SIGGRAPH 2006 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
Professor, Faculty of Electrical Engineering & Computer Science
Technische Universität Berlin
marc@cs.tu-berlin.de

Alexis Angelidis
Postdoctoral Fellow, Department of Computer Science
University of Toronto
silex@dgp.toronto.edu

Marie-Paule Cani
Professor, Department of Computer Science
Co-director, GRAVIR
Institut National Polytechnique de Grenoble (INPG)
Marie-Paule.Cani@imag.fr

Sarah Frisken
Professor, Department of Computer Science
Tufts University
frisken@cs.tufts.edu

Karan Singh
Associate Professor, Department of Computer Science
University of Toronto
karan@cs.toronot.edu

Steven Schkolne
Faculty of Critical Studies and Integrated Media
California Institute of the Arts
steven@schkolne.com

Denis Zorin
Associate Professor, Department of Computer Science
New York University
dzorin@mrl.nyu.edu

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. Many new modeling techniques successfully use these paradigms for interactive design of digital shapes. Advanced geometric modeling representations and algorithms are an essential foundation for this type of tools. The course covers the gamut including fundamental mathematical representations of shape, efficient algorithms, interaction paradigms and specialized hardware user interface devices, with presentations unified by a strong emphasis on the use of each topic for interactive modeling applications. 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 apply recent research in interactive shape modeling applications.

Course Agenda and Contents:

8:30 – 9:30 Introduction & Motivation -- Conceptual Shape Design (Singh)
Pages 3 – 9

9:30 – 10:15 Mathematical representations of shape for modeling (Zorin)

10:15 – 10:30 Morning break

10:30 – 11:30 Global space & Free form deformations (Angelidis)
Pages 10 - 29

11:30 – 12:15 Multiresolution modeling (Zorin)
Pages 30 - 50

12:15 – 1:45 Lunch break

1:45 – 2:45 Mesh editing based on discrete Laplace and Possion models (Alexa)
Pages 51 - 59

2:45 – 3:30 Volumetric sculpting (Frisken)
Pages 60 - 66

3:30 – 3:45 Coffee break

3:45 – 4:45 Towards 'virtual clay' (Cani)
Pages 67 - 83

4:45 – 5:30 Gesture-based shape modeling (Schkolne)
Pages 84 - 93

For slide sets and more information please see http://www.interactiveshapemodeling.net/S2006

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.

CR Categories: I.3.3 [Computer Graphics]: Geometric modeling, User Interaction

Keywords: Shape modeling, User Interaction

1 Introduction

We shall not cease from exploring, and the end of all our exploring,
will be to arrive where we started, and know the place for the first time. -*T.S.Eliot*.

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

*e-mail: karan@dgp.toronto.edu

such that the user is oblivious of the underlying mathematical surface representation. [Singh 1999] 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 [Igarashi et al. 1999] and SKETCH [Zeleznik et al. 1996] 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 [Igarashi et al. 1999],[Tsang et al. 2004], 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 [Balakrishnan et al. 1999], a skill that allows them to lay out designs on large surfaces using tape of varying thickness and tension (see Figure 4).

Figure 1: Curvature continuous surfaces

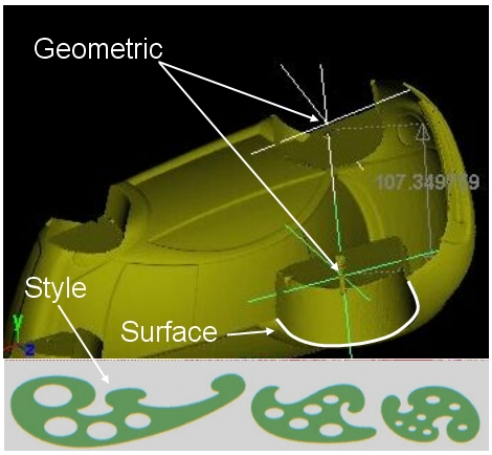


Figure 2: Automotive design constraints

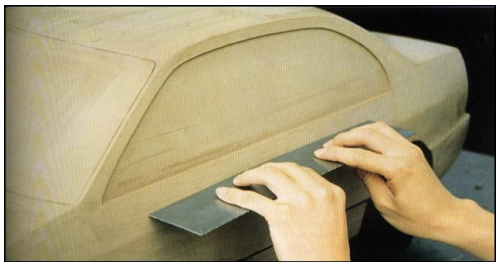


Figure 3: Editing a physical model prototype

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.

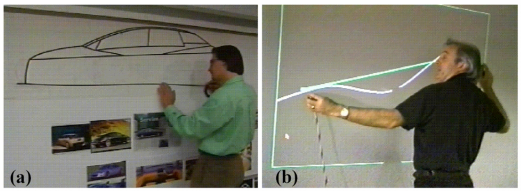


Figure 4: Digital Tape Drawing [Balakrishnan et al. 1999]

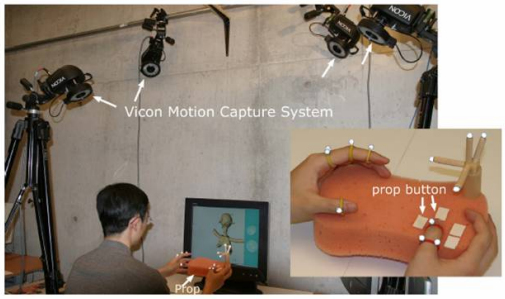


Figure 5: Sculpting with motion capture [Sheng 2004]

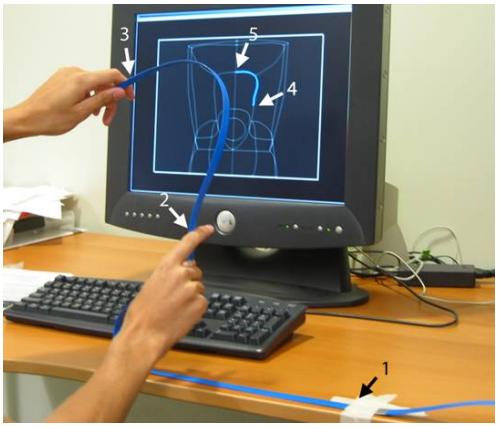


Figure 6: Manipulating curves with ShapeTape [Grossman et al. 2003]

- **Curvature continuous shapes**
  Automobile surfaces display a high degree of continuity, 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 [W. Buxton 2000] 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 [T. Grossman 2002],[Llamas et al. 2003], motion capture systems [Sheng 2004] (see Figure 5) and ShapeTape [Grossman et al. 2003] (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 [Curless] and the data structured using reverse engineering techniques [V. Krishnamurthy 1996]. Creating 3D models from 2D sketches is far a trickier problem [Eggli et al. 1997],[Lowe 1991] 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 [Tsang et al. 2004]. The success of a technique is likely to be in its

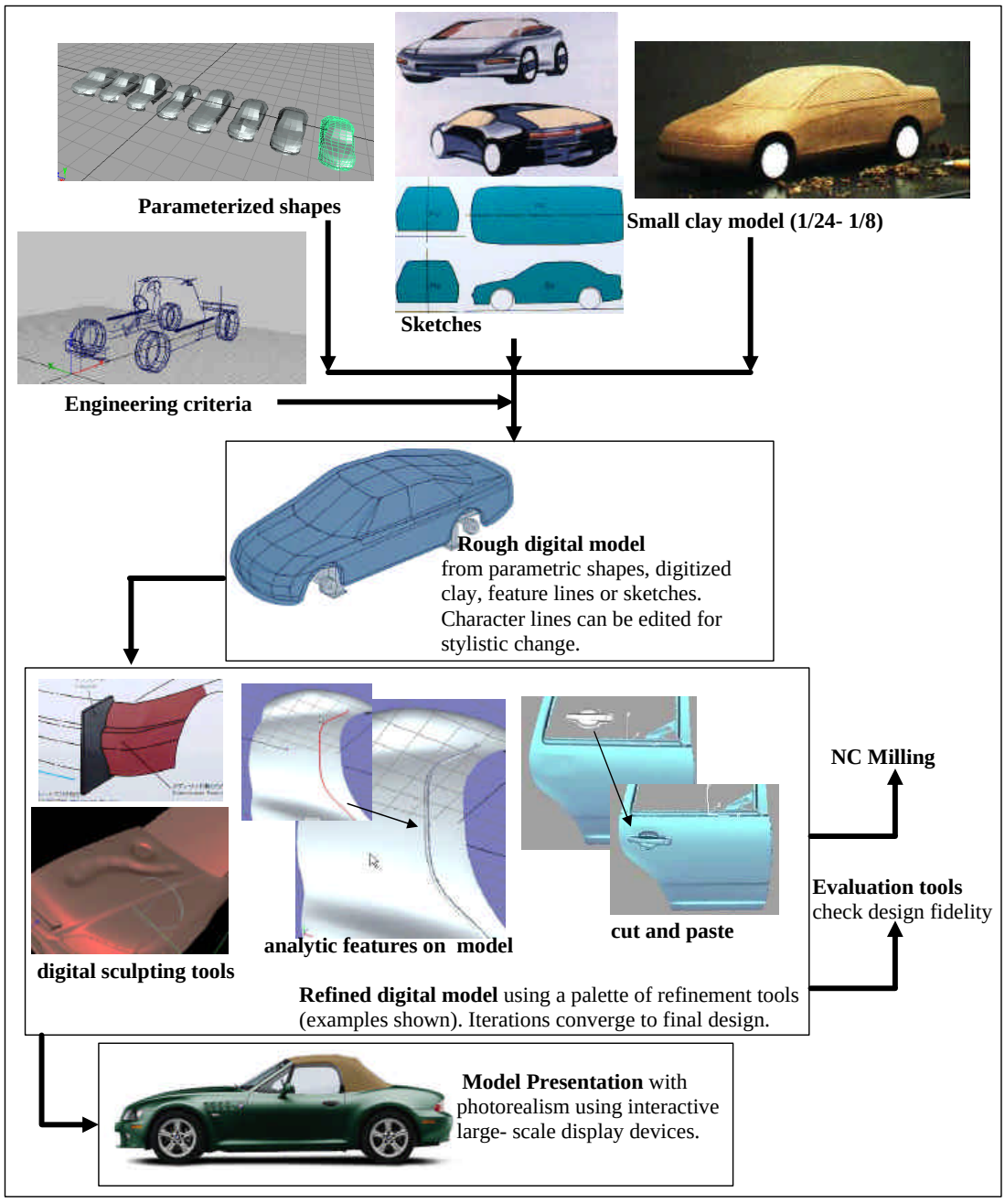


Figure 7: Proposed Automotive Design Workflow

judicious use of user input to help resolve ambiguities in the given input.

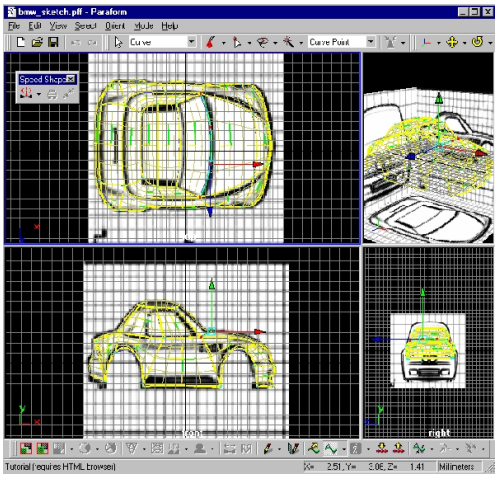


Figure 8: Aligning orthographic sketches into a common 3D space

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 [Llamas et al. 2003].
- Cut and paste [Biermann et al. 2002].
- Surface-Feature based editing [Sorkine et al. 2004].
- Local deformation and sculpting of object detail [Massie T. H. 1994].

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 [P. Poulin and Frasson 1998]. 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 [P. Poulin 1997] 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.

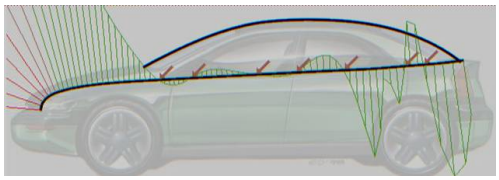


Figure 9: Curvature comb plot showing curvature discontinuities

5 Current modeling trends

It is clear that conceptual design in the future will require the coexistence of both physical and digital representations of objects. Physical models are converted to digital models using scanning devices [Curless] 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).

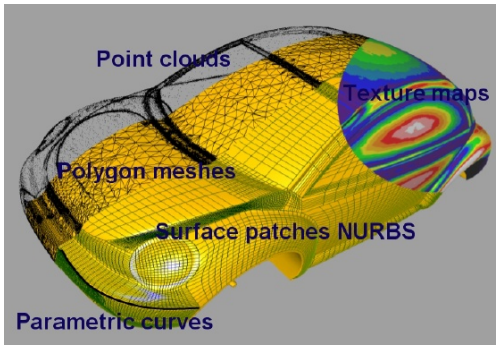


Figure 10: Various geometric representations used in automotive design

Historically, continuous parametric curve and surface patches constructed from piecewise polynomial splines, have been used to represent industrial design objects [Farin 2001]. 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 [S. Rusinkiewicz 2000], 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 pa-

rameterization 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 [Chaikin] 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 [Bloomenthal 1997]. 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 [J.C. Carr 2001]. 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 [S. Rusinkiewicz 2000]. 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 [Curless]. 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 [Massie T. H. 1994]. 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 [Grossman et al. 2003] and a motion capture system [Sheng 2004] 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 [Liu 2003].

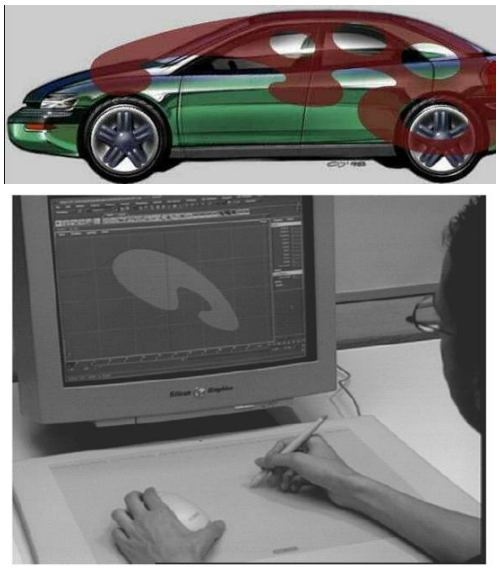


Figure 11: Curve modeling with sweeps [Singh 1999]

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.

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

BALAKRISHNAN, R., FITZMAURICE, G., KURTENBACH, G., AND BUXTON, W. 1999. Digital tape drawing. In *Proceedings of the 12th annual ACM symposium on User interface software and technology*, ACM Press, 161–169.

BIERMANN, H., MARTIN, I., BERNARDINI, F., AND ZORIN, D. 2002. Cut-and-paste editing of multiresolution surfaces. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 312–321.

BLOOMENTHAL, J. 1997. *Introduction to Implicit Surfaces*. Morgan Kauffmann.

CHAIKIN, G. An algorithm for high speed curve generation.

CURLESS, B. From range scans to 3d models.

EGGLI, L., HSU, C., BRUDERLIN, B., AND ELBER, G. 1997. Inferring 3d models from freehand sketches and constraints. *Computer-Aided Design 29*, 2, 101–112.

FARIN, G. 2001. *Curves and srufacs for CAGD, A practical guide 5th edition*. Morgan Kauffmann.

GROSSMAN, T., BALAKRISHNAN, R., AND SINGH, K. 2003. 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*, ACM Press, 185–192.

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

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

LIU, X. 2003. Plasticine surgery: editing digital models using physical materials. In *Master Thesis, Computer Science, University of Toronto*.

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

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

M. PAULY, M. G. 2001. Spectral processing of point-sampled geometry. In *SIGGRAPH '01*.

MASSIE T. H., S. J. K. 1994. The phantom haptic interface : A device for probing virtual objects. In *Proceedings of ASME'94*.

P. POULIN, M. O., AND FRASSON, M.-C. 1998. Interactively modeling with photogrammetry. In *Eurographics Workshop on Rendering '98*.

P. POULIN, K. RATIB, M. J. 1997. Sketching shadows and highlights to position lights. In *Computer Graphics International*, 56–63.

S. RUSINKIEWICZ, M. L. 2000. Qsplat: A multiresolution point rendering system for large meshes. In *SIGGRAPH '00*.

SHENG, J. 2004. An interface for virtual 3d sculpting via physical proxy. In *Master Thesis, Computer Science, University of Toronto*.

SINGH, K. 1999. Interactive curve design using digital french curves. In *ACM Symposium on Interactive 3D Graphics*, 23–30.

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

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

TSANG, S., BALAKRISHNAN, R., SINGH, K., AND RANJAN, A. 2004. A suggestive interface for image guided 3d sketching. In *Proceedings of CHI 2004*, 591–598.

V. KRISHNAMURTHY, M. L. 1996. Fitting smooth surfaces to dense polygon meshes. In *SIGGRAPH '96*, ACM Press/Addison-Wesley Publishing Co., 313–324.

W. BUXTON, G. FITZMAURICE, R. B. G. K. 2000. Large displays in automotive design. *IEEE Computer Graphics and Applications*, 68–75.

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

Space Deformations and their Application To Shape Modeling

Alexis Angelidis
Dynamic Graphics Project, University of Toronto*

Karan Singh
Dynamic Graphics Project, University of Toronto

Abstract

We present an overview of a set of techniques called space deformations, also known as free-form deformations, warps, skinning or deformers. This family of techniques has various applications in modeling, animation, rendering or simulation, and we focus especially on their application to modeling. Space deformation techniques are mappings of space onto another space, and can therefore be applied conveniently to any embedded geometry. This independence from the underlying geometric representation of deformed shape makes even the simplest and earliest deformation techniques still applicable and popular in current industrial practice.

1 Introduction

Computer Graphics representations of shape are typically defined using a discrete set of parameters that have a visual manifestation, such as the vertices of a mesh, control points of parameteric curves and surfaces or skeletal shapes of implicit surfaces. These visual parameters, traditionally, also serve as handles for the interactive manipulation of the underlying shape. Unfortunately, simply using underlying mathematical handles as a manipulation interface has two major disadvantages. First, there is no connection between the resolution and visual layout of the shape handles and the user desired manipulation. Creating a diagonal surface ridge, for example, by moving control vertices of a rectangular patch is a extremely difficult. Second,deformations defined using the handles of a specific representation cannot be trivially be applied to other shape representations or even different instances of the same shape representation. Space deformations are a family of techniques that address these defficiencies by defining manipulations of space that are directly applicable to any embedded shape representation. We present an overwiew of space deformation in Section 2, followed by a more detailed presentation of various space deformation techniques that are particularly applicable to shape modeling in Section 3. In Section 4, we summarize the techniques presented with a taxonomy of space deformation.

1.1 Principle of Modeling by Space Deformation

With space deformations, a deformed shape is obtained by 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 final shape $S(t_n)$ is defined by composition of functions applied to the initial shape $S(t_0)$:

$$S(t_n) = \left\{ \bigop_{i=0}^{n-1} f_{t_i \mapsto t_{i+1}}(\mathbf{p}) \mid \mathbf{p} \in S(t_0) \right\} \quad (1)$$

$$\text{where } \bigop_{i=0}^{n-1} f_{t_i \mapsto t_{i+1}}(\mathbf{p}) = f_{t_{n-1} \mapsto t_n} \circ \cdots \circ f_{0 \mapsto 1}(\mathbf{p})$$

The operator Ω 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 $\mathbf{p}$ of space at time t_i into a point of space at time t_{i+1}. Sections 2 will focus on defining functions $f_{t_i \mapsto t_{i+1}}$. Section 3 will address issues related to representing $S(t_i)$.

Normal Deformation: Computing accurate normals to the surface is very important, since normals are used for shading and their level of quality will dramatically affect the visual quality of the shape. Let us recall that in order to compute the new normal after deformation, the previous normal needs to be multiplied by the co-matrix[1] of the Jacobian of the deformation [Barr 1984]. The Jacobian of f at $\mathbf{p}$ is the matrix $J(f, \mathbf{p}) = (\frac{\partial f}{\partial x}(\mathbf{p}), \frac{\partial f}{\partial y}(\mathbf{p}), \frac{\partial f}{\partial z}(\mathbf{p}))$, and the following expression is a convenient way to compute the co-matrix of $J = (\mathbf{j}_x, \mathbf{j}_y, \mathbf{j}_z)$, where the vectors $\mathbf{j}_x$, $\mathbf{j}_y$ and $\mathbf{j}_z$ are column vectors and $\times$ denotes the cross product:

$$J^{\mathrm{C}} = (\mathbf{j}_y \times \mathbf{j}_z, \mathbf{j}_z \times \mathbf{j}_x, \mathbf{j}_x \times \mathbf{j}_y) \quad (2)$$

Note that the multiplication of a vector by J^{C} does not preserve its length. It is therefore necessary to divide a deformed normal by its magnitude.

Generic blending: An practical advantage of space deformation techniques is that they may be treated as blackboxes and blended in a generic manner. Let us consider n deformations f_i and define a partition of unity w_i, possibly scalar fields. The deformations can be applied simultaneously to a point:

$$\mathbf{p} + \sum_{i=1}^{n} w_i (f_i(\mathbf{p}) - \mathbf{p}) \quad (3)$$

The space deformation family of techniques can therefore be seen as a toolbox in which the tools can be combined by handcrafting the weights w_i. In the following, we will however present them independently from each other to underline their strength and weaknesses.

2 Space Deformations Techniques

This section reviews several space deformation techniques, organized in four groups based on the *geometric connectivity* between the control handles: point/parameter controls,

*On leave from the Graphics & Vision Research Lab., U. of Otago, New Zealand.

[1] Matrix of the co-factors.

curve controls, surface controls, lattice-based controls and blendable controls. Although all space deformation may be blended using the above generic method, some of the techniques include convenient blending techniques in their formalism, and the interacting handles are not restricted by any connectivity.

For the sake of clarity, we present existing space deformations aligned with the orthonormal axes $\mathbf{e}_x$, $\mathbf{e}_y$ and $\mathbf{e}_z$ and within the unit cube $[0,1]^3$ whenever possible, because a mere change of coordinates allows the artist to place the deformation anywhere in space. To compare existing deformation techniques from the same point of view, we also use $\mathbf{e}_z$ as the common axis of deformation when applicable, thus we reformulate some of the original formulas. To begin with, note that affine transformations (translation, rotations, and scale) are the simplest examples of space deformations.

2.1 Point/parameters Control

This section contains the subset of space deformations whose control parameters are disconnected elements often without any explicit visual handle.

2.1.1 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 [Barr 1984]. We denote $(x,y,z)^{\mathrm{T}}$ 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 4×4 homogeneous matrices to be applied to the coordinates of every point to be deformed.

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.

Twisting operation: The function θ 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.

Bending operation: This operation bends space along the axis y, in the $0 < z$ half-space. The desired radius of curvature is specified with ρ. The angle corresponding to ρ is $\theta = \hat{z}/\rho$. The value of $\hat{z}$ is the value of z, clamped in the interval $[0, z_{\max}]$.
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, assuming the deformation is reversible.

$$\begin{pmatrix} \cos\theta & 0 & \sin\theta & \rho - \rho\cos\theta - \hat{z}\sin\theta \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & \rho\sin\theta - \hat{z}\cos\theta \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

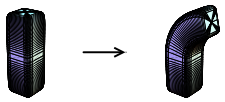


Figure 3: Bend deformation of a super-ellipsoid.

