) Thickness control (side view).*

The interface is the same for extending or carving a shape as for creation. The first surface point hit by the user, which must be a point on the object, gives the depth of the shape to be constructed. When the stroke is complete, the user presses the stylus if he wishes to add the shape to the existing object, or the eraser (at the opposite pen's end) of he wishes to carve it into the object. See Figure 6 (c),(d),(e) and (f),(g),(h).

The user controls the thickness of the shape as follows: if the pen is held orthogonal to the tablet while pressing to reconstruct, the shape is a flat one (see Figure 6 (k)), otherwise it is a thick one (see Figure 6 (i)), the thickness being dictated by the pen's bend.

The user can model smaller details by zooming to get closer to the object. The large object parts will smoothly blend with each other, while the small details (e.g. eyes, nose of a character) will have a sharper blending.

## 5. Generation of a 3D shape

### 5.1. Basic ideas

A skeleton, in the form of a graph of branching polylines and polygons, is first extracted from the user's sketch. The 3D shape is then defined as a convolution surface generated by this skeleton. Our formulation has the advantage of requiring no optimisation step for fitting the 3D shape to the 2D contours. This yields interactive performances and avoids any non-desired oscillation of the reconstructed surface.

The subsequent 2D strokes are used to infer new object parts, which are combined with the existing shape using CSG operators, thus representing the final shape as a hierarchy of blended implicit primitives.

### 5.2. Skeleton from 2D contour

An overview of our algorithm is given in Figure 7.

When the stylus pressure indicates that the drawing is finished, the strokes image is recovered as a 2D bitmap. The image is then compressed and reduced in size using a pixel averaging technique, in order to smooth the input. This also reduces the amount of computation for the skeleton.

In order to perform the skeleton extraction we iteratively construct a connected pixels skeleton, invariant to object geometric transformations and allowing the inverse transform. The pixel skeleton is then sampled and the polygons are triangulated in order to obtain a graph of interconnected segments and triangles.

More precisely, we proceed as follows:

1. The object is separated from the background (see Figure 7 (b)).
2. The Weighted Distance Transform (WDT) is computed (Figure 7 (c)). The result is shown in Figure 7(c). We then detect the set of Centres of Maximal Discs (CMDs) which will be used in skeleton extraction.
3. The object is thinned iteratively keeping all the CMD's. This produces a connected pixel graph (Figure 7 (d)), which is pruned to eliminate the insignicant branches. This graph will provide the skeleton segments.
4. The second pass is performed on the initial WDT image and thins the object without preserving the CMD, the result being an eroded shape of the object (Figure 7 (e)). This image will be used to compute the skeleton's polygons.
5. The image from Figure 7 (e) is subtracted from (d) and the result is a collection of free-form lines of one pixel width. Each line is re-sampled to reduce the number of segments (Figure 7 (f)).