2.1.2 A Generic Implementation of Axial Procedural Deformation Techniques,

C. Blanc extends A. Barr's work to mold, shear and pinch deformations [Blanc 1994]. Her transformations use a function of one or two components. She names this function the *shape function*. Examples and formulas are shown in Figures 4, 5, and 6.

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

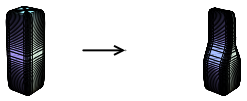


Figure 4: Pinch deformation of a super-ellipsoid.

$$\begin{pmatrix} r(\tan^{-1}(x,y)) & 0 & 0 & 0 \\ 0 & r(\tan^{-1}(x,y)) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

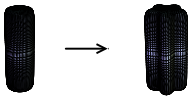


Figure 5: Mold deformation of a super-ellipsoid.

2.1.3 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 [Decaudin 1996]. 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 [Hirota et al. 1999].

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 9. A restriction on the tool is to be star-convex with respect to its center $\mathbf{c}$. The deformation function is[2] (see Figure 8):

$$f_{3D}(\mathbf{p}) = \mathbf{c} + \sqrt[3]{\rho(\mathbf{p})^3 + r(\mathbf{p})^3}\ \mathbf{n} \qquad (4)$$

- $\rho(\mathbf{p})$ is the magnitude of the vector $\mathbf{u} = \mathbf{p} - \mathbf{c}$.

[2]The 2D case is obtained by replacing 3 with 2.

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

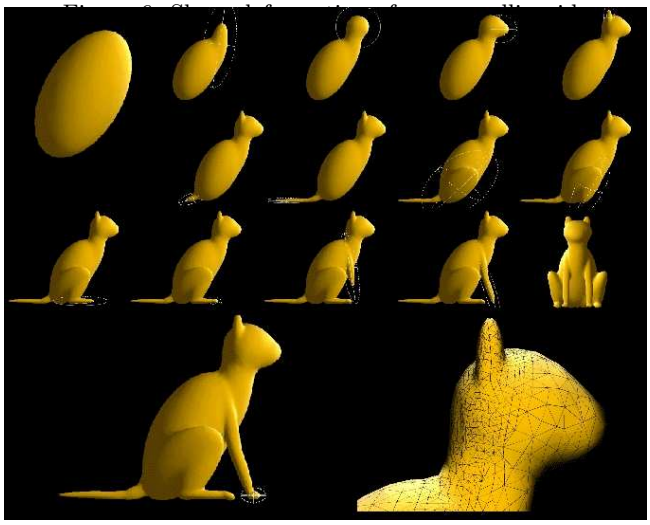


Figure 7: Steps of the modeling of a cat, image by P. Decaudin.

- $r(\mathbf{p})$ is the distance between $\mathbf{c}$ and the intersection of the tool with the half-line $(\mathbf{c}, \mathbf{u})$.
- $\mathbf{n} = \mathbf{u}/\|\mathbf{u}\|$ is the unit vector pointing from $\mathbf{c}$ to $\mathbf{p}$.

If the tool was not a star-convex in $\mathbf{c}$, then $r(\mathbf{p})$ would be ambiguous. The deformation is foldover-free. It is continuous everywhere except at the center $\mathbf{c}$. The effect of the deformation converges quickly to the identity with the increasing distance from $\mathbf{c}$. The deformation can be considered local, and is smooth everywhere except at $\mathbf{c}$. An example in $3D$ is shown in Figure 7. A feature of this space deformation which is rare, is that it has an exact yet simple inverse in the space outside the tool:

$$f_{3D}^{-1}(\mathbf{p}) = \mathbf{c} + \sqrt[3]{\rho(\mathbf{p})^3 - r(\mathbf{p})^3}\ \mathbf{n} \tag{5}$$

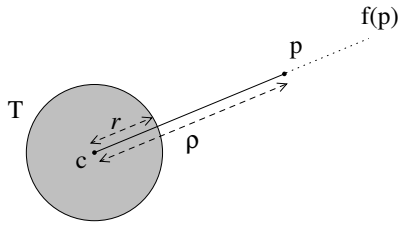


Figure 8: The insertion of a tool at center $\mathbf{c}$ affects the position of point $\mathbf{p}$. See the deformation in Equations (4).

2.1.4 Interactive Space Deformation with Hardware Assisted Rendering

Y. Kurzion and R. Yagel present *ray deflectors* [Kurzion and Yagel 1997]. 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 $\mathbf{c}$. Let ρ be the distance from the center of the deflector $\mathbf{c}$ and a point $\mathbf{p}$.

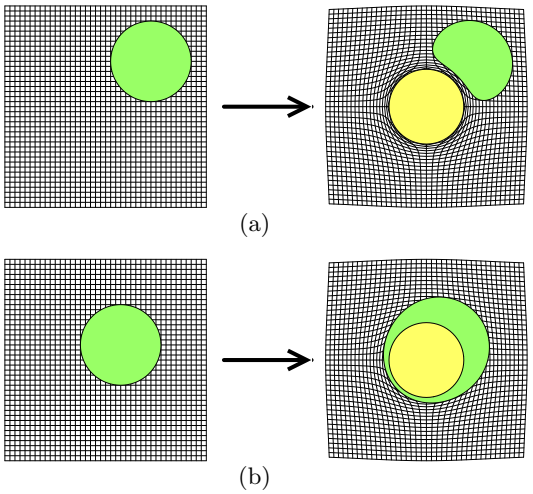


Figure 9: (a) Deformation of a shape (green) by blowing up a tool (yellow) outside the shape. The shape's area is preserved. (b) Deformation of a shape by blowing up a tool inside the shape. The shape's area is increased by that of the tool.

$$\rho = \|\mathbf{p} - \mathbf{c}\| \tag{6}$$

Translate deflector: To define a translate deflector, the artist has to provide a translation vector, $\mathbf{t}$. The effect of the translate deflector will be to transform the center point, $\mathbf{c}$, into $\mathbf{c} + \mathbf{t}$.

$$f_{\mathrm{T}}(\mathbf{p}) = \begin{cases} \mathbf{p} - \mathbf{t}(1 - \frac{\rho^2}{r^2})^2 & \text{if } \rho < r \\ \mathbf{p} & \text{otherwise} \end{cases} \quad \text{where } \theta \in \mathbb{R} \tag{7}$$

Rotate deflector: To define a rotate deflector, the artist has to provide an angle of rotation, θ, and a vector, $\mathbf{n}$, about which the rotation will be done. The reader can find the expression of a rotation matrix, $R_{\theta',\mathbf{n},\mathbf{c}}$, in Appendix **??**. Let us call θ' an angle of rotation that varies in space:

$$\theta' = -\theta(1 - \frac{\rho^2}{r^2})^4$$

$$f_{\mathrm{R}}(\mathbf{p}) = \begin{cases} R_{\theta',\mathbf{n},\mathbf{c}} \cdot \mathbf{p} & \text{if } \rho < r \\ \mathbf{p} & \text{otherwise} \end{cases} \quad \text{where } \|\mathbf{t}\| \in [0, \frac{3\sqrt{3}r}{8}] \tag{8}$$

Scale deflector: To define a scale deflector, the artist has to provide a scale factor s. The scale deflector acts like a

magnifying glass.

$$f_S(\mathbf{p}) = \begin{cases} \mathbf{p} - (\mathbf{p}-\mathbf{c})(1-\frac{\rho^2}{r^2})^4 s & \text{if } \rho < r \\ \mathbf{p} & \text{otherwise} \end{cases} \quad (9)$$
$$\text{where } s \in [-1,1]$$

Discontinuous deflector: To define a discontinuous deflector, the artist has to provide a translation vector, $\mathbf{t}$. The deflector is split into two halves, on each side of a plane going through $\mathbf{c}$ and perpendicular to $\mathbf{t}$. In the half pointed at by $\mathbf{t}$, the discontinuous deflector will transform $\mathbf{c}$, into $\mathbf{c}+\mathbf{t}$, while in the other half, the discontinuous deflector will transform $\mathbf{c}$, into $\mathbf{c}-\mathbf{t}$. The effect will be to cut space. The deformation applied to the rays is:

$$f_D(\mathbf{p}) = \begin{cases} \mathbf{p} - \mathbf{t}(1-\frac{\rho^2}{r^2})^2 & \text{if } \rho<r \text{ and } 0<(\mathbf{p}-\mathbf{c})\cdot\mathbf{t} \\ \mathbf{p} + \mathbf{t}(1-\frac{\rho^2}{r^2})^2 & \text{if } \rho<r \text{ and } (\mathbf{p}-\mathbf{c})\cdot\mathbf{t}<0 \\ \mathbf{p} & \text{otherwise} \end{cases} \quad (10)$$
$$\text{where } \theta \in \mathbb{R}$$

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 10(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 [Schein and Elber 2004].

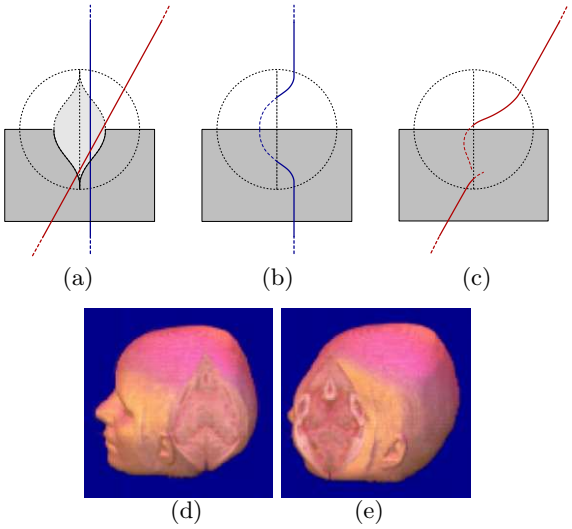


Figure 10: (a) Discontinuous deflector as observed by the artist. Two arbitrary rays are shown. (b) Simple case, where the ray of light crosses only one hemisphere. (c) When the ray of light changes hemisphere, the curve of light is subject to a discontinuity. (d, e) Images by Y. Kurzion and R. Yagel.

2.1.5 Twister

I. Llamas et al propose a method called twister in which a twist transformation of points is weighted with a scalar function [Llamas et al. 2003], 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 $(\mathbf{o},\mathbf{u},\mathbf{v},\mathbf{w})$ into $(\mathbf{o}',\mathbf{u}',\mathbf{v}',\mathbf{w}')$. The axis of the twist is defined by a direction $\mathbf{d}$ and point $\mathbf{a}$ on the axis, while the angle of rotation around the axis is α and the translation factor along the axis is d:

$$\begin{aligned} \mathbf{d} &= \tfrac{\mathbf{g}}{\|\mathbf{g}\|} \\ &\text{where } \mathbf{g} = (\mathbf{u}'-\mathbf{u})\times(\mathbf{v}'-\mathbf{v}) + (\mathbf{v}'-\mathbf{v})\times(\mathbf{w}'-\mathbf{w}) + (\mathbf{w}'-\mathbf{w})\times(\mathbf{u}'-\mathbf{u}) \\ \alpha &= 2\arcsin(\tfrac{\|\mathbf{u}'-\mathbf{u}\|}{2\|\mathbf{d}\times\mathbf{u}\|}) \\ d &= \mathbf{d}\cdot(\mathbf{o}'-\mathbf{o}) \\ \mathbf{a} &= \tfrac{\mathbf{o}+\mathbf{o}'-d\mathbf{d}}{2} + \tfrac{\mathbf{d}\times(\mathbf{o}-\mathbf{o}')}{2\tan(\alpha/2)} \end{aligned} \quad (11)$$

Their procedure for deforming a point $\mathbf{p}$ with a twist parameterized in t is:

1. Bring $\mathbf{p}$ into local coordinates: translate by $-\mathbf{a}$ and then rotate by a rotation that maps $\mathbf{d}$ onto $\mathbf{z}$.
2. Apply the twist in local coordinates: translate by $t\,d$ along $\mathbf{z}$ and rotate by $t\,\alpha$ around $\mathbf{z}$
3. Finally bring $\mathbf{p}$ back into world coordinates: rotate by a rotation that maps $\mathbf{z}$ onto $\mathbf{d}$ and translate by $\mathbf{a}$

To weight the twist, they propose to use a piecewise scalar function:

$$t(\mathbf{p}) = \cos^2(\pi\|\mathbf{p}-\mathbf{o}\|/2r) \quad (12)$$

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.

2.2 Curve Control

Curve Control 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 $\mathbf{e}_z$ as the common axis of deformation, thus we reformulate some of the original formulas.

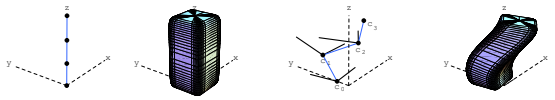
2.2.1 A Generalized de Casteljau Approach to 3d Free-Form Deformation

Y.K. Chang and A.P. Rockwood propose a polynomial deformation that efficiently achieves "Barr"-like deformations and more [Chang and Rockwood 1994], using a Bézier curve with coordinate systems defined along $\mathbf{e}_z$ at the curve's control knots $(z_0, z_1 \ldots, z_n) \in [0,1]^{n+1}$. The initially straight segment $z \in [0,1]$ is deformed by defining coordinate systems $(\mathbf{c}_i, \mathbf{u}_i, \mathbf{v}_i, \mathbf{w}_i)$ along that segment. The shape follows the deformation of the segment, as shown in Figure 11.
To compute the image $\mathbf{q}$ of a point $\mathbf{p}$ of the original shape, the matrix transforming a point to a local coordinate systems is needed:

$$M_i = \begin{pmatrix} u_{i,x} & v_{i,x} & w_{i,x} & c_{i,x} \\ u_{i,y} & v_{i,y} & w_{i,y} & c_{i,y} \\ u_{i,z} & v_{i,z} & w_{i,z} & c_{i,z} \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (13)$$

where $\mathbf{w}_i = \mathbf{c}_{i+1} - \mathbf{c}_i$, and $\mathbf{u}_i$, $\mathbf{v}_i$ are the handles.

Using this matrix, the deformation of a point is obtained recursively with the de Casteljau algorithm for evaluating a



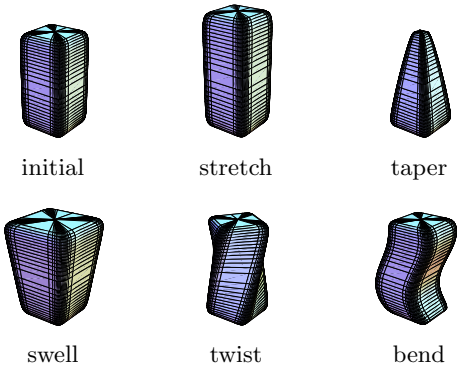
straight axis | initial shape | control points and handles | deformed shape

Figure 11: 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.

Bézier curve:

$$f_i^j(\mathbf{p}) = (1-\mathbf{p}_z) f_i^{j-1}(\mathbf{p}) + \mathbf{p}_z f_{i+1}^{j-1}(\mathbf{p}) \quad (14)$$
$$\text{where } f_i^0(\mathbf{p}) = M_i \cdot \mathbf{p}$$

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 12, this method is capable of performing "Barr"-like deformations and more.



initial | stretch | taper

swell | twist | bend

Figure 12: Deformation of a super-ellipsoid.

2.2.2 Axial Deformation

A limitation of the above method is the initial rectilinear axis. If the shape is initially 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 [Lazarus et al. 1994]. Let us define a parametric curve $\mathbf{c}(u)$. A point $\mathbf{p}$ in space is attached to local coordinates along the curve. The origin of this local coordinate system is $\mathbf{c}(u_\mathbf{p})$, the closest point to $\mathbf{p}$ on the curve, and the axes are those of an extended Frenet frame that discards vanishing points [Bloomenthal 1990]. To find the closest point to $\mathbf{p}$ on curves, B. Crespin proposes an efficient algorithm based on subdivision [Crespin 1999]. 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 $\mathbf{t}(u)$, a normal $\mathbf{n}(u)$ and a binormal $\mathbf{b}(u)$. The propagated frame is computed as follows:

- the unit tangent at the origin is given by the equation of the curve: $\mathbf{t}(0) = \frac{\mathbf{dc}(0)}{\mathrm{d}u} / \| \frac{\mathbf{dc}(0)}{\mathrm{d}u} \|$.
- 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 fast formulation:

$$R = \begin{pmatrix} a_{xx}+\theta & a_{xy}+b_z & a_{zx}-b_y \\ a_{xy}-b_z & a_{yy}+\theta & a_{yz}+b_x \\ a_{zx}+b_y & a_{yz}-b_x & a_{zz}+\theta \end{pmatrix} \quad (15)$$

$$\text{where } \begin{array}{ll} (a_x,a_y,a_z)^\top = \frac{\mathbf{t}(u_i)\times\mathbf{t}(u_{i+1})}{\|\mathbf{t}(u_i)\times\mathbf{t}(u_{i+1})\|} & \alpha = 1-\theta \\ \theta = \mathbf{t}(u_i)\cdot\mathbf{t}(u_{i+1}) & \beta = \sqrt{1-\theta^2} \end{array} \quad (16)$$

$$\begin{array}{lll} a_{xx} = \alpha a_x^2 & a_{xy} = \alpha a_x a_y & b_x = \beta a_x \\ a_{yy} = \alpha a_y^2 & a_{yz} = \alpha a_y a_z & b_y = \beta a_y \\ a_{zz} = \alpha a_z^2 & a_{zx} = \alpha a_z a_x & b_z = \beta a_z \end{array} \quad (17)$$

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: $\mathbf{t}(u_{i+1}) = \frac{\mathbf{dc}(u_{i+1})}{\mathrm{d}u} / \| \frac{\mathbf{dc}(u_{i+1})}{\mathrm{d}u} \|$.
- the normal is given by the rotation of the previous normal: $\mathbf{n}(u_{i+1}) = R \cdot \mathbf{t}(u_i)$.
- the binormal is given by a cross product: $\mathbf{b}(u_{i+1}) = \mathbf{t}(u_i) \times \mathbf{n}(u_i)$.

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 [Crespin 1999].

2.2.3 Blendeforming: Ray Traceable Localized Foldover-Free Space Deformation

As explained in the introduction, the motivation for which a space deformation should be foldover-free is its reversibilty, with applications such as undoing operations or raytracing. D. Mason and G. Wyvill introduce *blendeforming* [Mason and Wyvill 2001]. 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: $b_{\mathbf{x},\mathbf{y}} : \mathbb{R}^2 \mapsto [-d_{\max}, d_{\max}]$ and $b_\mathbf{z} : [0,1] \mapsto [0,1]$. Function $b_{\mathbf{x},\mathbf{y}}$ controls the amount of deformation for each individual z-streamlines, and the choice of function $b_\mathbf{z}$ affects the maximum compression of space along the streamlines. The deformation of point $\mathbf{p} = (x,y,z)^\top$ is

$$\mathbf{p}_{def} = (x, y, z_{def})^\top \quad (18)$$
$$\text{where } z_{def} = z + b_{\mathbf{x},\mathbf{y}}(x,y)\ b_\mathbf{z}(z)$$

It is the definition of $b_\mathbf{z}$ together with a corresponding threshold $d_{\max}$ that prevents foldovers, as shown in Figure 13. The following function is a possible choice for $b_\mathbf{z}(z)$, used in the example:

$$b_\mathbf{z}(z) = \begin{cases} 16z^2(1-z)^2 & \text{if } z \in [0,1] \\ 0 & \text{otherwise} \end{cases} \quad (19)$$
$$\text{with } \ d_{\max} = \frac{3\sqrt{3}}{16} \simeq 0.324$$

Functions permitting larger values for $d_{\max}$ can be found in the original paper. Since $b_{\mathbf{x},\mathbf{y}}$ 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 blendeforming bending functions defined in cylindrical coordinates, or using control curves

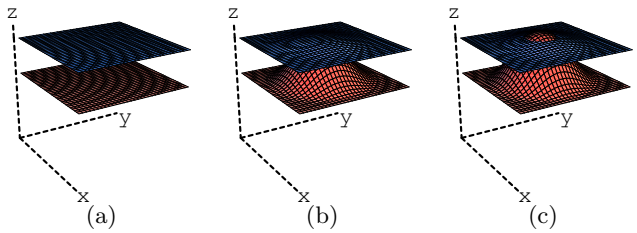


Figure 13: (a) Initial scene: two parallel planes. (b) Blendeforming, with $b_{\mathbf{x},\mathbf{y}}(x, y) = (x^2 - x + y^2 - y - 1/2)^2$. The value of $d_{\max}$ guarantees that the two planes will never intersect. (c) With $d_{\max} < d$, foldover occurs: the lower plane intersects the higher plane.

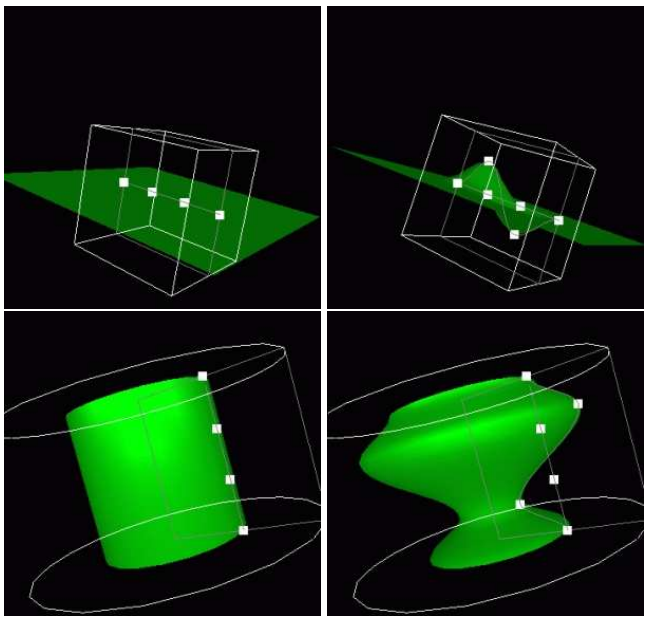


Figure 14: *Top*: a control curve with height control points. *Bottom*: a control curve with radial control points. Image by D. Mason and G. Wyvill.

2.2.4 Interactive Skeleton-Driven Dynamic Deformations

S. Capell et al. propose a framework for skeleton-driven animation of elastically deformable characters [Capell et al. 2002]. This techique defines over a character's skeleton-structure a layer of FFD lattice [Sederberg and Parry 1986] with the control points driven by the dynamic equation of elasticity. Simulation is beyond the scope of this overview, thus we refer to the original paper for detail.

2.3 Surface Control

2.3.1 Surface-Oriented Free-Form Deformation (SOFFD)

K. Singh and E. Kokkevis introduce *Surface-Oriented Free-Form Deformation* (SOFFD) in the context of character animation [Singh and Kokkevis 2000].

To deform a shape S, a SOFFD is defined as a triple $(\mathrm{D}, \mathrm{R}, l)$: the reference surface R, the driver surface D and a scalar value l that controls the influence. The SOFFD process is made of three phases: bind, registration and deformation.

In the binding phase, the surfaces R and D are constructed as low resolution representation of S. The surfaces R and D are initially identical, and their patches define local coordinate systems $M_{\mathrm{R},i}$ $M_{\mathrm{D},i}$ that correspond to each other.

In the registration phase, the local position of $\mathbf{p}_{\mathrm{S}}$ in each patch of R is computed using $\mathbf{q}_{\mathrm{S},i} = M_{\mathrm{R},i}^{-1} \cdot \mathbf{p}_{\mathrm{S},i}$, and the influence u_i of each patch of R at a point $\mathbf{p}_{\mathrm{S}}$ of S are computed using the distance d_i to the i^{th} patch of R :

$$\begin{aligned} u_i &= \frac{w_i}{\sum_j w_j} \\ \text{where } w_i &= \frac{1}{1 + d_i^l} \end{aligned} \tag{20}$$

In the deformation phase, the weighted effect are added to compute the final deformation $\mathbf{p}'_{\mathrm{S}}$ a point $\mathbf{p}_{\mathrm{S}}$ of S:

$$\mathbf{p}'_{\mathrm{S}} = \sum u_i \; M_{\mathrm{D},i} \cdot \mathbf{q}_{\mathrm{S}} \tag{21}$$

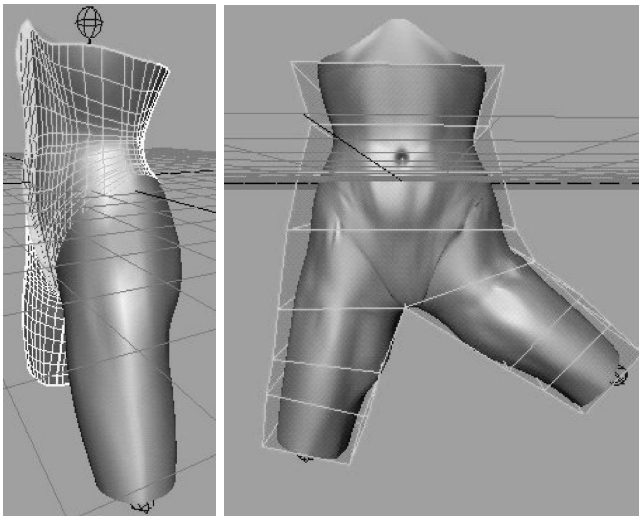


Figure 15: Deformation with SOFFD. Image by K. Singh and E. Kokkevis

2.4 Lattice Control

The limitation of curve or surface controlled 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 [Singh and Fiume 1998]. 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, $(\mathbf{p}_i, \mathbf{q}_i)$. The deformation f will interpolate or approximate the pairs in this way $f(\mathbf{p}_i) = f_{\mathbf{p}}(\mathbf{p}_i) \approx \mathbf{q}_i$. The function $f_{\mathbf{p}}$ 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, $(\mathbf{p}_i, \mathbf{v}_i)$. The deformation f will interpolate or approximate the pairs in this way $f(\mathbf{p}_i) = \mathbf{p}_i + f_{\mathbf{v}}(\mathbf{p}_i) \approx \mathbf{p}_i + \mathbf{v}_i$. The function $f_{\mathbf{v}}$ 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 [Rutherford 1990; Griffiths 1999]. 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.

2.4.1 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 [Sederberg and Parry 1986].

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 $\mathbf{v}_{ijk}$ the $(l+1)\times(m+1)\times(n+1)$ *control vectors* defined by the artist. The smoothed vector field is a mapping $\mathbf{p} \in [0,1]^3 \mapsto \mathbb{R}^3$.

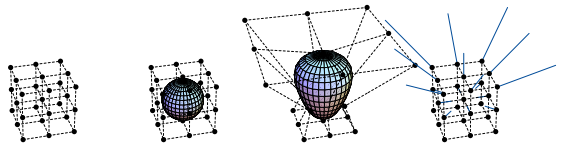
$$\mathbf{v}(\mathbf{p}) = \sum_{i=0}^{l} \binom{i}{l} (1-x)^{l-i} x^i \left(\sum_{j=0}^{m} \binom{j}{m} (1-y)^{m-j} y^j \left(\sum_{k=0}^{n} \binom{k}{n} (1-z)^{n-k} z^k \right) \right) \mathbf{v}_{ijk} \tag{22}$$

Then the deformation of a point is a translation of that point

$$\mathbf{p}_{def} = \mathbf{p} + \mathbf{v}(\mathbf{p}) \tag{23}$$

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 $\mathbf{v}_{ijk}$ 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 [Griessmair and Purgathofer 1989].

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 16(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 that the manipulation of of lattice of control points requires a strong sense of spatial perception from the artist. Clearly, practical FFD manipulation through control-points can only be done with reasonably small lattices.



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

Figure 16: FFD deformation. (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.

2.4.2 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) [Coquillart 1990].

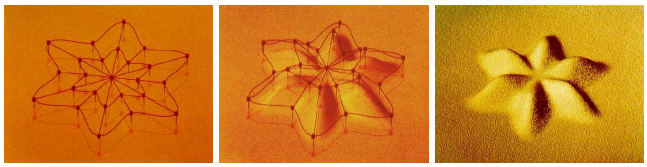
To model with EFFD, the artist first builds a lattice by placing the extended cells anywhere in space, and then manipulates the cells to deform the shape. An extended cell is a small FFD of size 4^4. The transformation from the cell's *local coordinates* $\mathbf{s} = (u, v, w)^\top$ to world coordinates is:

$$\mathbf{p}(\mathbf{s}) = \sum_{i=0}^{3} \binom{i}{3} (1-u)^{3-i} u^i \left(\sum_{j=0}^{3} \binom{j}{3} (1-v)^{3-j} v^j \left(\sum_{k=0}^{3} \binom{k}{3} (1-w)^{3-k} w^k \right) \right) \mathbf{p}_{ijk} \tag{24}$$

The eight corners $\mathbf{p}_{ijk\in\{0,3\}^3}$ 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 $\mathbf{s}$ in a cell is not straightforward. The local coordinates of a point $\mathbf{p}$ in a cell are found by solving Equation (24) in $\mathbf{s}$ using a numerical iteration. This can be unstable in some cases, although the authors report they did not encounter such cases in practice. Once $\mathbf{s}$ is found, the translation to apply to $\mathbf{p}$ is found by substituting in Equation (24) the control points $\mathbf{p}_{ijk}$ with the control vectors $\mathbf{v}_{ijk}$. 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. An example is shown in Figure 17.

2.4.3 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) (b) (b)

Figure 17: EFFD deformation, images by S. Coquillart. (a) Control lattice. (b) Deformed lattice. (c) Result: a sand-pie.

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 [Gain and Dodgson 2001]. 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 3×3 matrices are all positive, there is no foldover.

$$\phi_{ijk} = s \det\left(\mathrm{q}_{i\pm 1jk} - \mathrm{q}_{ijk} \ , \ \mathrm{q}_{ij\pm 1k} - \mathrm{q}_{ijk} \ , \ \mathrm{q}_{ijk\pm 1} - \mathrm{q}_{ijk}\right)$$

where the sign s is 1 if $(i\pm1, j\pm1, k\pm 1)$ 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 $(\mathbf{p}_i, \mathbf{v}_i)$, the pairs of points and vectors defining the DMFFD. If a foldover is detected, the DMFFD is recursively split into two parts: $(\mathbf{p}_i, \mathbf{v}_i/2)$ and $(\mathbf{p}_i + \mathbf{v}_i/2, \mathbf{v}_i/2)$. The procedure eventually converges, and the series of DMFFDs obtained are foldover-free and can be applied safely to the shape.

2.4.4 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 [MacCracken and Joy 1996]. 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 $\{L_1, L_2, \ldots L_l\}$. 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 [MacCracken and Joy 1996] 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.

2.4.5 Scalar-Field Guided Adaptive Shape Deformation and Animation (SFD)

J. Hua and H. Qin create a technique called SFD [Hua and Qin 2004]. 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 $\phi(t, \mathbf{p}(t))$, the scalar field which is animated in time, t. Since a moving point, $\mathbf{p}(t)$, is attached to a level-set of the scalar field, the value of ϕ at $\mathbf{p}$ is constant in time:

$$\frac{\mathrm{d}\phi}{\mathrm{d}t} = 0 \tag{25}$$

The square of Equation (25) gives a constraint:

$$(\frac{\mathrm{d}\phi}{\mathrm{d}t})^2 = 0 \tag{26}$$

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, $\|\mathbf{v}(t)\|$, 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, $\mathbf{v}$, and its neighbors' speed, $\mathbf{v}_k$:

$$(\nabla \cdot \mathbf{v})^2 \approx \frac{1}{k} \sum_k (\mathbf{v} - \mathbf{v}_k)^2 \tag{27}$$

Since this is a constrained optimization problem [Weisstein], there exists a Lagrange multiplier λ such that:

$$\frac{\mathrm{d}}{\mathrm{d}\mathbf{v}}(\frac{\mathrm{d}}{\mathrm{d}t}\phi(t, \mathbf{p}(t)))^2 + \lambda \frac{\mathrm{d}}{\mathrm{d}\mathbf{v}}(\nabla \cdot \mathbf{v})^2 = \vec{0} \tag{28}$$

According to the authors, λ 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 ϕ with the chain rule:

$$\frac{\mathrm{d}}{\mathrm{d}\mathbf{v}}\left((\nabla\phi \cdot \mathbf{v} + \frac{\partial\phi}{\partial t})^2 + \lambda(\nabla \cdot \mathbf{v})^2\right) = \vec{0} \tag{29}$$

Let us define $\hat{\mathbf{v}}$, the average of the velocity of all the adjacent neighbors connected with edges to point $\mathbf{p}$. If we substitute $(\nabla\cdot\mathbf{v})^2$ for its approximate given by Equation (27), and then apply the derivative with respect to $\mathbf{v}$, we obtain:

$$(\nabla\phi \cdot \mathbf{v} + \frac{\partial\phi}{\partial t})\nabla\phi + \lambda(\mathbf{v} - \hat{\mathbf{v}}) = \vec{0} \tag{30}$$

By solving the system of Equation (30), the updated position is:

$$\mathbf{v} = \hat{\mathbf{v}} - \frac{\hat{\mathbf{v}} \cdot \nabla\phi + \frac{\partial\phi}{\partial t}}{\lambda + (\nabla\phi)^2}\nabla\phi \tag{31}$$

The algorithm deforms a set of vertices in n substeps. If n is set to one, the deformation takes one step:

for i = 1 to n **do**
  for all $\mathbf{p}_k$ in the list of vertices to update **do**
    Update the scalar field $\phi(t + \Delta t, \mathbf{p}_k)$.
    Deduce $\frac{\partial \phi}{\partial t} = \frac{\phi(t+\Delta t, \mathbf{p}_k) - \phi(t, \mathbf{p}_k)}{\Delta t}$
    Calculate $\nabla\phi$, possibly with finite differences.
    Compute $\hat{\mathbf{v}}$ according to neighbors' velocities.
    Deduce $\mathbf{v}$ according to Equation (31).
    Update vertex positions with $\mathbf{p}_k(t + \Delta t) = \mathbf{p}_k(t) + \mathbf{v}\frac{\Delta t}{n}$
    Improve surface representation using a mesh refinement and simplification strategy.
    if $\phi(t + \Delta t, \mathbf{p}_k(t + \Delta t)) \approx \phi(t, \mathbf{p}_k(t))$ **then**
      remove $\mathbf{p}_k$ 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:

$$\mathbf{v} = -\frac{\frac{\partial \phi}{\partial t}}{\lambda + (\nabla\phi)^2}\nabla\phi \tag{32}$$

(a) (b) (c)

Figure 18: SFD applied to a digitized model of a dinosaur, images by J. Hua and H. Qin.

Note that this technique requires an explicit surface in order to compute the divergence of the speed. 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, necessary for shape modeling.

2.5 Blendable

All deformations can be combined together by combining the deformations with a partition of unity defined in space. Some deformation technique however include geometric blending more strongly in their formalism, and define blending methods that provides a variety of user control and total freedom in placing the control handles.

2.5.1 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) [Borrel and Bechmann 1991; Hsu et al. 1992]. The artist specifies translations $\mathbf{v}_i$ at points $\mathbf{p}_i$ in the form $(\mathbf{p}_i, \mathbf{v}_i)$. The DMFFD algorithm finds control vectors that satisfy, if possible, the artist's desire. Let us define a single input vector $\mathbf{v}$ at point $\mathbf{p}$. The FFD Equation (22) must satisfy

$$\mathbf{v} = \mathrm{B}(\mathbf{p})(\mathbf{v}_{ijk}) \tag{33}$$

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 $\mathbf{p}$. 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 $(\mathbf{v}_{ijk})$ is $3(l+1)(m+1)(n+1)$. When the artist specifies μ pairs $(\mathbf{p}_i, \mathbf{v}_i)$, the FFD Equation (22) must satisfy a larger set of equations:

$$\begin{pmatrix} \mathbf{v}_1 \\ \vdots \\ \mathbf{v}_\mu \end{pmatrix} = \mathbf{B} \cdot \begin{pmatrix} \mathbf{v}_{ijk} \\ \vdots \\ \mathbf{v}_{ijk} \end{pmatrix} \quad \text{where } \mathbf{B} = \begin{pmatrix} \mathrm{B}(\mathbf{p}_1) \\ \vdots \\ \mathrm{B}(\mathbf{p}_\mu) \end{pmatrix} \tag{34}$$

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 $\mathbf{v}_{ijk}$. The authors use the Moore-Penrose pseudo-inverse, $\mathbf{B}^+$. If the inverse of $\mathbf{B}^\top \cdot \mathbf{B}$ exists, then

$$\mathbf{B}^+ = (\mathbf{B}^\top \cdot \mathbf{B})^{-1} \cdot \mathbf{B}^\top \tag{35}$$

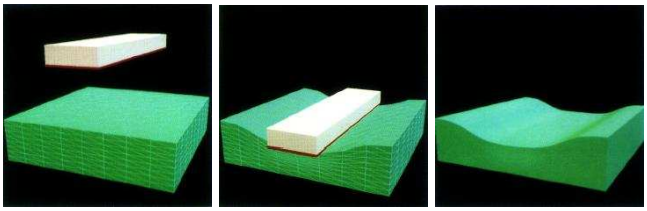
It is however preferable to compute the Moore-Penrose pseudo-inverse using singular value decomposition (SVD). The $\mu \times \nu$ matrix $\mathbf{B}$ can be written

$$\mathbf{B} = U \cdot D \cdot V^\top \tag{36}$$

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.

$$\mathbf{B}^+ = V \cdot D^{-1} \cdot U^\top \tag{37}$$

Here, the diagonal terms of D^{-1} are simply the inverse of the diagonal terms of D.



(a) (b) (b)

Figure 19: DMFFD deformation, images by W.M. Hsu et al. (a) Initial scene. (b) The deformation is created according to the displacement of several vertices of the green object. (c) Result. The authors do not describe how the vertices on the green object are selected.

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 [Borrel and Bechmann 1991]. 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:

$$\mathrm{B}_t(\mathbf{p}, t) = \begin{pmatrix} (t - t_0)(t - t_f) \\ (t - t_0)(t - t_f)t \\ (t - t_0)(t - t_f)t^2 \\ \vdots \end{pmatrix} \tag{38}$$

2.5.2 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) [Borrel and Rappoport 1994]. The artist defines constraint triplets $(\mathbf{p}_i, \mathbf{v}_i, r_i)$: a point, a vector that defines the desired image of the point, and a radius of influence. Let $\phi_i(\mathbf{p})$ denote the scalar function $\phi(\frac{\|\mathbf{p}-\mathbf{p}_i\|}{r_i})$ 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 $\mathbf{p}_i$. A naive vector field would be:

$$\mathbf{v}(\mathbf{p}) = \sum_{i=1}^{n} \mathbf{v}_i \phi_i(\mathbf{p}) \tag{39}$$

Unless the points $\mathbf{p}_i$ are far apart enough, Equation (39) will not necessarily satisfy the artist's input $\mathbf{v}(\mathbf{p}_i) = \mathbf{v}_i$ if the functions ϕ_i overlap. However, this can be made possible by substituting the vectors $\mathbf{v}_i$ with suitable vectors $\mathbf{w}_i$.

$$\mathbf{v}(\mathbf{p}) = \sum_{i=1}^{n} \mathbf{w}_i \phi_i(p) \tag{40}$$

These vectors $\mathbf{w}_i$ can be found by solving a set of $3n$ equations:

$$\mathbf{v}_i = (\mathbf{w}_1 \dots \mathbf{w}_n) \cdot \begin{pmatrix} \phi_1(\mathbf{p}_i) \\ \vdots \\ \phi_n(\mathbf{p}_i) \end{pmatrix} \text{ where } i \in [1, n] \tag{41}$$

Let us take the transpose, and arrange the n equations in rows. The following equation is the equivalent of Equation (34), but with radial basis functions:

$$\begin{pmatrix} \mathbf{v}_1^\top \\ \vdots \\ \mathbf{v}_n^\top \end{pmatrix} = \begin{pmatrix} \phi_1(\mathbf{p}_1) & \dots & \phi_n(\mathbf{p}_1) \\ \vdots & & \\ \phi_1(\mathbf{p}_n) & \dots & \phi_n(\mathbf{p}_n) \end{pmatrix} \cdot \begin{pmatrix} \mathbf{w}_1^\top \\ \vdots \\ \mathbf{w}_n^\top \end{pmatrix} \quad \text{where } i \in [1, n] \tag{42}$$

Let $\boldsymbol{\Phi}$ be the $n \times n$ square matrix of Equation (42). This matrix takes the role of $\mathbf{B}$ in Equation (34). Since $\boldsymbol{\Phi}$ can be singular, the authors also use its pseudo-inverse $\boldsymbol{\Phi}^+$ to find the vectors $\mathbf{w}_i$.

2.5.3 Dirichlet Free-Form Deformation (DFFD)

With DFFD, L. Moccozet and N. Magnenat-Thalmann propose a technique that builds the cells of a lattice automatically [Moccozet and Magnenat-Thalmann 1997], relieving the artist from a tedious task. The lattice cells are the cells of a Voronoï diagram of the control points, shown in Figure 20. The location of a point within a cell is neatly captured by the Sibson coordinates. The naive deformation of a point $\mathbf{p}$ is given by interpolating vectors defined at the control points with the Sibson coordinate.

$$\mathbf{p} \mathrel{+}= \sum_{i=1}^{n} \frac{a_i}{a} \mathbf{v}_i \tag{43}$$

Where a_i is the volume of cell i stolen by $\mathbf{p}$, and a is the volume of the cell of $\mathbf{p}$. This interpolation is only C^0. They use a method developed by G. Farin [Farin 1990] 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.

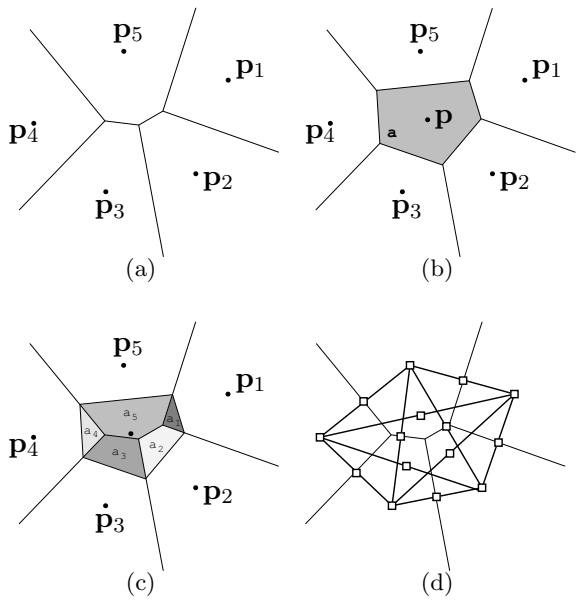


Figure 20: 2D illustration of the Sibson coordinates (a) Voronoï cells of the control points. (b) Voronoï diagram is updated after the insertion of point $\mathbf{p}$. (c) The areas stolen by the point $\mathbf{p}$ from its natural neighbors give the Sibson coordinates a_i/a. (d) Local control net, with *Bézier abscissa*.

2.5.4 Implicit Free-Form Deformations (IFFD)

B. Crespin introduces Implicit Free-Form Deformations (IFFD) [Crespin 1999]. 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 document [Bajaj et al. 1997]. The field $\phi \in [0, 1]$ generated by a skeleton modulates a transformation, M, of points. The deformation of point $\mathbf{p}$ with a single function is:

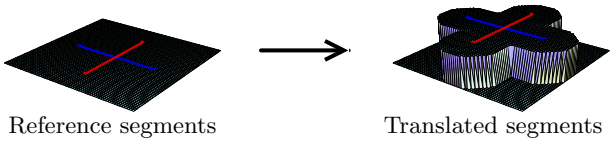
$$f(\mathbf{p}) = \mathbf{p} + \phi(\mathbf{p})(M \cdot \mathbf{p} - \mathbf{p}) \tag{44}$$

He proposes two ways to combine many deformations simultaneously. Let use denote $\mathbf{p}_i$ the transformation of $\mathbf{p}$ with deformation f_i. The first blending is shown in Figure 21. For M, we have used a translation matrix. The second blending attempts to solve the continuity issue, but requires the definition of supplementary profile functions, γ_i. The purpose of the index i is to assign individual profiles to skeletons.

In order to produce Figure 22, the following γ_i function was used:

$$\gamma_i(\mathbf{p}) = \begin{cases} 1-(1-\sigma^2)^2 & \text{if } \sigma \in [0,1], \text{ where } \sigma = \sum_{i=1}^{n} \phi_i(\mathbf{p}) \\ 1 & \text{otherwise} \end{cases} \tag{45}$$

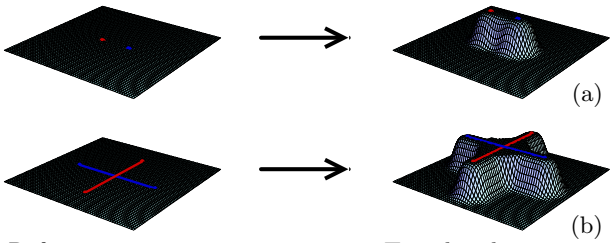
$$\mathbf{p}_{def} = \mathbf{p} + \frac{\sum_{i=1}^{n}(\mathbf{p}_i - \mathbf{p})\phi_i(\mathbf{p})}{\sum_{i=1}^{n}\phi_i(\mathbf{p})}$$



Reference segments Translated segments

Figure 21: Blending weights based on summed displacement magnitudes. The deformation is only defined where the amounts ϕ 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.

$$\mathbf{p}_{def} = \mathbf{p} + \frac{\sum_{i=1}^{n}(\mathbf{p}_i - \mathbf{p})\phi_i(\mathbf{p})\gamma_i(\mathbf{p})}{\sum_{i=1}^{n}\phi_i(\mathbf{p})}$$



Reference segments Translated segments

Figure 22: 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.

2.5.5 Wires: a Geometric Deformation Technique

K. Singh and E. Fiume introduce *wires*, a technique which can easily achieve a rich set of deformations with curves as control features [Singh and Fiume 1998]. Their technique is inspired by armatures used by sculptors.

A wire is defined by a quadruple $(\mathrm{R}, \mathrm{W}, s, r)$: 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 $\mathbf{p}_C$ denote the parameter value for which $\mathrm{C}(\mathbf{p}_C)$ is the closest point to $\mathbf{p}$. Let us also denote $\mathrm{C}'(\mathbf{p}_C)$ the tangent vector at that parameter value.