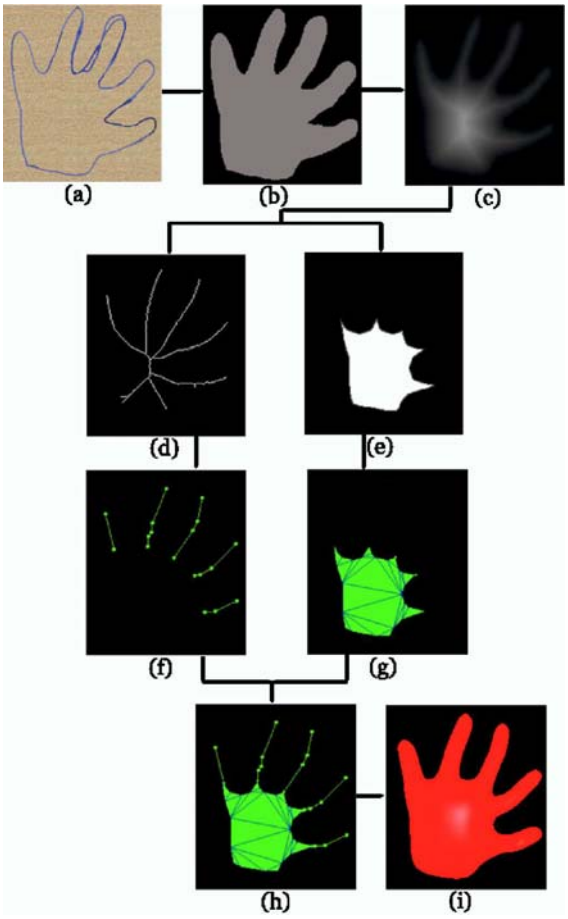



**Figure 7:** *Algorithm overview : (a) User strokes. (b) Identification of the object boundaries. (c) WDT transform. (d) Iteratively thinning with preservation of the CMDs. (e) Iteratively thinning without preservation of the CMDs. (f ) Sampling the segments. (g) Sampling and triangulating the polygons. (h) Final skeleton graph. (i) Final convolution surface.*

6. To compute polygons, the object contour is recovered from the eroded image (Figure 7 (e)) and sampled (Figure 7 (g)). If the image has several disconnected object parts, then each part produces a polygon. The polygons are split into triangles using constrained Delaunay triangulation (Figure 7 (g)).
7. The graph connections are be computed from the two images in Figure 7 (f) and (g). This produces the final skeleton graph (Figure 7(h)).

### 5.3. Creating a 3D shape

The graph previously obtained is used to generate a convolution surface with a varying radius [AC02]. We control it by setting weights at every skeleton vertex. The weights are assigned according to the distance from the vertex to the contour, which can be directly read using the WDT image.

An exact fit of the user drawn contour would require an optimisation step, which might cause the surface to oscillate. We prefer to obtain an approximate fit and a smooth non-oscillating surface. For this purpose, we have set the weights manually for a large number of cases, considering the fact that they vary with the distance to the contour, but also with the distance from the user's point of view. We then computed a polynomial function which best interpolated all the found values and use it to automatically assign weights without any adjustment afterwards. This technique yields very satisfying results in practice.

The implicit shape that reconstructs a part of the object is obtained by the convolution of the skeleton graph with a convolution kernel. A shape's capacity of blending with the previously existing shapes is described by its blending function. We use the functions defined by Barthe [BGC98]. We set the blending parameter according to the level of detail. Small details produce sharper blending. The unblended surface is then polygonized and displayed. The blending hierarchy is automatically updated.

The control of the objects thickness orthogonal to the view direction is achieved by providing every implicit shape with a scaling factor. The function is composed with a scale in the depth direction. The scaling factor to be assigned to the current drawn shape, is computed from the sum of the two angles formed by the pen with X and Y tablet axes. The angle is measured when the pen been pressed to indicate reconstruction.

Some steps of the modeling of a cartoon character are depicted on Figure 8.

### 5.4. Adaptive modeling and rendering

The level of detail of the skeleton remains constant in the image space, so it increases with respect to the 3D shape when the user zooms in. The level of detail determines the blending parameters, the skeleton weights and the size of the polygonization cell for the shape to be reconstructed. The polygonization of the current object part is computed and displayed immediately, while a process in the background computes the final surface polygonization, taking the blending between surface components into account. Meanwhile the user continues his modeling task. If the polygonization is available, the final mesh is displayed, replacing the two meshes of the disconnected components. If the object has been updated, the polygoniz-

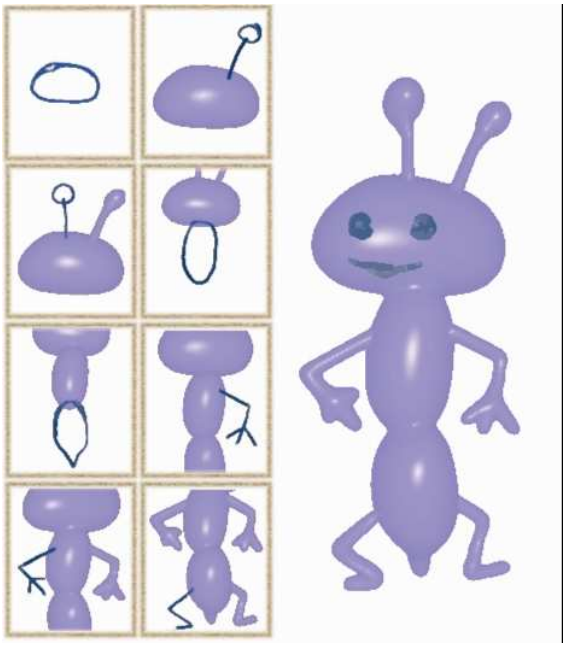


**Figure 8:** *Modeling a cartoon character with our system.*

ing process is notified and it restarts the computation. This allows maintaining interactive rates and a quick application response during the modeling process so that the user feels free to pursue his modeling activity.

### 5.5. Results and conclusions

As our results in Figure 9 show, our system allows non-expert users to generate a wide variety of free form shapes with an easy to use sketch-based interface.

Each objects was entirely modeled within our system in less than 5mn each. Only three strokes was necessary to create each one of the birds, with symmetry enabled for the wings and legs. The small details of the objects are well preserved due to our adaptive blending functions. For example, the sun's eyes and mouth are small details compared to the face but they are well preserved by the blending. The shape may have an arbitrary topological genius (ex. chairs, teapot) and can be carved (teapot, mugs). The applications of our system can be fun and education, but also story boarding for films making (ex. cartoons, as in Figure 9), where the scenario writer is not necessarily a 3D designer.

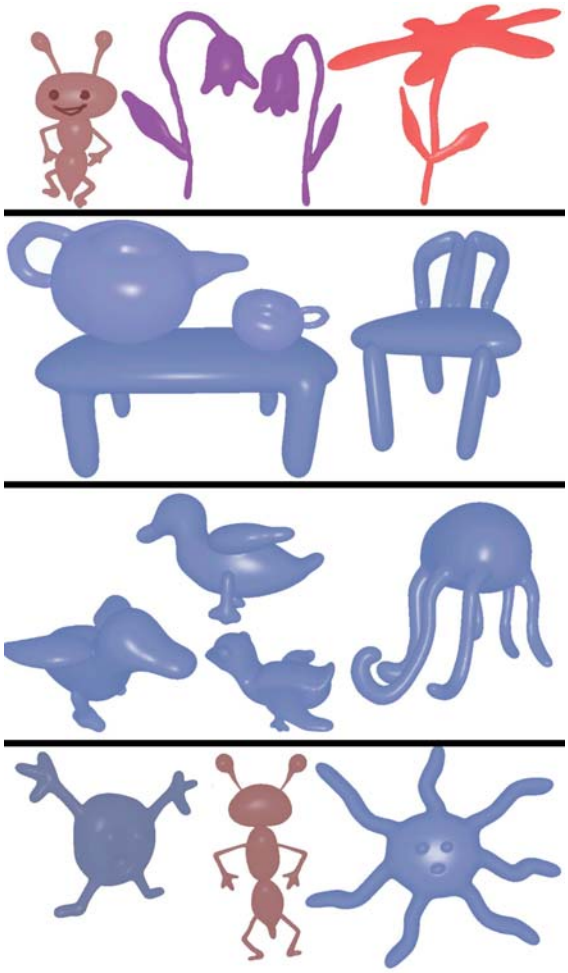


**Figure 9:** *Objects modeled with our system. The user took 2 to 5 minutes for modeling each object and used 3 to 9 strokes, each of them modeling a different object part.*

## 6. A sketching system for garment design

Dressing virtual characters is usually a very tedious process, since it involves defining and positioning 2D cloth patterns, stitching them and tuning physically-based parameters and running a simulation to get the rest shapes of the garments. Although sketching is a usual tool for fashion designers and gives very quickly an idea of the 3D garment one is thinking of, it was not exploited, up to know, for digital cloth design.