The reference curve, R, generates a scalar field $\mathrm{F} : \mathbb{R}^3 \mapsto [0, 1]$. 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 $\mathbf{q}$ of a point $\mathbf{p}$ influenced by a single deformation consists of three steps, illustrated in Figure 23:

- Scaling step. The scaling factor is modulated with F. The image of a point $\mathbf{p}$ after scaling is: $\mathbf{p}_s = \mathrm{R}(\mathbf{p}_R) + (\mathbf{p} - \mathrm{R}(\mathbf{p}_R))(1 + s\mathrm{F}(\mathbf{p}))$, where $\mathbf{p}_R$ denotes the parameter value for which $\mathrm{R}(\mathbf{p}_R)$ is the closest to $\mathbf{p}$.
- Rotation step. Let θ be the angle between the tangents $\mathrm{R}'(\mathbf{p}_R)$ and $\mathrm{W}'(\mathbf{p}_R)$. The point p_s is rotated around axis $\mathrm{R}'(\mathbf{p}_R) \times \mathrm{W}'(\mathbf{p}_R)$ about center $\mathrm{R}(\mathbf{p}_R)$ by the modulated angle $\theta\ \mathrm{F}(\mathbf{p})$. This results in point $\mathbf{p}_r$
- Translation step. Finally, a translation is modulated to produce the image $\mathbf{p}_{def} = \mathbf{p}_r + (\mathrm{W}(\mathbf{p}_R) - \mathrm{R}(\mathbf{p}_R))$.

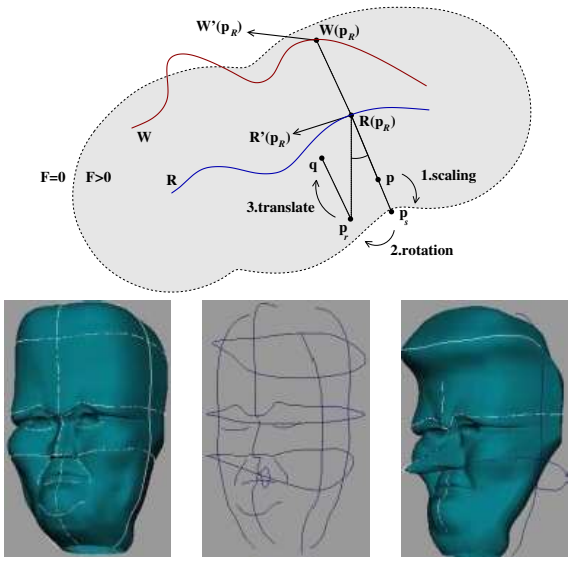
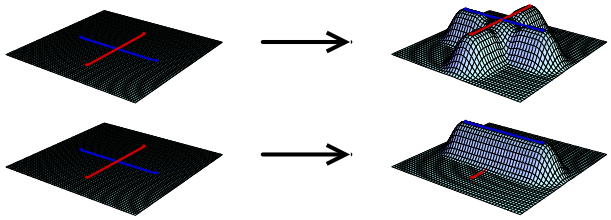


Figure 23: Left: deformation of a point by a single wire: the reference curve is in blue and the wire curve is in red. Right: deformation of a shape with multiple wires (the three images on the right by K. Singh and E. Fiume). The first image shows the initial shape, the second shows the reference curves and the third shows the wire curves and the deformed shape.

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 $\mathbf{p}_i$ the deformation of $\mathbf{p}$ by wire i. Let $\Delta\mathbf{p}_i = \mathbf{p}_i - \mathbf{p}$. The simplest deformation is:

$$\mathbf{p}_{def} = \mathbf{p} + \frac{\sum_{i=1}^{n}\Delta\mathbf{p}_i \|\Delta\mathbf{p}_i\|^m}{\sum_{i=1}^{n}\|\Delta\mathbf{p}_i\|^m}$$



Reference curves Wire curves

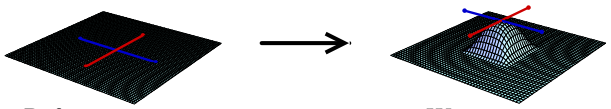
Figure 24: Blending weights based on summed displacement magnitudes. This blending is not free from artifacts: notice the creases around the intersection in the upper-right figure.

The scalar m is defined by the artist. This expression is not defined when m is negative and $\|\Delta\mathbf{p}_i\|$ 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:

In order to use unmoved wires as anchors that hold the surface, they use $\mathrm{F}_i(\mathbf{p})$ instead of $\Delta\mathbf{p}_i$ as a measure of proximity:

Other capabilities of wires can be found in the original paper [Singh and Fiume 1998]. Note that important to the algorithm is computing the distance from each curve to each deformed surface point.

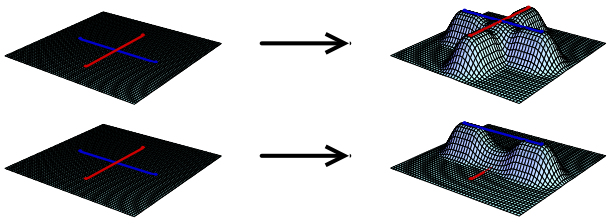
$$\mathbf{p}_{def} = \mathbf{p} + \frac{\sum_{i=1}^{n} \Delta\mathbf{p}_i \prod_{j\neq i} \|\Delta\mathbf{p}_j\|^{|m|}}{\sum_{i=1}^{n} \prod_{j\neq i} \|\Delta\mathbf{p}_j\|^{|m|}}$$



Reference curves Wire curves

Figure 25: Blending weights based on multiplied displacement magnitudes. The deformation is defined at the intersection of the reference curves.

$$\mathbf{p}_{def} = \mathbf{p} + \frac{\sum_{i=1}^{n} \Delta\mathbf{p}_i \mathrm{F}_i(\mathbf{p})^m}{\sum_{i=1}^{n} \mathrm{F}_i(\mathbf{p})^m}$$



Reference curves Wire curves

Figure 26: 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.

2.5.6 Sweepers: Modeling with Gesture

Sweepers is a frameworks for shape modeling by gesture[Angelidis et al. 2004b]. Note that there exist similar work [Gain and Marais 2005; Kil et al. 2006]. The input that defines transformations is a gesture, obtained with a mouse or hand tracking device. A simple space deformation can be defined with a 4×4 transformation matrix M (translation, rotation, scale, etc.) whose effect is spatially weighted with a scalar field $\varphi(\mathbf{p}) \in [0, 1]$. The function φ encodes the amount of transformation at $\mathbf{p} \in \mathbb{R}^3$. To define a tool, a funtion φ is defined by composing a smooth function μ to the distance to a shape.

$$\mu_\lambda(d) = \begin{cases} 0 & \text{if} \quad \lambda \leq d \\ 1 + (\frac{d}{\lambda})^3(\frac{d}{\lambda}(15 - 6\frac{d}{\lambda}) - 10) & \text{if} \quad d < \lambda \end{cases} \tag{46}$$

This function is C^2-continuous and antisymmetric about 0.5. The tools proposed in the original papers are a ball tool, a filled ellipsoid tool, or more generic mesh tools.

There are several ways to weight a transformation with a weight, and sweepers uses fractions of transformation, by using the exponential and logarithm of matrices (see a complete overview in [Alexa 2002]). Note that as opposed to the numerical method proposed by Alexa operator, sweepers *do not evaluate* exp *and* log *numerically*, since some cases reduce to more efficient and elegant closed-form formulas. Thus the transformation M can be weighted with φ as follows:

$$\ddot{f}(\mathbf{p}) = \exp(\varphi(\mathbf{p}) \log M) \cdot \mathbf{p} \tag{47}$$

The deformation $\ddot{f}$ is 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 φ onto points outside from the support of φ, thus folding space onto itself.

Single Sweeper: By decomposing the transformation into a series of s small enough transformations, and applying each of them to the result of the previous one, foldovers are avoided, for the same reason that the solution to a first order differential equation is foldover-free (there is no acceleration). The decomposition in s steps for a general transformation is expressed as follows:

$$\begin{aligned} f(\mathbf{p}) &= \bigcirc_{k=0}^{s-1} f_k(\mathbf{p}) \\ \text{where } f_k(\mathbf{p}) &= \exp(\tfrac{\varphi^k(\mathbf{p})}{s} \log M) \cdot \mathbf{p} \\ \text{and } \varphi^k(\mathbf{p}) &= \varphi(\exp(-\tfrac{k}{s} \log M) \cdot \mathbf{p}) \end{aligned} \tag{48}$$

The value returned by φ^k is that of the scalar field φ transformed by $\exp \frac{k}{s} \log M$, a fraction of M. It can be shown that there exists a finite number of steps such that the deformation is foldover-free (see [Angelidis 2005]). We propose the following as a lower bound to the required number of steps s:

$$\max_{\mathbf{p}} \|\nabla\varphi(\mathbf{p})\| \max_{l\in[1,8]} \|\log(M) \cdot \mathbf{p}_l\| < s \tag{49}$$

where $\mathbf{p}_{l\in[1,8]}$ are the corners of a box oustide which the function φ equals zero.

Efficiency: In a single tool scenario, the transformations convenient to input are translations, non-uniform and uniform scaling and rotations. In these cases, there is a closed-form to the logarithm and exponential of matrices. If M is a translation of vector $\mathbf{d}$, the minimum number of steps is:

$$\max_{\mathbf{p}} \|\gamma(\mathbf{p})\| \ \|\mathbf{d}\| < s \tag{50}$$

The s vertex and normal deformations are:

$$f_k(\mathbf{p}) = \quad \mathbf{p} + \tfrac{\varphi_k(\mathbf{p})}{s}\mathbf{d} \tag{51}$$

$$g_k(\mathbf{n}) = \quad \mathbf{n} + \tfrac{1}{s}(\gamma_k \times \mathbf{n}) \times \mathbf{d} \tag{52}$$

If M is a uniform scaling operation of center c and scaling factor σ, the minimum number of steps is:

$$\max_{\mathbf{p}} \|\gamma(\mathbf{p})\| \ \sigma \log(\sigma) d_{\max} < s \tag{53}$$

where $d_{\max}$ is the largest distance between a point in the deformed area and the center $\mathbf{c}$, approximated using a bounding box. Let $\vec{\chi} = \frac{\log(\sigma)}{s}(\mathbf{p} - \mathbf{c})$. The s vertex and normal deformations are:

$$f_k(\mathbf{p}) = \quad \mathbf{p} + (\sigma^{\frac{\varphi_k(\mathbf{p})}{s}} - 1)(\mathbf{p} - \mathbf{c}) \tag{54}$$

$$g_k(\mathbf{n}) = \quad \mathbf{n} + (\gamma_k \times \mathbf{n}) \times \vec{\chi} \tag{55}$$

If M is a rotation of angle θ, center $\mathbf{r}$ and axis $\mathbf{v} = (v_x, v_y, v_z)^\top$, the minimum number of steps is:

$$\max_{\mathbf{p}} \|\gamma(\mathbf{p})\| \ \theta r_{\max} < s \tag{56}$$

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:

$$f_k(\mathbf{p}) = \mathbf{p} + (\cos\frac{\varphi_k\theta}{s} - 1)\xi \times \mathbf{n} + \sin\frac{\varphi_k\theta}{s}\xi \quad \text{where } \xi = \mathbf{v} \times (\mathbf{p} - \mathbf{r}) \tag{57}$$

The s normal deformations are:

$$\begin{aligned} g_k(\mathbf{n}) &= (\mathbf{n} \cdot \mathbf{v})\mathbf{v} + \mathbf{v} \times (\cos(h)\mathbf{n} \times \mathbf{v} - \sin(h)\mathbf{n}) \\ &\quad + \theta\gamma \times (\mathbf{n} \times \xi \\ &\quad + ((\cos(h) - 1)(\mathbf{n} \times \xi) \cdot \mathbf{v} \\ &\quad + \sin(h)\mathbf{n} \cdot \xi)\mathbf{v}) \\ \text{where } h &= \tfrac{\varphi_k\theta}{s} \end{aligned} \tag{58}$$

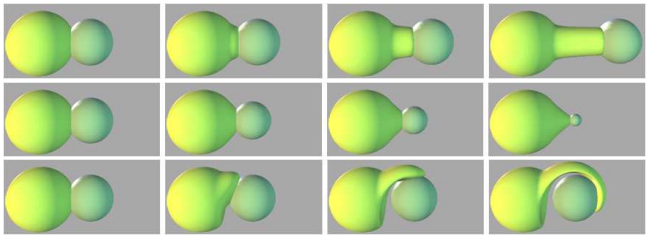


Figure 27: Translatiom, scale and rotation.

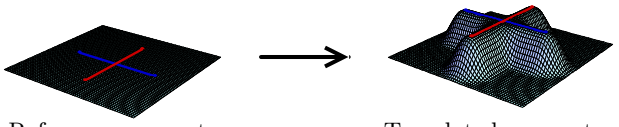
Simultaneous Sweepers: Let us consider n operations, defined with $M_{i\in[1,n]}$ and $\varphi_{i\in[1,n]}$. A naive way to achieve simultaneous deformations is

$$f(\mathbf{p}) = \exp(\sum_{i=1}^{n} \varphi_i(\mathbf{p}) \log M_i) \cdot \mathbf{p} \tag{59}$$

This function is naive because it adds the effect of each operation. The following expression provides a *normalized* and *smooth*[3] combination of all the transformations at any point $\mathbf{p}$ in space[4]:

$$\begin{cases} \mathbf{p} & \text{if } \sum_k \varphi_k = 0 \\ \exp(\sum_{i=1}^{n} \left(\frac{1-\prod_k (1-\varphi_k)}{\sum_k \varphi_k} \varphi_i \log M_i \right) \cdot \mathbf{p}) & \text{otherwise} \end{cases} \tag{60}$$

where $\frac{1}{\sum_k \varphi_k}$ is required to produce a *normalized* combination of the transformations and $1-\prod_{k=1}^{n}(1-\varphi_k(\mathbf{p}))$ *smooths* the deformation in the entire space. Figure 29 shows a comparison between additive blending of Equation (59) and the correct one of Equation (60). In Figure 28, we show the blending of sweepers in a secenario similar to other blending presented in this section.



Reference segments Translated segments

Figure 28: Blending with sweepers. The resulting surface is nice and smooth, as opposed to surfaces in Figures 24 25, 26, 21 and 22.

Equation (60) may produces foldovers for similar reasons to the case of a single tool, with Equation (47). If we decompose it into small steps, foldovers can be avoided:

$$\begin{aligned} f(\mathbf{p}) &= \bigcirc_{k=0}^{s-1} f_k(\mathbf{p}) \\ \text{where } f_k(\mathbf{p}) &= \begin{cases} \mathbf{p} & \text{if } \sum_j \varphi_j^k = 0 \\ & \text{otherwise} \\ \exp(\sum_{i=1}^{n} \left(\frac{1-\prod_j (1-\varphi_j^k)}{\sum_j \varphi_j^k} \varphi_i^k \log M_i \right)) \cdot \mathbf{p} \end{cases} \\ \text{and } \varphi_j^k(\mathbf{p}) &= \exp(\varphi_j((\tfrac{k}{s} \log M_j^{-1})) \cdot \mathbf{p} \end{aligned} \tag{61}$$

The following expression is a lower bound to the required number of steps, generalizing the single tool condition (see justification in [Angelidis 2005]):

$$\sum_j \max_{\mathbf{p}}(\|\nabla\varphi_j(\mathbf{p})\|) \max_{l\in[1,8]} \left|\left|\log M_j \cdot \mathbf{p}_{l_j}\right|\right| < s \tag{62}$$

[3] as smooth as the φ_i.

[4] The operator $\bigoplus$ expresses a repetive sum: $\bigoplus_{i=1}^{n} M_i = M_1 \oplus M_2 \oplus \cdots \oplus M_n$.

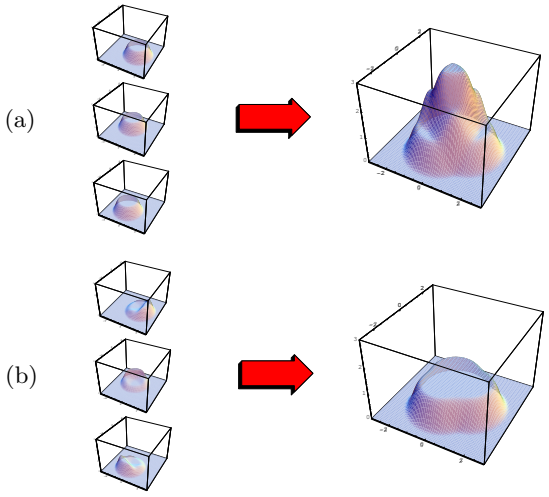


Figure 29: Blending of three scalar fields. To illustrate the behaviour of our blending in this figure, we directly combine the scalar fields instead of using them to modulate a transformation. (a) Adding the scalar fields. (b) By multiplying each field with $(1-\prod(1-\varphi_k))/\sum\varphi_k$, the sum of the fields is normalized.

where $\mathbf{p}_{l_j\in[1,8]}$ are the corners of a bounding box oustide which the function φ_j equals zero. For an operation symmetric about a plane, the transformation matrices are of the same type, thus blending them leads to simple expressions (see [Angelidis 2005]).

The set of possible deformations with sweepers is quite large 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.

2.5.7 Swirling-sweepers: Constant Volume Modeling

In a non-virtual modeling context, one of the most important factors which affects the artist's technique is the amount of available material. The notion of an amount of material is not only familiar to professional artists, but also to children who 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 increase the intuitiveness of use. In Swirling-Sweepers, the artist inputs a position $\mathbf{h}$ and translation $\mathbf{t}$, and the technique will create a deformation that transforms $\mathbf{h}$ into $\mathbf{h}+\mathbf{t}$ while the volume of the shape is preserved implicitly, simply because the deformation satisfies a differential property. Thus the volume of the shape does not require to be computed, and the deformation can be applied to an open surface.

Swirling-sweepers use *swirls* as building blocks. A swirl twists space locally without compression or dilation (see proof in [Angelidis et al. 2004a]): it preserves volume. A swirl is defined by a point $\mathbf{c}$, a rotation of angle θ around an axis $\mathbf{v}$ (see Figure 31), and a scalar function φ describing

the amount of swirl

$$\varphi(\mathbf{p},\lambda) = \mu(\frac{\|\mathbf{p}-\mathbf{c}\|}{\lambda}) \quad (63)$$

$$\text{where } \mu(d) = \begin{cases} 0 & \text{if } 1 \le d \\ 1 + d^3(d(15-6d)-10) & \text{if } 1 > d \end{cases}$$

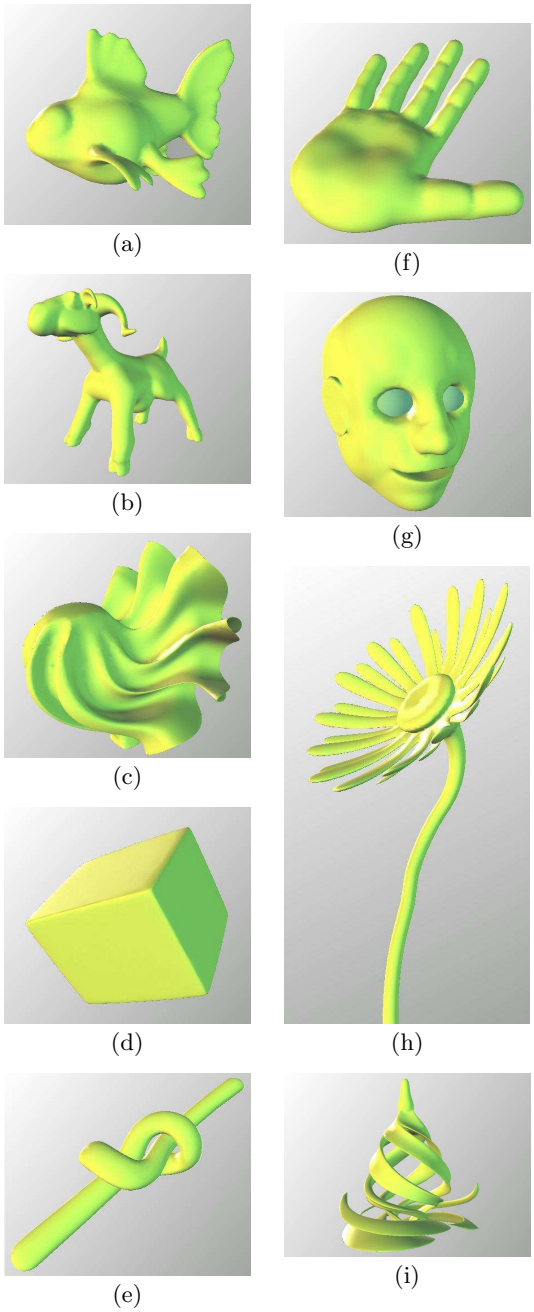


Figure 30: All these shapes were modeled starting with a sphere, in at most one hour. In (c), the first modeling step was to squash the sphere into a very thin disk. In (g), eyeballs were added.

The effect of a swirl is defined using the exponential and logarithm of matrices (see a complete overview in **??**), for which there are *closed-forms* given bellow:

$$f(\mathbf{p}) = \exp(\varphi(\mathbf{p},\lambda)\log R)\cdot\mathbf{p} \quad (64)$$

where R denote the 4×4 matrix of a rotation of center $\mathbf{c}$, axis $\mathbf{v}$ and angle θ. This equation could have been defined in several equivalent ways, for example using quaternions or an algebraic formula. The above notation is however convenient to combine multiple swirls:

$$f(\mathbf{p}) = \left(\exp(\sum_{i=0}^{n-1}(\varphi_i(\mathbf{p},\lambda)\ \log R_i))\right)\cdot\mathbf{p} \quad (65)$$

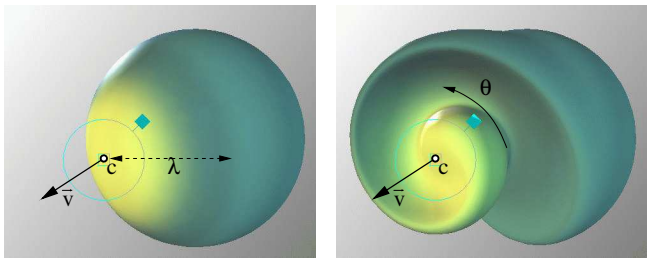


Figure 31: The effect on a sphere of a swirl centered at c, with a rotation angle θ around $\vec{v}$. The two shapes have the same volume.