This section presents a method, first introduced in [TCH04], for reconstructing the 3D geometry of a virtual garment from a 2D sketch. As in the guided design example of section 3.3 the user sketches the

**Figure 10:** *(left) The user has drawn a few lines to indicate the shape of the skirt; the corner-detector has detects a breakpoint that the user does not want. The user will make a deletion gesture (a curve in the shape of an $\alpha$ enclosing the mistaken point) to delete it. (right) Here is the surface inferred by the system after the user asks for a reconstruction.*

garment directly on the 3D virtual actor body model. The method presented in here outputs a full 3D geometry for the garment. The resulting system can thus be used to model and position virtual clothes to be used in character animation. This system shows an example where a priori knowledge specific to an application is used to infer 3D geometry. This allows to create a 3D garment from a single 2D sketch.

### 6.1. Sketch-based interface

#### Typical user interaction

To give a sense of the system's performance, we describe a typical interaction, in which a user sketches a skirt on a female model. The user first (see figure 10(a)) draws a line across the waist, indicating the top of the skirt, and then a line down the side, indicating the silhouette of the skirt, then a line across the bottom in a vee-shape indicating that he wants the front of the skirt to dip down, and finally the last side. A simple corner-detection process is applied to break the sketch into parts; one extra corner is detected by accident (at the bottom of the vee) and the user can delete it with a deletion gesture. He could also add new breakpoints as well, but none are necessary. Breakpoints play an important role in the 3D positioning process, since they determine the global 3D position of the cloth with respect to the body. The way they are used is detailed in Section 6.3. The two lines on the sides are classified as *silhouettes* (lines that do not cros the body), and the others are classified as *border lines* (lines that do cross the body), as shown in the figure.

Now the user asks to see the garment inferred by the system; a surface matching the drawn constraints, but adapted to the shape of the underlying form (look near the waistine, for instance) appears (see figure 10(b)).

#### Gestural interface components

The user's marks are interpreted as gestures, with the default being the construction of silhouette and border line segments. Other gestures add breakpoints for the classification process, delete breakpoints, delete a segment or an entire chain of segments, and clear all segments, as shown schematically in figure 11. (Our gestures are similar to those of Tsang et al. [TBSR04]).

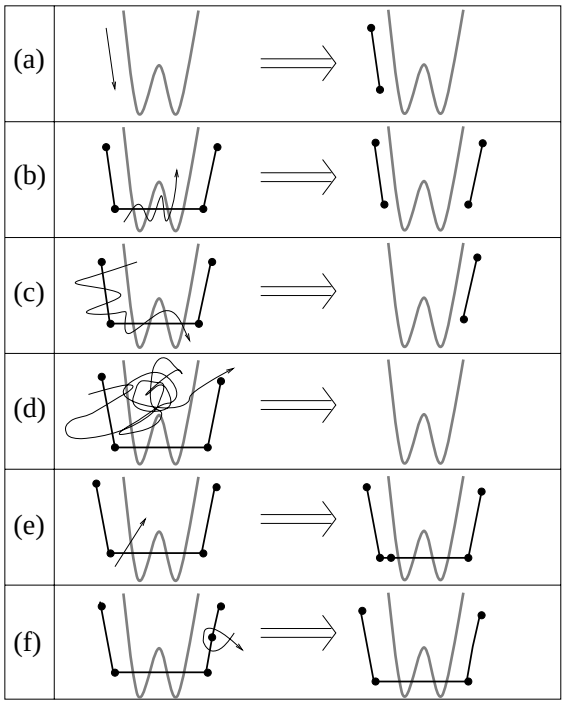



**Figure 11:** *The gestures in the interface; the model is drawn in thick lines, prior strokes as medium lines with dots at breakpoints, and new strokes with thin lines; the arrowhead on the new-stroke lines is not actually drawn by the user, but merely indicates the direction in which the stroke is drawn. (a) adding a segment; (b) deleting a segment (stroke must intersect the segment at least five times); (c) deleting several segments (the stroke must intersect more than one segment, and intersect segments at least five times. If the stroke intersects segments from two different chains, both chains are deleted entirely.); (d) clearing all segments (the stroke must intersect some segment and intersect itself at least 40 times) (e) adding a breakpoint (f) deleting a breakpoint (the stroke must intersect the segments on either side of the breakpoint, and intersect itself once).*

### 6.2. Interpreting sketches of garments: basic ideas

For the sake of explaining the process, we'll assume that the character is aligned with the $xy$-plane, viewed

along the $z$ direction. The user draws the contours of the 2D projection of the garment in the $(x, y)$ plane over the rendered character model. From these, we need to infer the $z$-information at every point of the contour and the interior of the garment.

Clearly this problem is under-constrained; fortunately, by considering the special nature of clothing, we can generate a plausible guess of the user's intentions. In particular, silhouettes of clothing not only indicate points where the tangent plane to the cloth contain the view direction; they usually occur as the clothing passes around the body, so we can estimate the $z$-depth of a silhouette edge as being the $z$-depth of the nearest point on the body (or, for a body placed fairly symmetrically about the $xy$-plane, simply use $z = 0$ as a quicker estimate). Moreover, the distance from a silhouette to the body helps us to infer the distance elsewhere, since a cloth which tightly fits the body in the $(x, y)$ plane will also tend to fit it in the $z$ direction, while a loose cloth will tend to float everywhere.

**Overview of the algorithm**

Our algorithm, whose different steps will be detailed in the following sections, develops as follows:

1. Process the 2D sketch of the garment:
   - Find high-curvature points (breakpoints) that split the contours into segments;
   - Let user add and/or delete breakpoints.
   - Classify segments of curve between breakpoints into border lines (which cross the character's body) or silhouette lines.
2. Infer the 3D position of contour lines:
   - For each breakpoint that does not lie over body, find distance to body, $d$, and set the point's depth, $z$, to the depth of the closest point on the body.
   - For each silhouette curve segment, interpolate $z$ linearly over the interior of the segment, and use interpolated $z$-values to compute $d$-values (distance to the body in the 3D space) over the interior of the segment.
   - For each border line, interpolate $d$ over interior linearly to establish a desired distance to the model for each point on the line;
3. Generate the interior shape of the garment:
   - Create a mesh consisting of points within the 2D simple closed curve that the user has drawn, sampled on a rectangular grid in the $(x, y)$ plane.
   - Extend the values of $d$, which are known on the boundary of this grid, over the interior.
   - for each interior grid point $(x, y)$, trace from the viewpoint to determine the first value of $z$ for which the distance from $(x, y, z)$ to the body is the associated value of $d$.
   - Adjust this tentative assignment of $z$ values to take into account the surface tension of the garment between two limbs of the character.
   - Tesselate the grid with triangle, clipped to the boundary curves, and render the triangles.

**Pre-computing a distance field**

To accelerate steps 2 and 3 of the algorithm above, a distance field to the character's model is pre-computed when the model is loaded: for each point of a 3D grid around the model, we determine and store the distance to the nearest point of the model.

The distance field will be used each time we need to find the $z$ coordinate to assign to a point $p(x_0, y_0)$ so that is lies at a given distance from the model. This can easily be done by stepping along the ray $R(z) = (x_0, y_0, z)$ and stopping when the adequate distance value is reached (we interpolate tri-linearly to estimate distances for non-grid points). When this computation is performed during a sweeping procedure, the stepping starts at the $z$ value found at the previous pixel, which ensures the spatial coherence of the result. Else, the process starts near the view-point, since the part of the garment we are reconstructing is laying between the viewpoint and the character model.

The quality of the results depends directly on the resolution of the 3D grid storing the distance field, as does computation time. The size of the 3D grid is user-configurable, but we have generally used a $32 \times 32 \times 32$ grid.