We now define how to place the above swirls to transform $\mathbf{h}$ into $\mathbf{h}+\mathbf{t}$. Let us consider n points, $\mathbf{c}_i$, on the circle of center $\mathbf{h}$, and radius r lying in a plane perpendicular to $\mathbf{t}$. To these points correspond n consistently-oriented unit tangent vectors $\mathbf{v}_i$ (see Figure 32). Each pair, $(\mathbf{c}_i,\mathbf{v}_i)$, together with an angle, θ_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 θ_i will transform $\mathbf{h}$ exactly into $\mathbf{h}+\mathbf{t}$ (see justification in [Angelidis et al. 2004a]):

$$\theta_i = \frac{2\|\mathbf{t}\|}{nr} \quad (66)$$

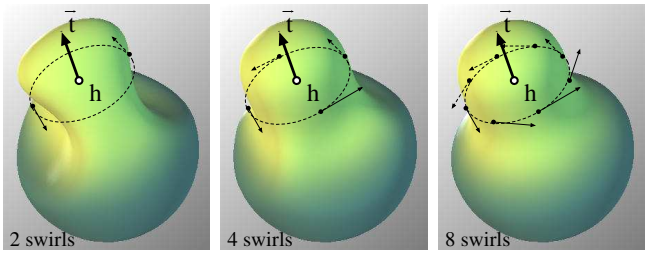


Figure 32: By arranging n basic swirls in a circle, a more complex deformation is achieved. In the rightmost image: with 8 swirls, there are no visible artifacts due to the discrete number of swirls.

We show in Figure 32 the effect of the tool for different values of n; in practice, we use 8 swirls. If the magnitude of the input vector $\mathbf{t}$ is too large, the deformation of Equation (65) will produce a self-intersecting surface, and will not preserve volume accurately. To correct this, it is necessary to subdivide $\mathbf{t}$ into smaller vectors for the same reasons that applies to solving discretely a first order differential equations. The number of steps must be proportional to the speed and inversely proportional to the size of the tool. We use:

$$s = \max(1, \lceil 4\|\mathbf{t}\|/r\rceil) \quad (67)$$

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. We summarize here the swirling-sweepers algorithm:

Input point $\mathbf{h}$, translation $\mathbf{t}$, and radius r
Compute the number of required steps s
Compute the angle of each step, $\theta_i = \frac{2\|\mathbf{t}\|}{nrs}$
for each step k from 0 to $s-1$ **do**
  for each point $\mathbf{p}$ in the tool's bounding box **do**
    $M = 0$
    for each swirl i from 0 to $n-1$ **do**
      $M \mathrel{+}= \varphi_k^i(\mathbf{p}) \log R_{i,k}$
    end for
    $\mathbf{p} = (\exp M) \cdot \mathbf{p}$
  end for
end for

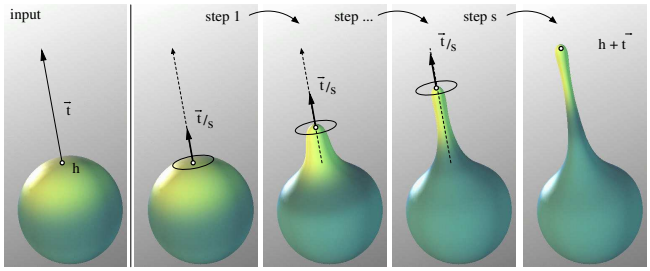


Figure 33: A volume preserving deformation is obtained by decomposing a translation into circles of swirls. 3 steps have been used for this illustration. As the artist pulls the surface, the shape gets thinner. The selected point's transformation is precisely controlled.

The point $\mathbf{c}_{ik}$ denotes the center of the i^{th} swirl of the k^{th} ring of swirls. For efficiency, a table of the basic-swirl centers, $\mathbf{c}_{ik}$, and a table of the rotation matrices, $\log R_{i,k}$, are precomputed. We have a closed-form for the logarithm of the involved matrix, given in Equations (68) and (69), saving an otherwise expensive numerical approximation:

$$\begin{aligned} \mathbf{n} &= \theta_i \mathbf{v}_i \\ \mathbf{m} &= \mathbf{c}_{i,k} \times \mathbf{n} \end{aligned} \tag{68}$$

$$\log M_{i,k} = \begin{pmatrix} 0 & -n_z & n_y & m_x \\ n_z & 0 & -n_x & m_y \\ -n_y & n_x & 0 & m_z \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{69}$$

Since these matrices almost antisymmetric, they are handled as pairs of vectors, $(\mathbf{n}, \mathbf{m})$. Once M is computed, we use a closed-form for computing $\exp M$. Since the matrix M is a weighted sum of matrices $\log R_{i,k}$, the matrix M is of the form of Equation (69), and can be represented with a pair $(\mathbf{n}_M, \mathbf{m}_M)$. If $\mathbf{n}_M = 0$, then $\exp M$ is a translation of vector $\vec{m}_M$. Else, if the dot product $\mathbf{m}_M \cdot \mathbf{n}_M = 0$, then $\exp M$ is a rotation of center $\mathbf{c}$, angle θ axis $\mathbf{v}$, as given by Equation (70):

$$\begin{aligned} \mathbf{c} &= \frac{\omega \times \mathbf{m}}{\|\omega\|^2} \\ \theta &= \|\mathbf{n}_M\| \\ \mathbf{v} &= \mathbf{n}_M / \theta \end{aligned} \tag{70}$$

Finally, in the remaining cases, we denote $l = \|\vec{n}_M\|$, and we use Equation (71) (see Appendix 2.5.7 for efficiency):

$$\exp M = \mathrm{I} + M + \frac{1-\cos l}{l^2} M^2 + \frac{l - \sin l}{l^3} M^3 \tag{71}$$

Symmetrical objects can be easily modeled by introducing a plane of symmetry about which the tool is reflected. The shapes shown in Figure 34 were modeled in real-time in *half an hour* at most, and were all made starting with a sphere. For instance 80 swirling-sweepers have been used to model the alien.

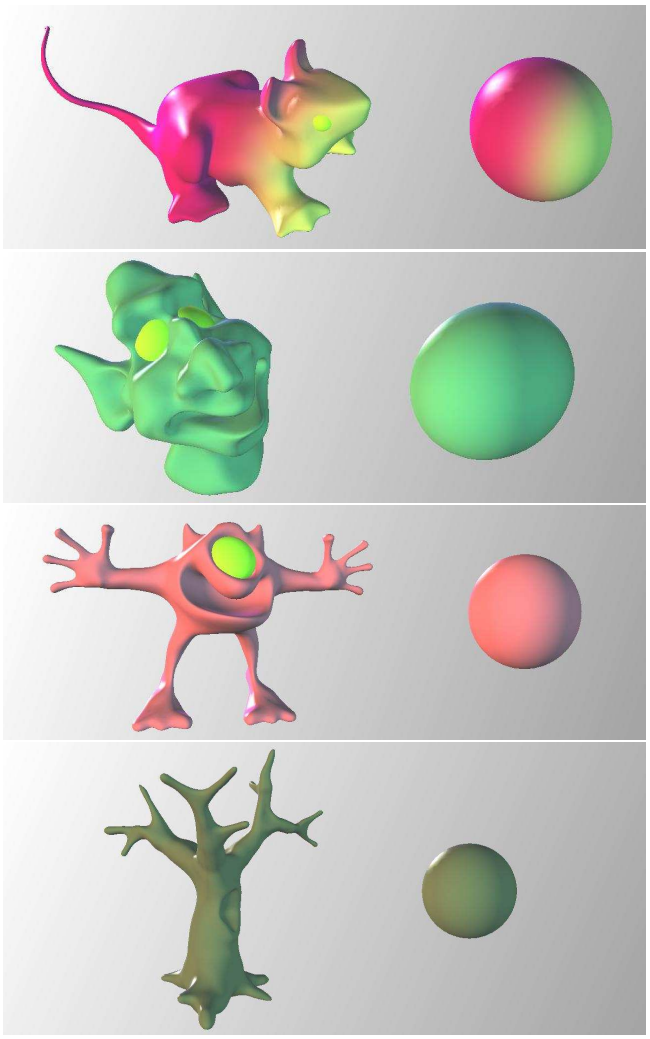


Figure 34: Examples of models modeled with swirling-sweepers. The mouse, the goblin, the alien and the tree have respectively 27607, 25509, 40495 and 38420 vertices. These objects were modeled in less than 30 min by one of the authors. Eyeballs have been added. The shapes volumes are respectively 101.422%, 99.993%, 101.158% and 103.633% of the initial sphere, due to the finite number of steps, and to our choice of shape representation.

3 Spape Deformation and Modeling

3.1 Desirable Properties for Modeling

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. Here a few guidelines for designing 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 [Barr 1984; Blanc 1994; Chang and Rockwood 1994; Lazarus et al. 1994]. Controlling the effect of global deformations using weights would requires a certain amount of craftsmanship from the artist.

Secondly, a large number of space deformation techniques requires the artist to specify a rather large number of control parameters [Sederberg and Parry 1986; Coquillart 1990; MacCracken and Joy 1996; Moccozet and Magnenat-Thalmann 1997; Hirota et al. 1999; Hua and Qin 2004]. We believe that for modeling, 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 [Decaudin 1996; Hsu et al. 1992; Llamas et al. 2003]. The absence of a model entirely developed in one piece with a single technique may be 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 couple of exceptions [Mason and Wyvill 2001; Gain and Dodgson 2001]. 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 the following are reasonable guidelines for deformation operations for shape modeling:

- 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.
- It should be sufficiently fast for existing computing devices.

3.2 A Shape Description for Modeling

Because space deformations operations are independent from the shape description, several choices are available to represent a shape being deformed: mesh [Gain and Dodgson 1999], particles [Pauly et al. 2003], deformed raytracing [Barr 1984], hybrid [Enright et al. 2002], and all the popular shape descriptions: subdivision surfaces, NURBS and more. 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. This section presents a shape description for interactive modeling which supports high deformation and does not break when highly stretched [Angelidis et al. 2004b].

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.

The presented method uses 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 C0 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 is an active area of research [Pauly et al. 2003].

The possibly large number of deformations applied by an artist requires some minimum surface sampling density. In order to maintains this density, the presented method requires the deformation to be capable of being split into substeps.

Let us assume the scene is initialized with a polygonal model, e.g. a sphere with a homogeneous density of nearly equilateral triangles. 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 Equation 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 [Gain and Dodgson 1999]. 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 vertices and normals should be able to represent them well enough. If this is not the case after deformation, then it means the surface is undersampled. 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 $(\mathbf{v}_0, \mathbf{v}_1)$ with normals $(\mathbf{n}_0, \mathbf{n}1)$, and the deformed edge e' defined by vertices $(\mathbf{v}'_0, \mathbf{v}'_1)$ with normals $(\mathbf{n}'_0, \mathbf{n}'_1)$. In addition to the conditions in [Gain and Dodgson 1999] based on edge length and angle between normals, we also base the choice of splitting edge e_0 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_0 well enough, then edge e is collapsed, and the new vertex is deformed. We define the fictitious vertex as the mid-vertex of a C^1 curve, since vertices and normals only provide 1^{rst} order information about the surface. The following cubic polynomial curve interpolates the vertices $\mathbf{v}'_0$ and $\mathbf{v}'_1$ with

corresponding shape tangents $\mathbf{t}_0$ and $\mathbf{t}_1$, defined below:

$$\begin{aligned}\mathbf{c}(u) &= (\mathbf{v}'_0(1+2u)+\mathbf{t}_0 u)(1-u^2)+ \\ &\quad (\mathbf{v}'_1(1+2(1-u))-\mathbf{t}_1(1-u))(1-(1-u)^2)\end{aligned} \quad (72)$$

The only constraint on tangent $\mathbf{t}_i$ is to be perpendicular to the corresponding normal $\mathbf{n}_i$. The following choice defines tangents of magnitude proportional to the distance between the vertices:

$$\begin{aligned}\mathbf{t}_0 &= \mathbf{g}-\mathbf{g}\cdot\mathbf{n}'_0\ \mathbf{n}'_0 \\ \mathbf{t}_1 &= \mathbf{g}-\mathbf{g}\cdot\mathbf{n}'_1\ \mathbf{n}'_1 \quad \text{where } \mathbf{g}=\mathbf{v}'_1-\mathbf{v}'_0\end{aligned} \quad (73)$$

With the above tangents, the expression of the middle vertex simplifies:

$$\mathbf{c}(0.5) = (\mathbf{v}'_0+\mathbf{v}'_1+(\mathbf{g}\cdot\mathbf{n}'_0-\mathbf{g}\cdot\mathbf{n}'_1)/4)/2 \quad (74)$$

With the fictitious vertex $\mathbf{c}(0.5)$, the tests to decide whether an edge should be split or collapsed can now be defined:

Too-long edge: An edge e0 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 λ_j is greater than $L_{\max}$.
- The angle between the normals $\mathbf{n}'_0$ and $\mathbf{n}'_1$ is larger than a constant $\theta_{\max}$. This condition keeps a minimum curvature sampling.
- 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.