### 6.3. Processing of contour lines

**2D processing**

First, one must classify the parts of the user's sketch. As the user sketches, a new line that starts or ends near (within a few pixels of) an endpoint of an already-sketched line is assumed to connect to it. When the sketch is complete (i.e., forms a simple closed curve in the plane), we classify the parts:

- First the sketched lines are broken into segments by detecting points of high (2D) curvature (breakpoints) (this is actually done on-the-fly, as the user draws the strokes).
- Each segment is classified as a silhouette or border line: border lines are ones whose projection meets the projection of the body in the $xy$-plane, silhouettes are the others. The mask of the bodys projection is pre-computed and stored in a buffer well call

the *body mask*, in order to make this classification efficient.
- The resulting segmentation is visible to the user, who may choose, if necessary, to add or delete breakpoints indicating segment boundaries, after which segments are re-classified.

**Distance and $z$-value at breakpoints**

Breakpoints that are located over the body model (which is tested using the body mask) are used to indicate regions of the cloth that fit very tightly to the body. They are thus assigned a zero distance to the model, and their $z$ value is set to the bodys $z$ at this specific $(x, y)$ location.

To assign a distance value $d$ to a breakpoint that does not lie over the body, we step along the ray from the eye in the direction of the breakpoint's projection into the $xy$-plane, checking distance values in the distance field data structure as we go, and find the $z$-value at which this distance is minimized. We assign to the breakpoint the discovered $z$ and $d$ values, thus positioning the breakpoint in 3D.

**Line positioning in 3D**

Our aim is to use the 3D position of the breakpoints we just computed to roughly position the garment in 3D, while the garments shape will mostly be inferred from distances to the body along the sketch silhouettes.

To position silhouette lines in 3D, we interpolate $z$ linearly along the edge between the two breakpoints that form the extremities of the silhouette. We then set the $d$-values for interior points of the silhouette from those stored in the pre-computed distance field.

Having established the values of $z$ and $d$ along silhouette edges, we need to extend this assignment to the border lines. We do this in the simplest possible way: we assign $d$ linearly along each border line. Thus, for example, in the skirt shown above, the $d$-values at the two ends of the waistline are both small, so the $d$-value for the entire waistline will be small, while the $d$-values for the ends of the hemline are quite large, so the values along the remainder of the hemline will be large too.

### 6.4. 3D Reconstruction of the garment's surface

**Using distance to guess surface position**

As for the contour lines, the interpolation of distances to the body will be our main clue for inferring the 3D position of the interior of the garment. The first process thus consists into propagating distance values inside the garment. This is done in the following way:

The 2D closed contour lines of the garment are used to generate a $(x, y)$ buffer (sized to the bounding box of the sketch) in which each pixel is assigned one of the values 'in', 'out' or 'border', according to its position with respect to the contour. The distance values already computed along the silhouette and border

lines are assigned to the border pixels. The distances for the 'in' pixels are initialized to the mean value of distances along the contour.

Then, distance information is propagated inside the buffer by applying several iterations of a standard smoothing filter, which successively recomputes each value as an evenly weighted sum (with coefficients summing to 1) of its current value and those of its "in" or "boundary" orthogonal or diagonal neighbors. The iterations stop when the maximum difference between values obtained after two successive iterations is under a threshold, or if a maximum number of steps has been reached.

A mesh, where each node corresponds to the centre of a pixel, is associated to the 'in' region of the grid, and tessellated into triangles. This mesh is extended to the 'border' pixels, with the difference that the $(x, y)$ position of the nodes are then moved form the centre of the pixel to the closest point from the centre of the contour line. The resulting mesh thus closely fits the sketched contours in the $(x, y)$ plane.

We then sweep in the 2D grid for computing $z$ values at mesh nodes: the $z$ value is computed by stepping along a ray in the $z$ direction, starting at the $z$ value we already assigned for a neighbouring point, and taking the $z$ that corresponds to the desired distance value, as stored in the pre-computed distance field.

**Mimicking the cloth's tension**

The previous computation gives a first guess of the garment's 3D position, but still results in artefacts in regions located between two limbs of the character: due to surface tension, a cloth should not tightly fit the limbs in the in-between region (as in figure 12, top), but rather smoothly interpolate the limb's larges $z$ value, due to its surface tension.