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 $\mathbf{n}'_0$ and $\mathbf{n}'_1$ 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 35 shows a twist on a simple U-shape. Figure 36 shows the algorithm preserving a fine triangulation only where required. Figure 37 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
```

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 renements and simplications 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.

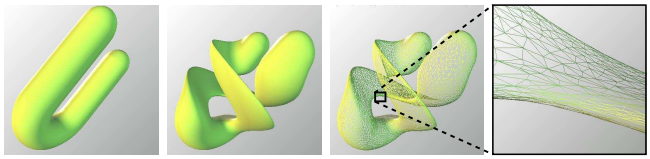


Figure 35: Example of our mesh-updating algorithm on a highly twisted U-Shape. The close-up shows a sharp feature, with finer elongated triangles.

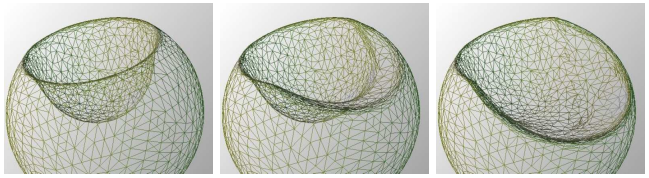


Figure 36: Behaviour of our mesh-updating algorithm on an already punched sphere. The decimation accompanying the second puch simplifies the small triangles of the first punch. The tool has been removed for a better visualization.

4 Comparing Techniques

Space deformations can be compared according to several criteria, thus the sequential presentation of Section 2 does not give the entire picture of the landscape of space deformations. We will identify objective criteria to attempt comparing techniques on a fair ground.

- *Modeling philosophy*: for the task of deforming a shape, the intended usage of a technique can be either to use many simple deformations, or a few but complex deformations. For example, a deformation that require the user to define a complex control structure will most likely be of the "few but complex deformations" kind. These approaches tend to be most efficient in the context of animation.

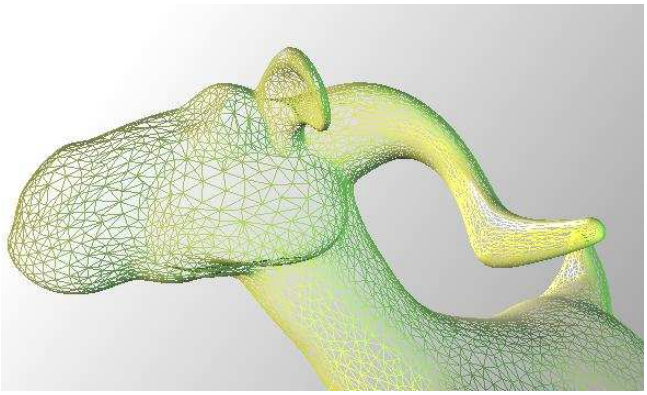


Figure 37: Close-up of the goat. Notice the large triangles on the cheek and the fine ones on the ear. The initial shape is a sphere.

- *Connectivity of control space*: deformations define several control parameters of type position, direction, affine transform, thickness, and more. Some of these controls have a direct relation with the Euclidean space, and we accord more importance to these since they are manipulated by the user in a geometric sense. The connectivity of the Euclidean controls can be $0D$, $1D$, $2D$ or $3D$, corresponding to the notion of parameter/point, a curve, a surface, or a block of jello. For example, a mouse can handle a $0D$ control, and will need to be used repeatedly to control higher dimensions, as opposed to a curve control interface [Grossman et al. 2003] that can contol a $1D$ control space all at once. Note that this is different from the dimension of the input related to hardware limitations: e.g. a mouse inputs $2D$ coordinates, while a curve control interface inputs $3D$ coordinates.

- *Free Control Blending*: all deformations can be combined together by combining the deformations with a partition of unity defined in space (see Section 1.1). Some deformation techniques include geometric blending more strongly in their formalism, and define blending methods that provides a variety of user control and a level of freedom in placing the control handles. These methods are have advantages as techniques with $3D$ connectivity control space, without any cumbersome structural constrain.

- *Differential properties*: by taking into account the time parameter, a deformation can be understood as a continuum deformation. By satisfying some differential properties, a deformation can implicitly preserve some properties of the shape being deformed such as surface self-interection avoidance, preserving volume or dynamics.

5 Conclusion

Space deformation is a set of very generic techniques that may be used in the context of modeling, rendering [Coleman and Singh 2004; Mei et al. 2005], animation and simulation. This overview focuses on the context of shape modeling, and we also present a method for representing the shape being deformed. Recent work has shown that it is possible to define properties of the shape with differential properties of the deformation. This may be a promissing direction for future work. Also, the similarity between space deformation and vector fields makes space deformation a pedagogical tool for understanding continuum mechanics, and they can be somehow used for handcrafting physical phenomena. For example, swirling-sweepers [Angelidis et al. 2004a] shares similarities with vortex-based smoke simulation [Angelidis and Neyret 2005].

Acknowledgments Many thanks to Marie-Paule Cani, Geoff Wyvill and Scott King for their contribution to the work presented in this chapter.

References

Alexa, M. 2002. Linear Combination of Transformations. *ACM Trans. Graph. 21*, 3 (Jul), 380–387.

Angelidis, A., and Neyret, F. 2005. Simulation of Smoke Based on Vortex Filament Primitives. In *SCA'05: Proc. of the 2005 Symposium on Computer Animation*, 87–96.

Angelidis, A., Cani, M.-P., Wyvill, G., and King, S. 2004. Swirling-sweepers: Constant-volume modeling. In *Pacific Graphics 2004*, IEEE, 10–15. Best paper award at PG04.

Angelidis, A., Wyvill, G., and Cani, M.-P. 2004. Sweepers: Swept user-defined tools for modeling by deformation. In *Proceedings of Shape Modeling and Applications*, IEEE, 63–73. Best paper award at SMI04.

Angelidis, A. 2005. *Shape Modeling by Swept Space Deformation*. PhD thesis, University of Otago.

Bajaj, C., Blinn, J., Bloomenthal, J., Cani-Gascuel, M.-P., Rockwood, A., Wyvill, B., and Wyvill, G. 1997. *Introduction to Implicit Surfaces*. Morgan-Kaufmann.

Barr, A. 1984. Global and Local Deformations of Solid Primitives. In *ACM Trans. Graph. (Proc of SIGGRAPH'84)*, 21–30.

Blanc, C. 1994. A generic implementation of axial procedural deformation techniques. In *Graphics Gems*, vol. 5, 249–256. Academic Press.

Bloomenthal, J. 1990. Calculation of reference frames along a space curve. *Graphics gems*, 567–571.

Borrel, P., and Bechmann, D. 1991. Deformation of n-dimensional objects. In *Proceedings of the first symposium on Solid modeling foundations and CAD/CAM applications*, 351–369.

Borrel, P., and Rappoport, A. 1994. Simple constrained deformations for geometric modeling and interactive design. In *ACM Transactions on Graphics*, vol. 13(2), 137–155.

Capell, S., Green, S., Curless, B., Duchamp, T., and Popović, Z. 2002. Interactive Skeleton-Driven Dynamic Deformations. *ACM Trans. Graph. 21*, 3 (Jul), 586–593.

Chang, Y.-K., and Rockwood, A. P. 1994. A generalized de Casteljau approach to 3d free-form deformation. In *Proceedings of SIGGRAPH'94*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, ACM, 257–260.

Reference	aka	Section	Phylosophy	Connectivity	Blendable	Differential properties
[Barr 1984]		2.1.1		0D	✗	
[Blanc 1994]		2.1.2		0D	✗	
[Decaudin 1996]		2.1.3	m/s	0D	✗	
[Kurzion and Yagel 1997]	Ray-deflectors	2.1.4	m/s	0D	✗	
[Llamas et al. 2003]	Twister	2.1.5	m/s	0D	✗	
[Chang and Rockwood 1994]		2.2.1	f/c	1D	✗	
[Lazarus et al. 1994]		2.2.2	f/c	1D	✗	
[Crespin 1999]		2.2.2	f/c	1D,2D	✗	
[Mason and Wyvill 2001]	Blendeformer	2.2.3	m/s	0D,1D	✗	foldover-free
[Capell et al. 2002]		2.2.4	f/c	1D	✗	dynamics
[Singh and Kokkevis 2000]	SOFFD	2.3.1	f/c	2D	✗	foldover-free
[Sederberg and Parry 1986]	FFD	2.4.1	f/c	3D	✗	
[Coquillart 1990]	EFFD	2.4.2	f/c	3D	✗	
[Gain and Dodgson 2001]		2.4.3	m/s	3D	✗	foldover-free
[MacCracken and Joy 1996]	SFFD	2.4.4	f/c	3D	✗	
[Hua and Qin 2004]	SFD	2.4.5	f/c	3D	✗	
[Borrel and Bechmann 1991; Hsu et al. 1992]	DMFFD	2.5.1	m/s	0D	✓	
[Borrel and Rappoport 1994]	scodef	2.5.2	f/c	3D	✓	
[Moccozet and Magnenat-Thalmann 1997]	DFFD	2.5.3	f/c	0D	✓	
[Crespin 1999]	IFFD	2.5.4	m/s	0D	✓	
[Singh and Fiume 1998]	Wires	2.5.5	f/c	1D	✓	
[Angelidis et al. 2004b]	Sweepers	2.5.6	m/s	0D	✓	foldover-free
[Gain and Marais 2005]		2.5.6	m/s	0D	✓	foldover-free
[Angelidis et al. 2004a]	Swirling-sweepers	2.5.7	m/s	0D	✓	volume-preserving

Table 1: f/c means "few complex" and m/s means "many simple". All deformations are blendable in a sense, thus blendable above means that the deformation provide additionnal blending features.

Coleman, P., and Singh, K. 2004. Ryan: rendering your animation nonlinearly projected. In *NPAR '04*, ACM, 129–156.

Coquillart, S. 1990. Extended free-form deformation: A sculpturing tool for 3d geometric modeling. In *Proceedings of SIGGRAPH'90*, ACM Press / ACM SIGGRAPH, vol. 24(4) of *Computer Graphics Proceedings, Annual Conference Series*, ACM, 187–195.

Crespin, B. 1999. Implicit free-form deformations. In *Proceedings of the Fourth International Workshop on Implicit Surfaces*, 17–24.

Decaudin, P. 1996. Geometric deformation by merging a 3d object with a simple shape. In *Graphics Interface*, 55–60.

Enright, D., Fedkiw, R., Ferziger, J., and Mitchell, I. 2002. A Hybrid Particle Level Set Method for Improved Interface Capturing. *J. Comput. Phys. 183*, 1, 83–116.

Farin, G. 1990. Surfaces over Dirichlet tessellations. *Computer Aided Geometric Design 7(1-4)* (June), 281–292.

Gain, J., and Dodgson, N. 1999. Adaptive refinement and decimation under free-form deformation. *Eurographics'99 7*, 4 (April), 13–15.

Gain, J., and Dodgson, N. 2001. Preventing self-intersection under free-form deformation. *IEEE Transactions on Visualization and Computer Graphics 7*, 4 (October-December), 289–298.

Gain, J., and Marais, P. 2005. Warp sculpting. *IEEE Transactions on Visualization and Computer Graphics 11(2)* (Apr), 217–227.

Griessmair, J., and Purgathofer, W. 1989. Deformation of solids with trivariate b-splines. In *Eurographics Conference Proceedings*, Elsevier Science, 137–148.

Griffiths, D. J. 1999. *Introduction to Electrodynamics*, third ed. Prentice Hall.

Grossman, T., Balakrishnan, R., and Singh, K. 2003. An interface for creating and manipulating curves using a high degree-of-freedom input device. In *CHI 2003 Conference Proceedings*, 185–192.

Hirota, G., Maheshwari, R., and Lin, M. 1999. Fast volume-preserving free form deformation using multi-level optimization. In *Proceedings of the fifth ACM symposium on Solid modeling and applications*, ACM, 234–245.

Hsu, W. M., Hughes, J. F., and Kaufman, H. 1992. Direct manipulation of free-form deformations. In *Proceedings of SIGGRAPH'92*, ACM Press / ACM SIGGRAPH, vol. 26(2) of *Computer Graphics Proceedings, Annual Conference Series*, ACM, 177–184.

Hua, J., and Qin, H. 2004. Scalar-field-guided adaptive shape deformation and animation. *The Visual Computer 1*, 1 (April), 47–66.

Kil, Y., Renzulli, P., Kreylos, O., Hamann, B., Monno, G., and Staadt, O. 2006. 3d warp brush modeling. *Journal of Computer and Graphics, ELSEVIER 30(4)*.

KURZION, Y., AND YAGEL, R. 1997. Interactive space deformation with hardware assisted rendering. *IEEE Computer Graphics and Applications 17(5)* (September/October), 66–77.

LAZARUS, F., COQUILLART, S., AND JANCÈNE, P. 1994. Axial deformations: an intuitive deformation technique. In *Computer-Aided Design*, vol. 26(8), 607–613.

LLAMAS, I., KIM, B., GARGUS, J., ROSSIGNAC, J., AND SHAW, C. 2003. Twister: A space-warp operator for the two-handed editing of 3d shapes. In *SIGGRAPH*, vol. 22(3) of *ACM Transactions on Graphics, Annual Conference Series*, ACM, 663–668.

MACCRACKEN, R. A., AND JOY, K. I. 1996. Free-form deformations with lattices of arbitrary topology. In *Proceedings of SIGGRAPH'96*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, ACM, 181–188.

MASON, D., AND WYVILL, G. 2001. Blendeforming: Ray traceable localized foldover-free space deformation. In *Proceedings of Computer Graphics International (CGI)*, 183–190.

MEI, C., POPESCU, V., AND SACKS, E. 2005. The occlusion camera. In *Eurographics 2005*, vol. 24(3).

MOCCOZET, L., AND MAGNENAT-THALMANN, N. 1997. Dirichlet free-form deformation and their application to hand simulation. In *Computer Animation'97*, 93–102.

PAULY, M., KEISER, R., KOBBELT, L., AND GROSS, M. 2003. Shape modeling with point-sampled geometry. In *Proceedings of SIGGRAPH'03*, vol. 22(3), ACM, 641–650.

RUTHERFORD, A. 1990. *Vectors, Tensors and the Basic Equations of Fluid Mechanics*. Dover.

SCHEIN, S., AND ELBER, G. 2004. Discontinuous free form deformations. In *Proceedings of Pacific Graphics*, IEEE, 227–236.

SEDERBERG, T., AND PARRY, S. 1986. Free-form deformation of solid geometric models. In *Proceedings of SIGGRAPH'86*, ACM Press / ACM SIGGRAPH, vol. 20(4) of *Computer Graphics Proceedings, Annual Conference Series*, ACM, 151–160.

SINGH, K., AND FIUME, E. 1998. Wires: a geometric deformation technique. In *Computer graphics, Proceedings of SIGGRAPH'98*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, ACM, 405–414.

SINGH, K., AND KOKKEVIS, E. 2000. Skinning Characters using Surface Oriented Free-Form Deformations". In *Graphics Interface*, 35–42.

WEISSTEIN, E. Lagrange multipliers. From Mathworld – A Wolfram Web Ressource http://mathworld.wolfram.com/LagrangeMultipliers.html.

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.

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 [Stam 1998; Zorin and Kristjansson 2002], editing [Biermann et al. 2001; Biermann et al. 2002a; Biermann et al. 2002b; Biermann et al. 2000], texturing [Piponi and Borshukov 2000], and conversion to other popular representations [Peters 2000] have been devised and hardware support for rendering of subdivision surfaces has been proposed [Boo et al. 2001; Bischoff et al. 2000; Pulli and Segal 1996].

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 [Dyn and Levin 2002; Sabin 2002] 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 [Zorin et al. 2000] 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 [Zorin et al. 2000; Warren and Weimer 2001].

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 [Doo and Sabin 1978], Catmull-Clark [Catmull and Clark 1978], Loop [Loop 1987], Butterfly [Dyn et al. 1990; Zorin et al. 1996], Kobbelt [Kobbelt 1996a], Midedge [Peters and Reif 1997]. Different schemes lead to limit surfaces with different smoothness characteristics. For design purposes, the Catmull-Clark [Catmull and Clark 1978], Loop [Loop 1987] 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 [Lounsbery et al. 1997; Pulli and Lounsbery 1997; Zorin et al. 1997]. Figure 2 illustrates the power of multiresolution in capturing complex shapes.

A close connection exists between multiresolutin subdivision and wavelets [Stollnitz et al. 1996]. 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*

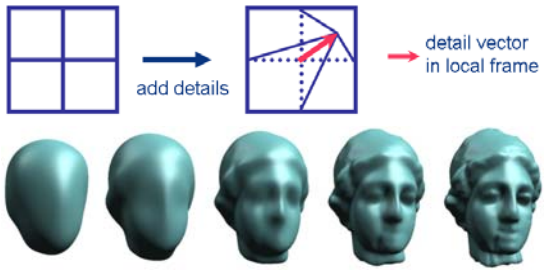


Figure 2: Top: multiresolution subdivision extends the concept of subdivision by introducing detail vectors at each level. Bottom: surfaces obtained by subdivision of the same coarse mesh look very different depending on the amount of detail introduced and the level at which it is introduced. From left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

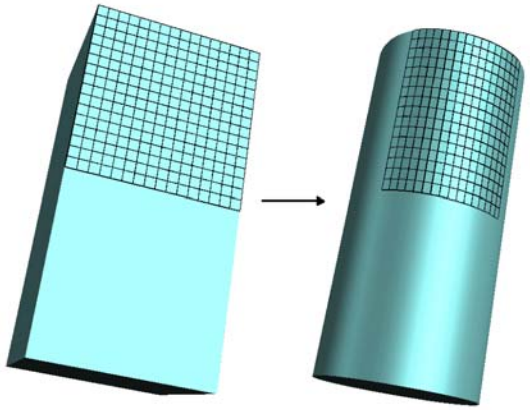


Figure 3: Natural parameterization of a subdivision surface. Each time we apply the subdivision rules to compute the finer control mesh we also apply midpoint subdivision to a copy of the initial control mesh. A mapping from a denser and denser subset of the control polyhedron (left) to the control points 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.

reconstructs the data on level i by subdividing the control mesh of level $i-1$ and adding the details [Zorin et al. 1997].

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 [Sabin 1971] and Bezier [Bézier 1974] 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., [Barr 1984; Sederberg and Parry 1986; Coquillart 1990; Halstead et al. 1993; Chang and Rockwood 1994; MacCracken and Joy 1996] [Singh and Fiume 1998; Kobbelt 1996b; Zorin et al. 1997; Pulli and Lounsbery 1997; Qin et al. 1998; Takahashi 1998; Weimer and Warren 1998] [McDonnell and Qin 2000; Turk and O'Brien 2002; Grinspun and Schröder 2001; Boier-Martin et al. 2004]).

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 [Barr 1984; Chang et al. 1994; Lazarus et al. 1994] which alter the axis of a shape to induce its deformation, lattice deformations [Sederberg and Parry 1986; Coquillart 1990; MacCracken and Joy 1996] which operate on the cells of a space lattice to deform the volume inside the lattice, manipulations on scalar field embeddings [Hua and Qin 2003], control mesh editing methods which shape parameterically-defined surfaces by imposing constraints on their control meshes [Zorin et al. 1997], and variational methods which operate by optimizing an energy functional over the surface under constraints [Takahashi 1998; Boier-Martin et al. 2004].

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 [Gibson and Mirtich 1997].

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 [Cohen et al. 2001] 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.

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 [Zorin et al. 1997; Pulli and Lounsbery 1997] for interactive multiresolution editing of Loop surfaces and in [DeRose et al. 1998] 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

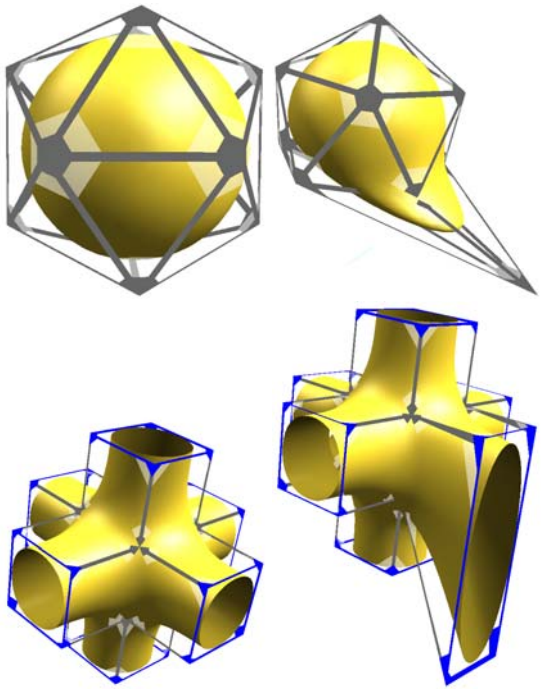


Figure 4: Shape modeling through control point manipulation: Loop subdivision surface (top), Catmull-Clark subdivision surface (bottom).

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 [Zorin et al. 1997]: 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* [Lee et al. 2000] and subdivision surface *fitting* [Suzuki et al. 1999; Litke et al. 2001a; Ma and Zhao 2000]. 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 [Loop 1987]. 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 [Barr 1984; Terzopoulos and Fleischer 1988; Welch and Witkin 1992]. Celniker and Gossard [Celniker and Gossard 1999] and later Welch and Witkin [Welch and Witkin 1992] 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 [Halstead 1996]. 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 [Terzopoulos et al. 1987].

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

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

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

where Ω denotes the parametric domain of the surface S. Most variational approaches take advantage of these expressions, although alternative approaches have been proposed (e.g., [Cirak et al. 2002]). The main differences are in the types of parameterizations used to derive the differential quantities. For example, Greiner [Greiner 1994] and later Kobbelt [Kobbelt 1996a] 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

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

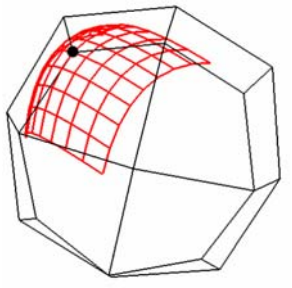


Figure 6: Local quadratic interpolant used to approximate first and second order derivatives [Boier-Martin et al. 2004].

In the context of subdivision surfaces, Halstead et al. [Halstead et al. 1993] 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.

Kobbelt [Kobbelt 1996b] 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 [Weimer and Warren 1998] 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 [Diewald et al. 2002]. 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. [Friedel et al. 2003] 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. [Boier-Martin et al. 2004] in which the trivial solution corresponds to the input surface). We distinguish between two classes of constraints [Welch and Witkin 1992]:

- *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 [Welch and Witkin 1992] or by reparameterizing the surface to align control points or edges with constraints [Boier-Martin et al. 2004] (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* [Levin 1999] (see also [Nasri 2000; Nasri and Abbas 2002; Schaeffer et al. 2004]).

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 [Kobbelt 2000] 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. [Takahashi 1998] 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 [Boier-Martin et al. 2004] 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 single point). One solution is to separate the high-frequency information before optimization and to "re-apply" it to the new shape [Kobbelt 2000]. This introduces an overhead related to saving and restoring surface details. To avoid this overhead, Boier-Martin et al. [Boier-Martin et al. 2004] 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.

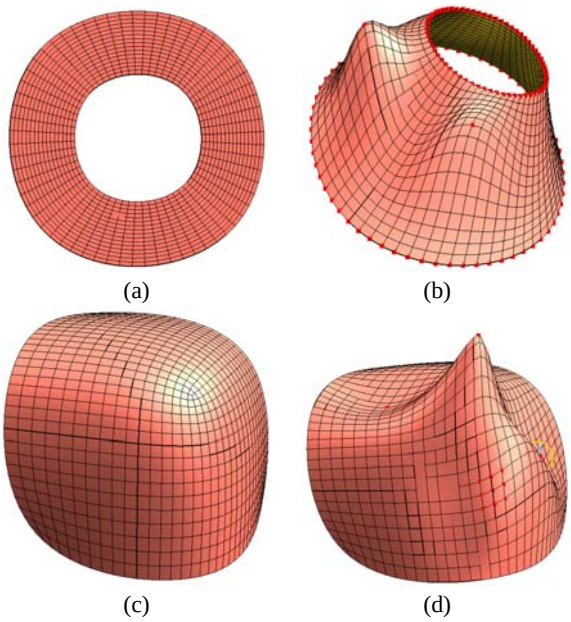


Figure 7: Constraint types: (a)-(b) point and discretized curve constraints; (c)-(d) normal constraints.

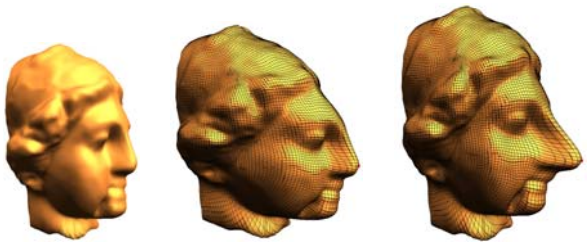


Figure 8: Region of influence of a multiresolution constraint: left – input model; middle – constraint is propagated to the coarsest subdivision level, inducing a global deformation of the head by pulling a single point on the nose; right – constraint is propagated only two levels coarser inducing a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

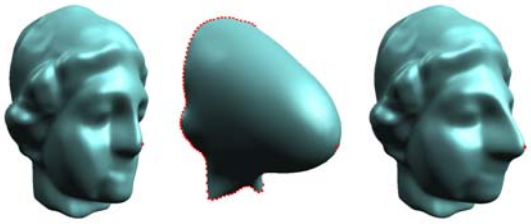


Figure 9: Energy optimization with constraints: left – input multiresolution subdivision surface with details; middle – optimization without detail preservation; right – optimization with detail preservation.

An added advantage of subdivision hierarchies is that they facilitate the use of multigrid methods [Briggs 1987] 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 [Boier-Martin et al. 2004] 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* [Terzopoulos and Qin 1994; Qin and Terzopoulos 1996]. Dynamic models based on subdivision surfaces have been proposed by Qin et al. [Qin et al. 1998]. 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 [Gonzalez-Ochoa and Peters 1999]. The *Localized hierarchy Surface Splines* allow adding handles and punching holes, while maintaining C^1 continuity across the surface which is represented explicitly in piecewise 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. [Guskov et al. 2002] 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 [Guskov et al. 2002], 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.

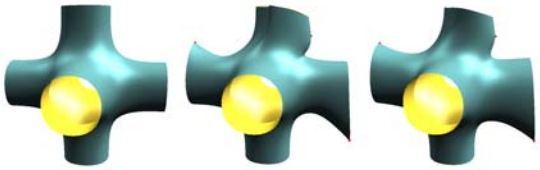


Figure 10: Computing a solution to the energy minimization problem with different accuracies: left - input model; middle - Catmull-Clark solution obtained interactively; right - multigrid solution.

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 [Linsen 2000] 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 [Lanquetin et al. 2003] for variations on the topic of computing intersection curves for subdivision surfaces).

Using a similar control-mesh based approach, Biermann et al. [Biermann et al. 2001] 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 [Loop 1987] 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 accelerate computations and a perturbation scheme [Seidel 1998] 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 α 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

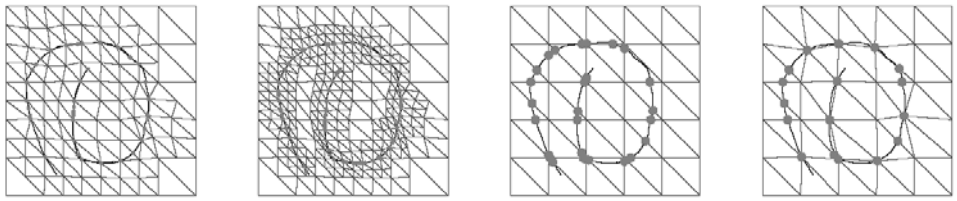


Figure 11: Refinement and snapping: two steps of refinement are shown on the left. The image of the curve in parameter space and vertex snapping are shown on the right (see [Biermann et al. 2001]).

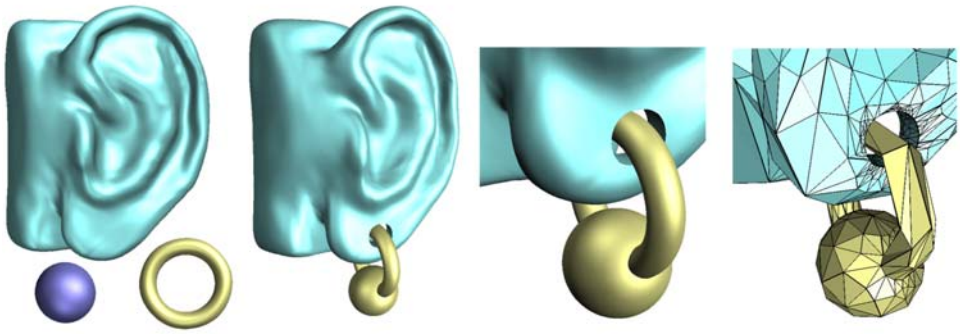


Figure 12: Boolean operations on multireolution subdivision surfaces [Biermann et al. 2001].

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 [Barghiel et al. 1994; Chan et al. 1997]. 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 [Barghiel et al. 1994] and applying a mapping that takes into account the topology of the target surface and the Greville displacement representation of the feature [Barghiel et al. 1994]. 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 [Conrad and Mann 2000] and makes use of quasi-interpolation [deBoor and Fix 1973] 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 [Ma 2000]. 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. [Biermann et al. 2002a] describe a more general procedure for cutting and pasting portions of existing surfaces using an intuitive approach, similar to those commonly used 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 [Biermann et al. 2002a], 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 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 [Biermann et al. 2002a] is to build the mapping 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. In-

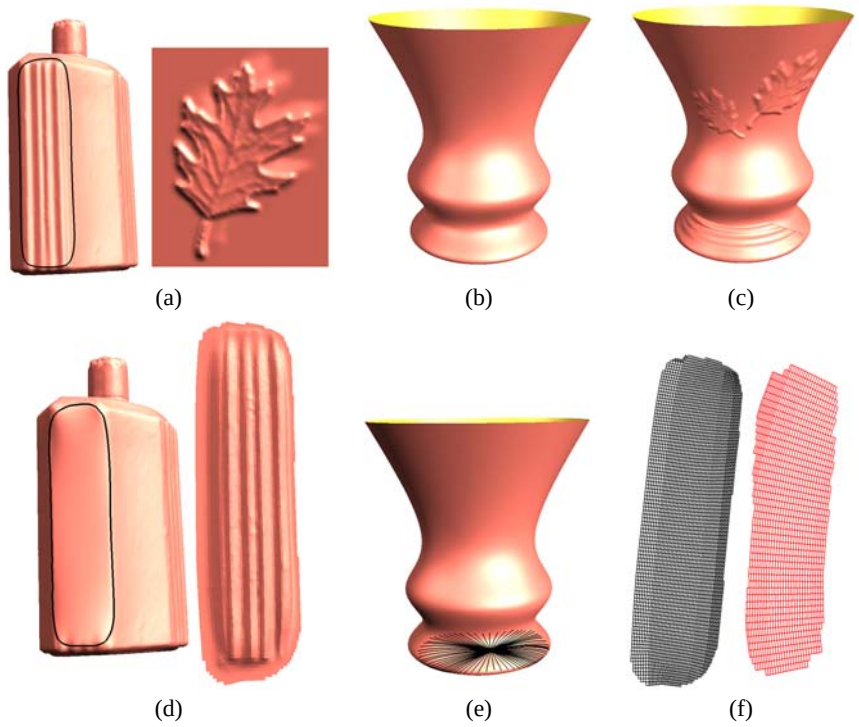


Figure 13: Feature-based design of an ornate vase: (a) input (source) surfaces; (b) target surface; (c) result after multiple pasting operations; Steps of a cut-and-paste sequence according to [Biermann et al. 2002a] shown for the bottle pattern: (d) source feature selection; (e) finding a target region 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).

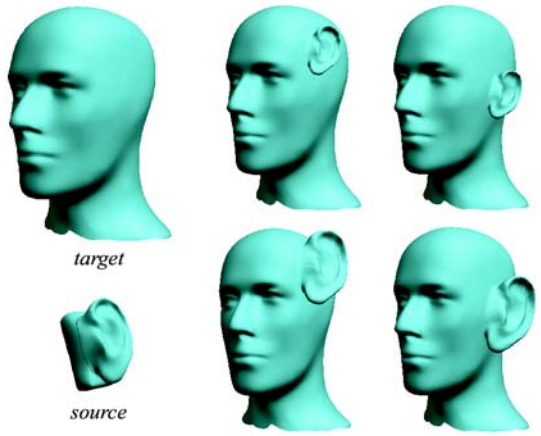


Figure 14: Pasting and interactively placing a complex feature: a digitized model of a clay ear constitutes the feature to be pasted onto the mannequin head. The ear can be interactively scaled, rotated, and translated on the surface of the head.

stead, flattening methods which have been much more extensively researched [Sheffer and de Sturler 2001; Desbrun et al. 2002] 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 [Biermann et al. 2002a] (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.

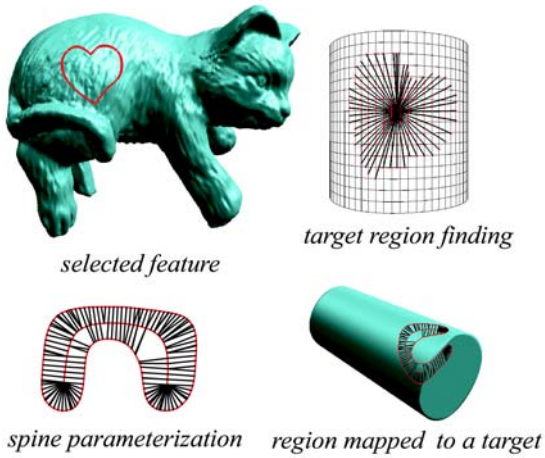


Figure 16: Finding a target region through radial parameterization of the feature outline

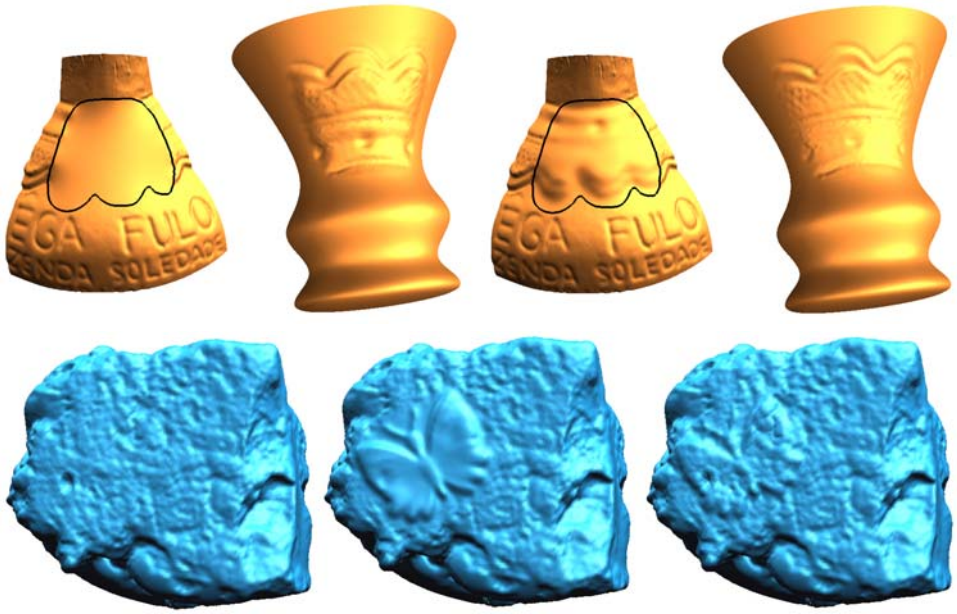


Figure 15: The effects of changing the base surface on the result of pasting: (top) digitized bottle detail appears on the vase differently, depending on the choice of base surface; (bottom) the butterfly feature is pasted on a rock model with and without preservation of target detail.

The pasting method of Biermann et al. [Biermann et al. 2002a] 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 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 [Furukawa et al. 2003]. 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

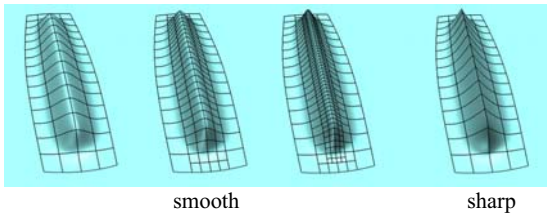


Figure 17: Multiresolution details are required to approximate sharp features using smooth surface representations (left three images). A piecewise smooth surface representation (right) allows sharp features to be modeled with detail vectors.

Early mention of special rules for surfaces with boundaries appeared in the work of Doo [Doo 1978] and Nasri [Nasri 1991],

however accompanied only by partial analyses of the resulting surfaces. The first rules leading to provably C^1-continuous surfaces were defined in [Hoppe et al. 1994] as a generalization of the Loop subdivision rules [Loop 1987]. The analysis of the resulting surfaces can be found in [Schweitzer 1996]. As pointed out in [Biermann et al. 2000], the rules introduced in [Hoppe et al. 1994] 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. [Biermann et al. 2000] for the Loop [Loop 1987] and Catmull-Clark [Catmull and Clark 1978] subdivision schemes.

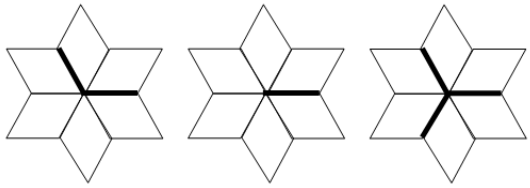


Figure 18: Mesh tags corresponding to (from left to right): crease, dart, and corner sharp features.

Modifications are sometimes applied to subdivision rules to achieve different effects. For example, deRose et al. [DeRose et al. 1998] 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. [Biermann et al. 2000] 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. [Biermann et al. 2000] are illustrated in Figure 19. A software library for piecewise smooth subdivision based on these rules is freely available from [Biermann and Zorin 1999].

A generalization of the subdivision concept that accommodates sharp features was developed by Sederberg et al. [Sederberg et al. 1998]. By drawing an analogy between recursive subdivision schemes and knot insertion for B-splines, the authors propose non-uniform versions of the Doo-Sabin [Doo 1978] and Catmull-Clark [Catmull and Clark 1978] 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. [Khodakovsky and Schröder 1999] 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.

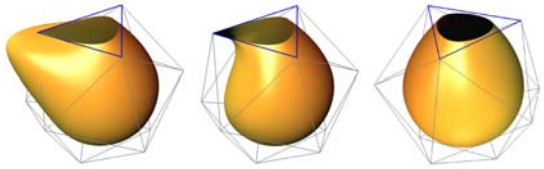


Figure 19: Sharp features generated with the rules proposed in [Biermann et al. 2000]. From left to right: concave corner, convex corner, and smooth crease.

This problem is solved in [Biermann et al. 2002b] 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 $\{v_0, v_1, \ldots\}$ and curve parameters $\{t_0, t_1, \ldots\}$. After a finite number of iterations of snapping and refinement, the resulting curve $[v_0, v_1, \ldots]$ 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.

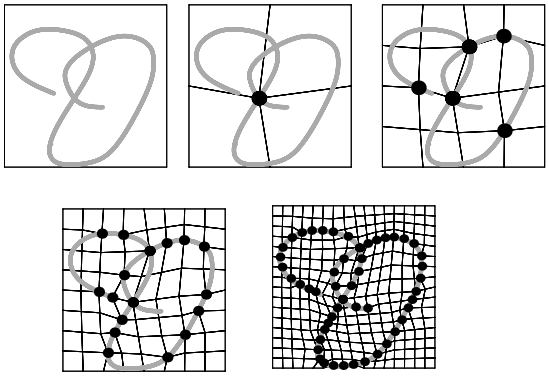


Figure 20: Reparameterization for approximating a feature curve: quads in parameter domain are recursively split and vertices are snapped to the curve. After several subdivision steps the curve is approximated by a sequence of vertices and follows along quad edges or diagonals.

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. [Biermann et al. 2002b] 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 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.

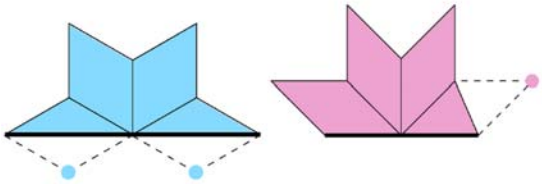


Figure 21: Standard sharp rules do not cover cases when the sharp edge (thick line) passes through a quad diagonal. New rules are necessary for such cases. Dotted lines and circles indicate vertices obtained by reflection used to define subdivision rules for such cases (see [Biermann et al. 2002b] for details).

For completeness, we also mention the trimming method proposed by Litke et al. [Litke et al. 2001b] which is complementary to that of Biermann et al. [Biermann et al. 2002b]. In this case quasi-interpolation is used to approximate a trimmed surface with a combined subdivision surface [Levin 1999].

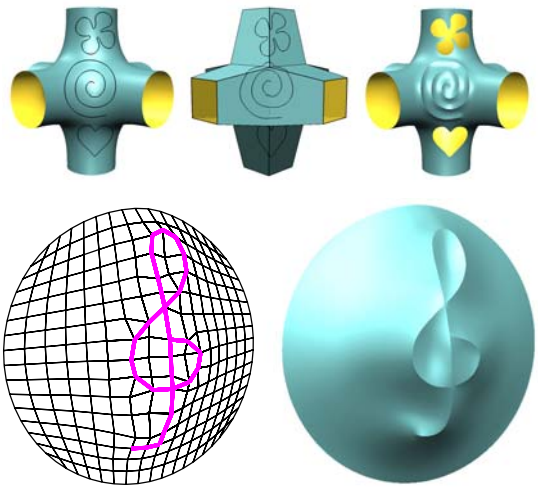


Figure 22: Surfaces obtained after trimming and embossing with sharp features using the method described in [Biermann et al. 2002b]. Top: input curves are shown on the surface (left) and projected into parameter space (middle). The surface obtained is shown on the right. Bottom: a self-intersecting feature.

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 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

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

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 [Peng et al. 2004].

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 [Kajiya and Kay 1989]. Our work was motivated by the work of F. Neyret and co-workers (e.g. [Neyret 1995; Neyret 1998; Meyer and Neyret 1998]) as well as recent work on fur rendering [Lengyel 2000; Lengyel et al. 2001].

Our geometry is to some extent similar to the slab representation used for modeling weathered stone ([Dorsey et al. 1999]) and for volume sculpting in [Agarwala 1999]. ([Dorsey et al. 1999]) uses the fast marching method (e.g. [Sethian 1999]) to construct layers around a surface. Envelope construction [Cohen et al. 1996] 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. [Pizer et al. 1994; Eberly et al. 1994]). 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 [Siddiqi et al. 1999] which solves the Hamilton-Jacobi equations for the medialness function on a regular grid to recover a skeleton. [Yezzi and Prince 2002] 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 ([Ricci 1973; Rockwood 1987; Pasko et al. 1995]), 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 [Bloomenthal 1997] for a list of references); some recent important work includes [Frisken et al. 2000; Carr et al. 2001]. Interactive and procedural volume sculpting techniques [Wang and Kaufman 1995; Agarwala 1999; Cutler et al. 2002] 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 [Henshaw 1996; Bern and Plassmann 2000] and the book [Steinberg and Knupp 1993]). 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 [Henshaw 1996].

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.

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 α. 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.

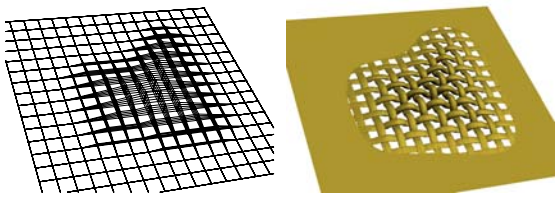


Figure 24: Surface with a volume layer attached.

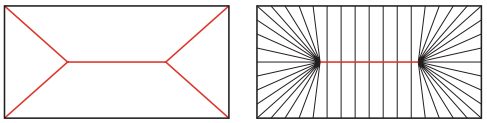


Figure 25: a. Medial axis of a box. b. The shell with target thickness exceeding one half of the box size constructed using the gradient along the medial axis. The shell director lines are shown.

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 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

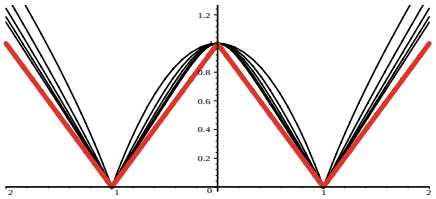


Figure 26: The red plot shows the standard distance function from a point on the line to the set of two points $\{-1,1\}$. Other lines show the averaged distance functions for different values of p.

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

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

This definition lends itself to a natural generalization:

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

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 ∞. 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[1]. 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

[1] 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.

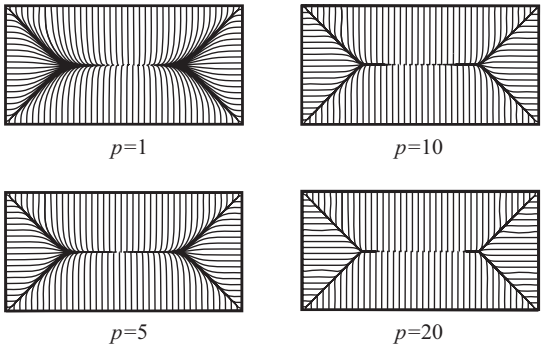


Figure 27: Field lines of the gradient field of the distance function for several values of p.

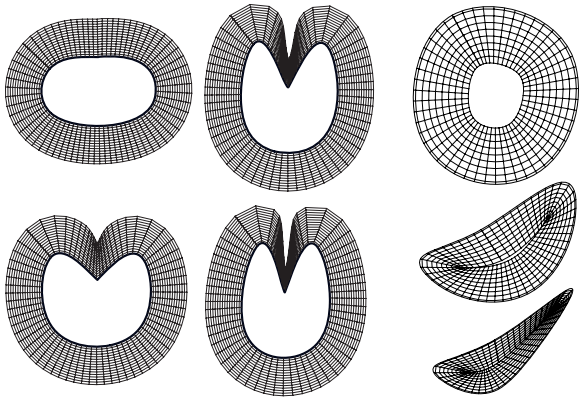


Figure 28: The four diagrams on the left show self-adjusting shell behavior of an exterior shell in the concave region. With an angle of up to 90 degrees, no compression in shell thickness is observed, but at greater angles the shell starts to compress. The three diagrams in the right column illustrate 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).

be expanded (see Figure 28 for the results of our two-dimensional experiments on deforming curves).

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 Δ 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 ε 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 ε;
  $\mathbf{x} = \mathbf{x}_{new}$; $\quad t += \Delta$;
  $g = g_{new}$;
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

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

where (u,v,w) is the coordinates of the point $\mathbf{x}$, and β is the counterclockwise 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.

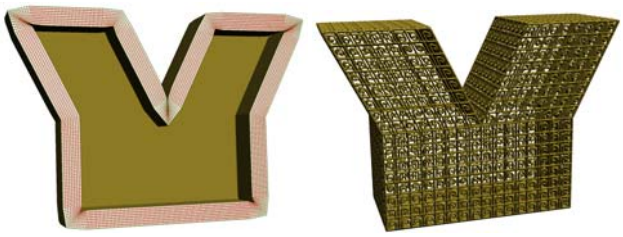


Figure 29: Left: cross-section of the shell for a shape with sharp corners; Right: same object with volume texture added.

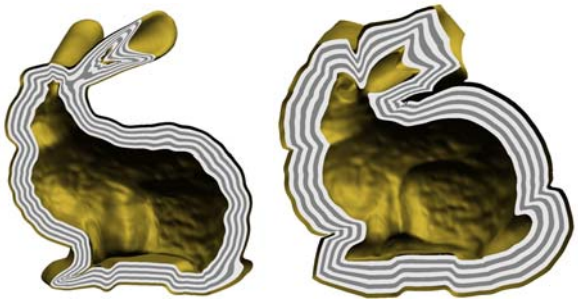


Figure 30: Cross-sections of interior and exterior shells of the bunny.

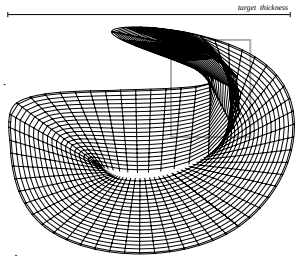


Figure 31: Folding for extreme shell thickness (prescribed thickness equal to the objects bounding box size, only 70% of the shell shown to show the fold clearly.)

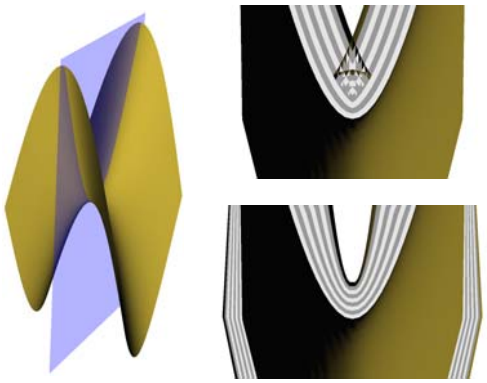


Figure 32: Comparison of the results of normal displacement method (upper right) and our method (lower right) for a saddle.

Although we only integrate over a small part of the surface, inside a ball near a given point, further acceleration helps. We use the Barnes and Hut algorithm [Barnes and Hut 1986] 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 [Sethian 1994]. Another alternative is the envelope construction [Cohen et al. 1996] 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 [Yezzi and Prince 2002] 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.

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.

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 [Biermann et al. 2002a]. 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.

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.

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).

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

AGARWALA, A. 1999. *Volumetric Surface Sculpting.* Master's thesis, MIT.

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

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

BARR, A. H. 1984. Global and local deformations of solid primitives. *Proc. of SIGGRAPH 84*, 21–30.

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.

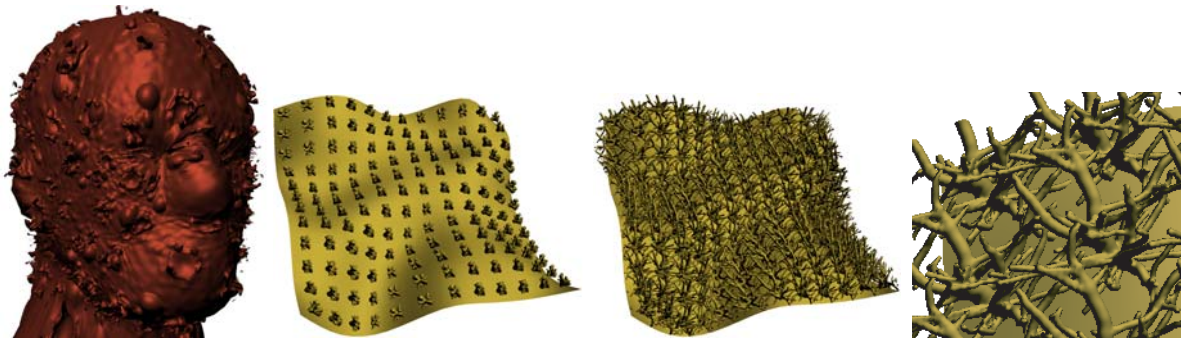


Figure 35: An animated bubbling texture applied on a deforming head and two stages of a growing bush texture with a zoomed-in view.

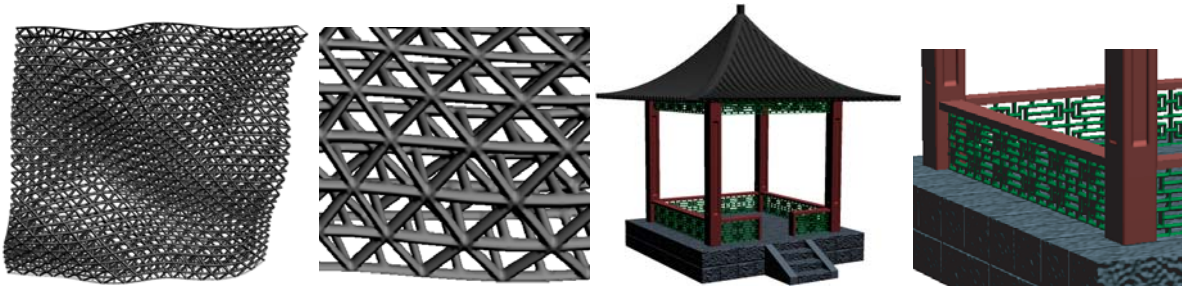


Figure 36: A structural texture on a deforming plane and a kiosk. The second and the forth pictures are showing zoomed-in details. Volume textures are used on the kiosk roof and walls.

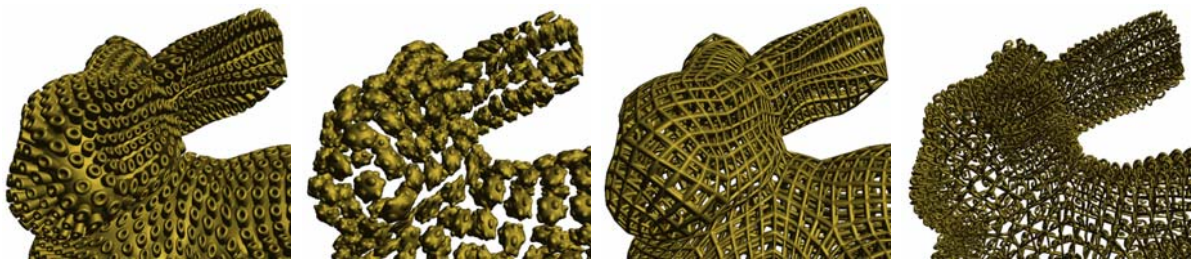


Figure 37: Several different volumetric textures applied on the bunny. Only the heads are shown to view the geometric details.

BERN, M., AND PLASSMANN, P. 2000. Mesh generation. In *Handbook of computational geometry*. North-Holland, Amsterdam, 291–332.

BÉZIER, P. 1974. Mathematical and practical possibilities of UNISURF. *CAD*, 127–152.

BIERMANN, H., AND ZORIN, D., 1999. Subdivide 2.0 software.

BIERMANN, H., LEVIN, A., AND ZORIN, D. 2000. Piecewise smooth subdivision surfaces with normal control. In *Proceedings of SIGGRAPH 00*, 113–120.

BIERMANN, H., KRISTJANSSON, D., AND ZORIN, D. 2001. Approximate boolean operations on free-form solids. In *Proceedings of SIGGRAPH 01*, 185–194.

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

BIERMANN, H., MARTIN, I., ZORIN, D., AND BERNARDINI, F. 2002. Sharp features on multiresolution subdivision surfaces. *Graphical Models 64*, 2, 61–77.

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

BLOOMENTHAL, J., Ed. 1997. *Introduction to implicit surfaces*. Morgan Kaufmann.

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

BOO, M., AMOR, M., DOGGETT, M., HIRCHE, J., AND STRASSER, W. 2001. Hardware support for adaptive subdivision surface rendering. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*.

BRIGGS, W. L. 1987. *Multigrid Tutorial*. SIAM.

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

In *http://www.catia.ibm.com*.

CATMULL, E., AND CLARK, J. 1978. Recursively generated B-spline surfaces on arbitrary topological meshes. *CAD 10*, 6, 350–355.

CELNIKER, G., AND GOSSARD, D. 1999. Energy-based models for free-form surface shape design. In *Proc. ASME Design Automation Conference*.