To achieve this, we first erode the 2D body mask of a proportion that increase with the underlying $d$ value (see figure 12, bottom). We then use a series of Bezier curves in horizontal planes to interpolate the $z$ values for the pixels in-between. We chose horizontal gaps because of the structure of the human body: for an upright human (or most other mammals), gaps between portions of the body are more likely to be bounded by body on the left and right than to be bounded above and below.

To maintain the smoothness of the garment surface

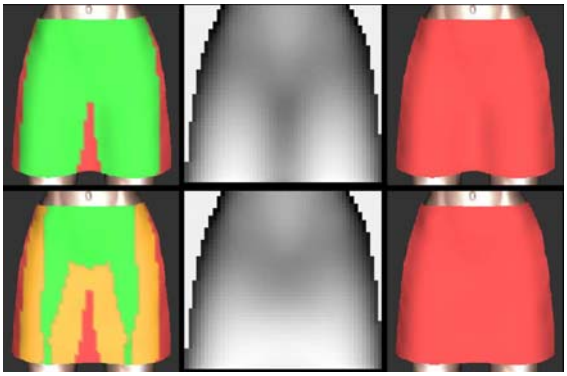


**Figure 12:** *Top: The use of distance to the body for inferring 3D position: (left) buffer storing propagated distance information; (right) the resulting garment surface. Note the undesired concavity between the character's legs. Bottom: An improved garment generated after mimicking the cloths tension.*

near the re-computed region, distances values are extracted from the new $z$ values and the distance field. Some distance propagation iterations are performed again in 2D, before re-computing the $z$ values everywhere as was done previously.

We finally add a smoothing step on the $z$ value in order to get a smoother shape for the parts of the cloth that float far from the characters model. This is done computing a smoothed version of the z-buffer from the application of a standard smoothing filter, and then by taking a weighed mean, at each pixel, of the old and smoothed $z$-values, the coefficients depending on the distance to the model at this pixel.

Finally, the triangles of the standard diagonally-subdivided mesh are used as the basis for the mesh that we render.

### 6.5. Results and future work

The examples presented in figure 13 were drawn in no more than 10 minute each. They include simple clothes such as skirts, trousers and shirts, but also less standard ones such as

In these examples, only the front part of a garment was reconstructed. To infer the shape of a full garment, we could use symmetry constraints around each limb to infer silhouettes for the invisible parts, and ask the user to sketch the border lines for the back view, for instance.

We'd also like to allow the user to draw folds, and take them into account when reconstructing the 3D

**Figure 13:** *Some 3D virtual garments generated from a single 2D sketch.*

geometry, so that even for a closely-fitting garment, there can be extra material, as in a pleated skirt.

## 7. Conclusion

This chapter has demonstrated that sketching can be a rich, intuitive, and very efficient way to create 3D shapes. The different usages of sketching go from the early stages of design, where it easily conveys imprecise ideas, to the generation of a surface ready to use for animation, as in the clothing design example. The methods for inferring 3D from sketches and for representing the created shape were very different for the three applications we covered. Many other good solutions, some of which dedicated to a very specific application were already proposed (we did not cover architectural design, which is a good example of such application domain). Others are still to be invented,

so we hope that this presentation will generate new work in the area.

**Acknowledgments**

Many thanks to David Bourguignon, George Drettakis, Anca Alexe, Loic Barthe, Emmanuel Turquin and John Hughes for their work on the sketching systems presented in this chapter.

**References**

[ABCG05] Alexe A., Barthe L., Cani M.-P., Gaildrat V.: Shape modeling by sketching using convolution surfaces. In *Submitted to publication* (april 2005).

[AC02] Angelidis A., Cani M.-P.: Adaptive implicit modeling using subdivision curves and surfaces as skeletons. In *Solid Modelling and Applications* (june 2002), ACM. Saarbrucken, Germany.

[Ali04] Alias: Maya 6.0, 3D animation and effects software, 2004.

[BCCD04] Bourguignon D., Chaine R., Cani M.-P., Drettakis G.: Relief: A modeling by drawing tool. In *First Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBM'04)* (Grenoble, France, sep 2004), Hughes J., Jorge J., (Eds.), Eurographics, pp. 151–160.

[BCD01] Bourguignon D., Cani M.-P., Drettakis G.: Drawing for illustration and annotation in 3d. In *Computer Graphics Forum* (sep 2001), Chalmers A., Rhyne T.-M., (Eds.), vol. 20 of *EUROGRAPHICS Conference Proceedings*, EUROGRAPHICS, Blackwell Publishers, pp. C114–C122.

[BGC98] Barthe L., Gaildrat V., Caubet R.: Combining implicit surfaces with soft blending in a csg tree. In *CSG Conference series* (april 1998).

[CSH*92] Conner D. B., Snibbe S. S., Herndon K. P., Robbins D. C., Zeleznik R. C., van Dam A.: Three-dimensional widgets. In *Proceedings of the Second ACM Symposium on Interactive 3D Graphics* (1992), pp. 183–188.

[EBE95] Eggli L., Brüderlin B. D., Elber G.: Sketching as a solid modeling tool. In *Proceedings of the Third ACM Symposium on Solid Modeling and Applications* (1995), ACM Press, New York, pp. 313–322.

[HH90] Hanrahan P., Haeberli P.: Direct WYSIWYG painting and texturing on 3D shapes. In *Proceedings of ACM SIGGRAPH 90* (1990), Addison-Wesley, Boston, Massachusetts, pp. 215–223.

[IH02] Igarashi T., Hughes J. F.: Clothing manipulation. In *Proceedings of the 15th annual ACM symposium on User interface software and technology* (2002), ACM Press, pp. 91–100.

[IMT99] Igarashi T., Matsuoka S., Tanaka H.: Teddy: A sketching interface for 3D freeform design. In *Proceedings of ACM SIGGRAPH 99* (1999), Addison-Wesley, Boston, Massachusetts, pp. 409–416.

[KHR02] Karpenko O., Hughes J. F., Raskar R.: Free-form sketching with variational implicit surfaces. *Computer Graphics Forum 21*, 3 (2002), 585–594.

[@La04] @Last Software: SketchUp 4.0, Intuitive and accessible 3D design tool, 2004.

[LS96] Lipson H., Shpitalni M.: Optimization-based reconstruction of a 3D object from a single freehand line drawing. *Computer-aided Design 28*, 8 (1996), 651–663.

[ONNI03] Owada S., Nielsen F., Nakazawa K., Igarashi T.: A sketching interface for modeling the internal structures of 3D shapes. In *Proceedings of Third International Symposium on Smart Graphics* (2003), Springer-Verlag, Berlin, pp. 49–57.

[Pix04] Pixologic: ZBrush 2.0, 2D and 3D painting, texturing and sculpting tool, 2004.

[SC04] Shesh A., Chen B.: SMARTPAPER: An interactive and user friendly sketching system. *Computer Graphics Forum (to appear)* (2004).

[Sut63] Sutherland I. E.: Sketchpad: A man-machine graphical communication system. In *Proceedings AFIPS Spring Joint Computer Conference* (1963), vol. 23, American Federation of Information Processing Societies Press, pp. 329–346.

[TBSR04] Tsang S., Balakrishnan R., Singh K., Ranjan A.: A suggestive interface for image guided 3d sketchng. In *Proceedings of CHI 2004, April 24–29, Vienna, Austria* (2004), pp. 591–598.

[TCH04] Turquin E., Cani M.-P., Hughes J.: Sketching garments for virtual characters. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling* (august 2004), Hughes J. F., Jorge J. A., (Eds.), Eurographics.

[Wil90] Williams L.: 3D paint. In *Proceedings of the First ACM Symposium on Interactive 3D Graphics* (1990), ACM Press, New York, pp. 225–234.

[ZHH96] Zeleznik R. C., Herndon K. P., Hughes J. F.: SKETCH: An interface for sketching 3D scenes. In *Proceedings of ACM SIGGRAPH 96* (1996), Addison-Wesley, Boston, Massachusetts, pp. 163–170.