CHAN, L. K. Y., MANN, S., AND BARTELS, R. 1997. World space surface pasting. In *Proceedings of Graphics Interface*, W. Davis, M. Mantei, and V. Klassen, Eds., 146–154.

CHANG, Y., AND ROCKWOOD, A. P. 1994. A generalized de casteljau approach to 3D Free-Form deformation. *Proc. of SIGGRAPH 94*, 257–260.

CHANG, M. M., SEZAN, M. I., AND TEKALP, A. M. 1994. Adaptive bayesian segmentation of color images. *Journal of Electronic Imaging 3*, 404–414.

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

COHEN, J., VARSHNEY, A., MANOCHA, D., TURK, G., WEBER, H., AGARWAL, P., JR., F. P. B., AND WRIGHT, W. 1996. Simplification envelopes. In *Proceedings of SIGGRAPH 96*, 119–128.

COHEN, E., RIESENFELD, R. F., AND ELBER, G. 2001. *Geometric Modeling with Splines*. A K Peters Ltd.

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

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

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

DEBOOR, C., AND FIX, G. J. 1973. Spline approximtion by quasi-interpolants. *Journal of Approximation Theory 8*, 19–45.

DEROSE, T., KASS, M., AND TRUONG, T. 1998. Subdivision surfaces in character animation. In *Proceedings of SIGGRAPH 98*, 85–94.

DESBRUN, M., MEYER, M., AND ALLIEZ, P. 2002. Intrinsic parameterizations of surface meshes. In *Eurographics conference proceedings*, 209–218.

DIEWALD, U., MORIGI, S., AND RUMPF, M. 2002. A cascadic geometric filtering approach to subdivision. *CAGD 19*, 9, 675 – 694.

DOO, D., AND SABIN, M. 1978. Analysis of the behaviour of recursive division surfaces near extraordinary points. *CAD 10*, 6, 356–360.

DOO, D. 1978. A subdivision algorithm for smoothing down irregularly shaped polyhedrons. In *Proceedings on Interactive Techniques in Computer Aided Design*, 157–165.

DORSEY, J., EDELMAN, A., LEGAKIS, J., JENSEN, H. W., AND PEDERSEN, H. K. 1999. Modeling and rendering of weathered stone. In *Proceedings of ACM SIGGRAPH 99*, 225–234.

In *http://www.discreet.com*.

DYN, N., AND LEVIN, D. 2002. Subdivision schemes in geometric modelling. *Acta Numerica 11*.

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

EBERLY, D., GARDNER, R., MORSE, B., PIZER, S., AND SCHARLACH, C. 1994. Ridges for image analysis. *Journal of Mathematical Imaging and Vision 4*, 351–371.

FRIEDEL, I., MULLEN, P., AND SCHRODER, P. 2003. Data-dependent fairing of subdivision surfacesn. In *Proc. of SM 03*.

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

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

GIBSON, S. F. F., AND MIRTICH, B. 1997. A survey of deformable modeling in computer graphics. Tech. Rep. TR-97-19, MERL, Cambridge, MA.

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

GREINER, G. 1994. Surface construction based on variational principles. In *Wavelets, Images and Surface Fitting*, P. J. Laurent, A. LeMéhauté, and L. Schumaker, Eds. AK Peters, 277–286.

GRINSPUN, E., AND SCHRÖDER, P. 2001. Normal bounds for subdivision-surface interference detection. In *Proc. of IEEE Visualization 01*.

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

HALSTEAD, M., KASS, M., AND DEROSE, T. 1993. Efficient, fair interpolation using Catmull-Clark surfaces. In *Proc. SIGGRAPH 1993*, 35–44.

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

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

HOPPE, H., DEROSE, T., DUCHAMP, T., HALSTEAD, M., JIN, H., MCDONALD, J., SCHWEITZER, J., AND STUETZLE, W. 1994. Piecewise smooth surface reconstruction. In *Computer Graphics Proceedings*, Annual Conference Series, ACM Siggraph, 295–302.

HUA, J., AND QIN, H. 2003. Free-Form deformations via sketching and manipulating scalar fields. In *Proc. ACM Symp. on Solid Modeling and Applications*, 328 – 333.

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

KHODAKOVSKY, A., AND SCHRÖDER, P. 1999. Fine level feature editing for subdivision surfaces. In *Proceedings of ACM Solid Modeling*.

KOBBELT, L. 1996. Interpolatory subdivision on openquadrilateral nets with arbitrary topology. In *Proc. of Eurographics 96*, 409–420.

KOBBELT, L. 1996. A variational approach to subdivision. *Comput. Aided Geom. Design 13*, 8, 743–761.

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

LANQUETIN, S., FOUFOU, S., KHEDDOUCI, H., AND NEVEU, M. 2003. Computing subdivision surface intersection. In *Proc. WSCG '03*.

LAZARUS, F., COQUILLART, S., AND JANCENE, P. 1994. Axial deformations: an intuitive deformation technique. *CAD 26*, 8, 607–61.

LEE, A., MORETON, H., AND HOPPE, H. 2000. Displaced subdivision surfaces. In *Proc. of SIGGRAPH 00*, 85–94.

LENGYEL, J. E., PRAUN, E., FINKELSTEIN, A., AND HOPPE, H. 2001. Real-time fur over arbitrary surfaces. In *2001 ACM Symposium on Interactive 3D Graphics*, 227–232.

LENGYEL, J. 2000. Real-time hair. In *Rendering Techniques 2000: 11th Eurographics Workshop on Rendering*, Eurographics, 243–256.

LEVIN, A. 1999. Interpolating nets of curves by smooth subdivision surfaces. *Proc. of SIGGRAPH 99*, 57–64.

LINSEN, L. 2000. Netbased modelling. In *Proc. SCCG '00*, 259–266.

LITKE, N., LEVIN, A., AND SCHRÖDER, P. 2001. Fitting subdivision surfaces. In *Proc. of IEEE Visualization 2001*, 319–324.

LITKE, N., LEVIN, A., AND SCHRODER, P. 2001. Trimming for subdivision surfaces. *Computer Aided Geometric Design 18*, 5, 463–481.

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

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

MA, W., AND ZHAO, N. 2000. Catmull-clark surface fitting for reverse engineering applications. In *Proceedings of Geometric Modeling and Processing*, 274–282.

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

MACCRACKEN, R., AND JOY, K. I. 1996. Free-Form deformations with lattices of arbitrary topology. In *Proc. of SIGGRAPH 96*, 181–188.

In *http://www.alias.com*.

MCDONNELL, K., AND QIN, H. 2000. Dynamic sculpting and animation of Free-Form subdivision solids. In *Proc. of IEEE Computer Animation*.

MEYER, A., AND NEYRET, F. 1998. Interactive volumetric textures. In *Eurographics Rendering Workshop 1998*, Springer Wein, New York City, NY, G. Drettakis and N. Max, Eds., Eurographics, 157–168.

NASRI, A. H., AND ABBAS, A. 2002. Designing Catmull-Clark subdivision surfaces with curve interpolation constraints. *Computers and Graphics*.

NASRI, A. H. 1991. Surface interpolation on irregular networks with normal conditions. *CAGD 8*, 89–96.

NASRI, A. 2000. Interpolating meshes of boundary intersecting curves by subdivision surfaces. *The Visual Computer 16*.

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

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

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

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

PETERS, J., AND REIF, U. 1997. The simplest subdivision scheme for smoothing polyhedra. *ACM TOG 16*, 4.

PETERS, J. 2000. Patching Catmull-Clark meshes. In *Proceedings of SIGGRAPH 00*, 255–258.

PIPONI, D., AND BORSHUKOV, G. 2000. Seamless texture mapping of subdivision surfaces by model pelting and texture blending. In *Proceedings of SIGGRAPH 00*, 471–478.

PIZER, S., BURBECK, C., COGGINS, J., FRITSCH, D., AND MORSE, B. 1994. Object shape before boundary shape: Scale-space medial axes. *Journal of Mathematical Imaging and Vision 4*, 303–313.

PULLI, K., AND LOUNSBERY, M. 1997. Hierarchical editing and rendering of subdivision surfaces. Tech. Rep. UW-CSE-97-04-07, Dept. of CS&E, University of Washington, Seattle, WA.

PULLI, K., AND SEGAL, M. 1996. Fast rendering of subdivision surfaces. In *Proc. Eurographics Workshop on Rendering*, 61–70.

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

QIN, H., MANDAL, C., AND VEMURI, B. 1998. Dynamic Catmull-Clark subdivision surfaces. *IEEE TVCG 4*, 3, 215–229.

RICCI, A. 1973. A constructive geometry for computer graphics. *The Computer Journal 16*, 2, 157–160.

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

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

SABIN, M. A. 2002. Subdivision surfaces tutorial. *SMI 02*.

SCHAEFFER, S., WARREN, J., AND ZORIN, D. 2004. Lofting curve networks using subdivision surfaces. *submitted*.

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

SEDERBERG, T. W., AND PARRY, S. R. 1986. Free-form deformation of solid geometric models. In *Proc. of SIGGRAPH 86*, 151–160.

SEDERBERG, T. W., ZHENG, J., SEWELL, D., AND SABIN, M. 1998. Non-uniform recursive subdivision surfaces. In *Proceedings of SIGGRAPH 98*, 387–394.

SEIDEL, R. 1998. The nature and meaning of perturbations in geometric computing. *Discrete Computational Geometry 19*, 1, 1–17.

SETHIAN, J. A. 1994. Curvature flow and entropy conditions applied to grid generation. *J. Comput. Phys. 115*, 2, 440–454.

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

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

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

SINGH, K., AND FIUME, E. 1998. Wires: A geometric deformation technique. In *Proc. of SIGGRAPH 98*, 405–414.

STAM, J. 1998. Exact evaluation of catmull-clark subdivision surfaces at arbitrary parameter values. In *Proceedings of SIGGRAPH 98*, 395–404.

STEINBERG, S., AND KNUPP, P. M. 1993. *Fundamentals of Grid Generation*. CRC Press.

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

SUZUKI, H., TAKEUCHI, S., KIMURA, F., AND KANAI, T. 1999. Subdivision surface fitting to a range of points. In *Proc. Pacific Graphics*.

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

TERZOPOULOS, D., AND FLEISCHER, K. 1988. Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. In *Proc. of SIGGRAPH '88*, 269–278.

TERZOPOULOS, D., AND QIN, H. 1994. Dynamic NURBS with geometric constraints for interactive sculpting. *ACM TOG 13*, 2, 103–136.

TERZOPOULOS, D., PLATT, J., BARR, A., AND FLEISCHER, K. 1987. Elastically deformable models. In *Proc. of SIGGRAPH 87*, 205–214.

TURK, G., AND O'BRIEN, J. 2002. Modelling with implicit surfaces that interpolate. *ACM TOG 21*, 4, 855 – 873.

WANG, S. W., AND KAUFMAN, A. E. 1995. Volume sculpting. In *Proceedings of the 1995 symposium on Interactive 3D graphics*, ACM Press, 151–156.

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

WEIMER, H., AND WARREN, J. 1998. Subdivision schemes for thin-plate splines. *Proc. EUROGRAPHICS '98 17*, 3.

WELCH, W., AND WITKIN, A. 1992. Variational surface modeling. In *Proceedings of SIGGRAPH '92*, vol. 26, 157–166.

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

ZORIN, D., AND KRISTJANSSON, D. 2002. Evaluation of piecewise smooth subdivision surfaces. *The Visual Computer 18*, 5–6, 299 – 315.

ZORIN, D., SCHRÖDER, P., AND SWELDENS, W. 1996. Interpolating subdivision for meshes with arbitrary topology. In *Proc. of SIGGRAPH 96*, 189–192.

ZORIN, D., SCHRÖDER, P., AND SWELDENS, W. 1997. Interactive multiresolution mesh editing. In *Proc. of SIGGRAPH 97*, 259–268.

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

Mesh Editing based on Discrete Laplace and Poisson Models

Marc Alexa
Faculty of EE & CS
TU Berlin

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.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—curve, 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 [Sederberg and Parry 1986; Coquillart 1990], cut and paste [Ranta et al. 1993; Biermann et al. 2002], fusion [Kanai et al. 1999], morphing [Alexa 2003a], 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 [Forsey and Bartels 1988; Zorin et al. 1997; Kobbelt et al. 1998; Kobbelt et al. 1999; Guskov et al. 1999]. 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 invalidated 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 [Alexa 2003b; Botsch and Kobbelt 2004; Sorkine et al. 2004; Yu et al. 2004]. 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 [Fattal et al. 2002; Pérez et al. 2003]. 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 $\mathscr{M}$ 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 $\mathscr{N}_i = \{j|(i,j) \in K\}$ and the *degree* d_i of this vertex is the number adjacent edges (or vertices), i.e. the number of elements in $\mathscr{N}_i$. 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

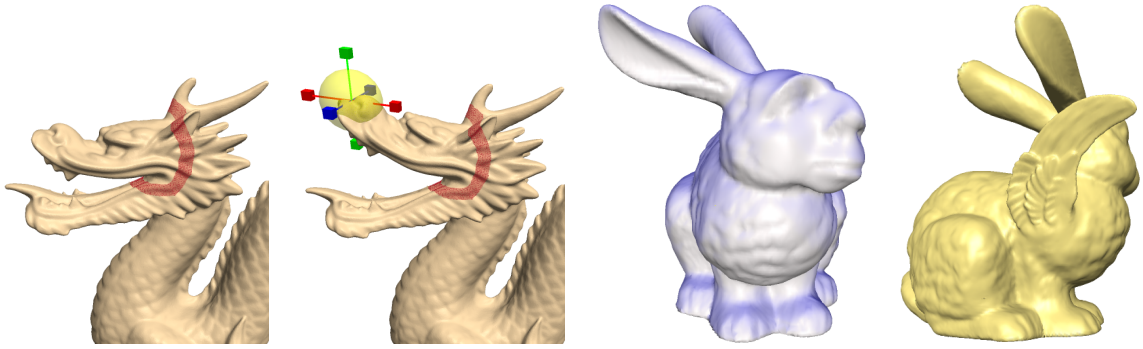
$$\delta_i = \mathbf{v}_i - c_{ij} \sum_{j \in \mathscr{N}_i} \mathbf{v}_j, \quad \sum_j c_{ij} = 1, \tag{1}$$

and differ in the definition of the coefficients c_{ij}. The topological Laplacian ([Taubin 1995]) 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. [Meyer et al. 2003]) perform best in most applications.

The transformation between V and Δ 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



(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.

$n-1$, which means V can be recovered from Δ 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 [Alexa 2003b; Lipman et al. 2004]. 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 [Botsch and Kobbelt 2004] 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. [Yu et al. 2004].

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

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

to the system matrix A.

Furthermore, also differentials could be prescribed. Note that δ_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 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 [Yu et al. 2004].
- They could be deduced from the membrane solution (i.e. $LV' = 0$) [Lipman et al. 2004].
- They could be implicitly defined by the solution, if the rotational part is linearized [Sorkine et al. 2004].

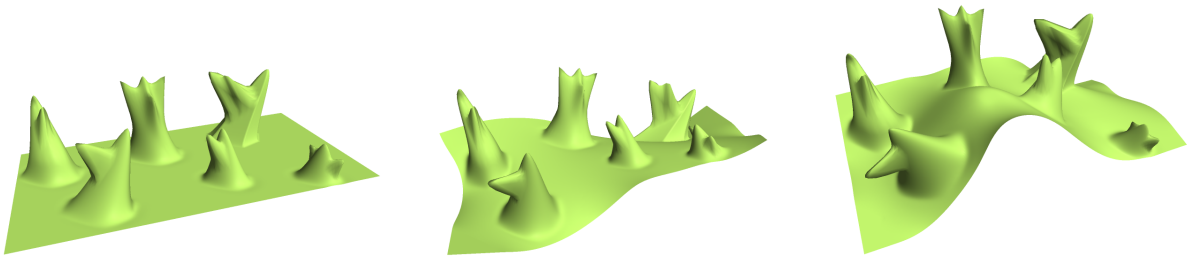
In any case, we first need to extend the definition of the local intrinsic representation to incorporate linear transformations.

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 δ_i with T_i. This results in a slightly modified functional defining the resulting geometry V':

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

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



(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.

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. [Yu et al. 2004] 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 [Yu et al. 2004]) 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. [Yu et al. 2004] 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. [Lipman et al. 2004] 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.

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:

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

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.

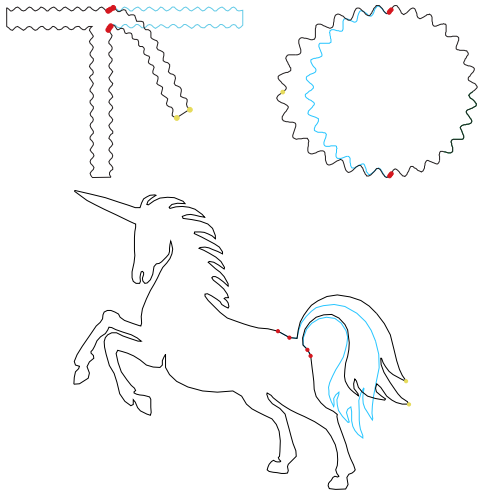


Figure 3: Editing 2D meshes using Laplacian-coordinates fitting. The red dots denote fixed anchor points and the yellow are the pulled handle vertices. The original meshes are colored blue.

4.3 Linearized implicit transformations

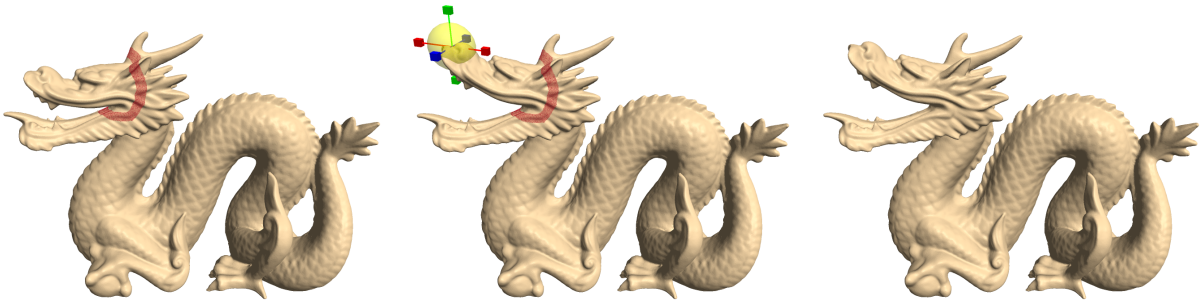
The main idea of [Sorkine et al. 2004] is to compute an appropriate transformation T_i for each vertex i based on the eventual new configuration of vertices V'. Thus, $T_i(V')$ is a function of V'.

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.

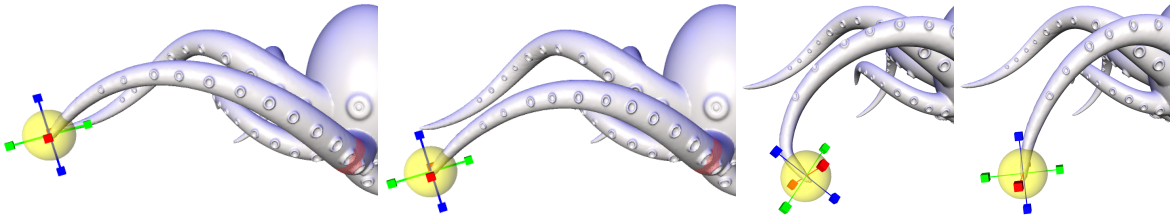
The transformation should be a linear function in the target configuration but constrained to isotropic scales and rotations. The class of matrices representing isotropic scales and rotation can be written as $T = s \exp(H)$, where H is a skew-symmetric matrix. In 3D, skew symmetric matrices emulate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

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



(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.

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[1]. As a linear approximation of the class of constrained transformations 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

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

[1]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

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

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.

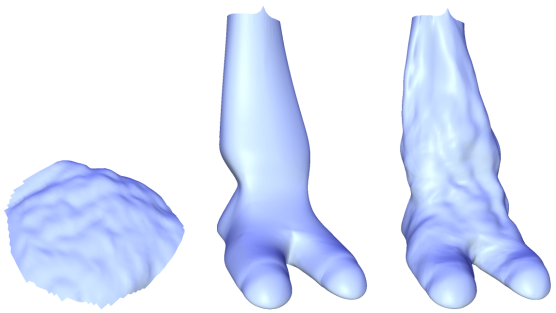
4.4 Adjusting T_i

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 δ_i is updated appropriately 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.

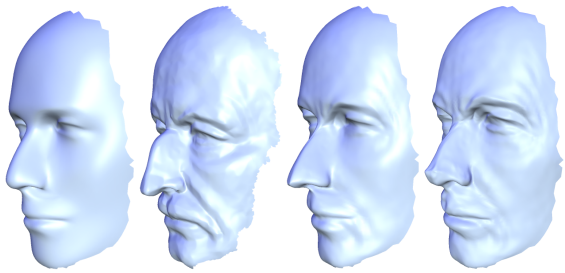
5 Mesh Editing

There are many different tools to manipulate an existing mesh. Perhaps the simplest form consists of manipulating a *handle*, which is a



(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).

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., [Bendels and Klein 2003] 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 [Desbrun et al. 1999; Fleishman et al. 2003]. 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 δ_i and $\tilde{\delta}_i$ be the Laplacian coordinates of the vertex i in S and $\tilde{S}$, respectively. We define ξ_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 ξ_j encode the details of S, since given the bare surface $\tilde{S}$ we can recover the original details simply by adding ξ_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 ξ, we would like to add the details ξ 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 ξ 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.

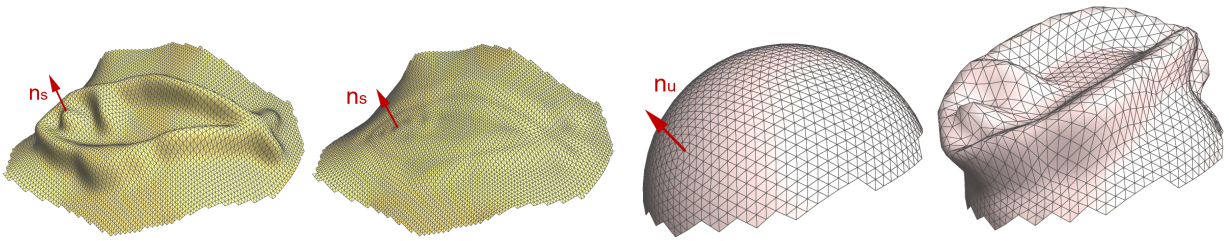
The following is an important property of the Laplacian coordinates:

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

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



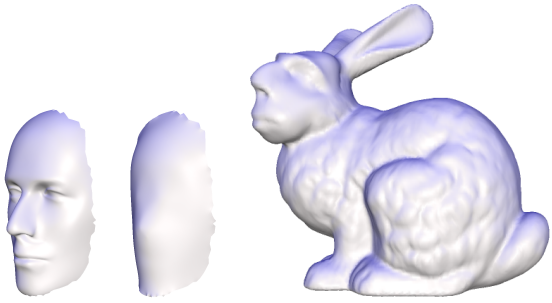
(a) (b) (c) (d)

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).

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 ξ_i, the detail transfer from S onto U is expressed as follows:

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

where Δ denotes the Laplacian coordinates of the vertices of U. Now the new surface U' has the details of U.



(a) (b) (c)

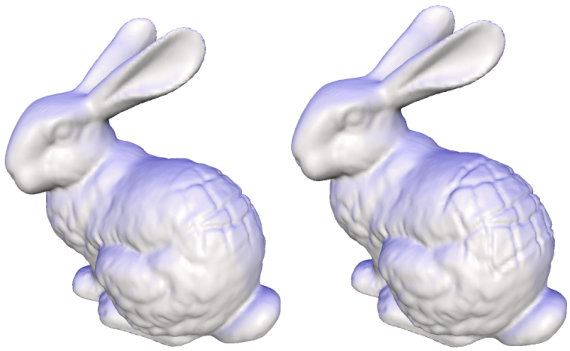
Figure 9: Transferring the details of the *Mannequin* onto the face of the *Bunny*. (a) The source surface S. It is significantly smoothed to peel the details. (b) The smoothed surface $\tilde{S}$. (c) The result of detail transfer onto the *Bunny*.

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 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 [Floater 2003], 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



(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).

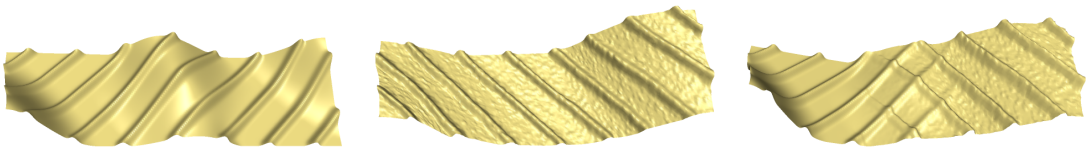
relevant details like facial features such as the brow wrinkles of the *Max Planck*. In this case the user provides a few additional (inner) point-to-point constraints which define a warp of the mean-value parameterization. In our implementation we use a radial basis function elastic warp, but any reasonable 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 τ. 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 ξ_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



(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).

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 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.

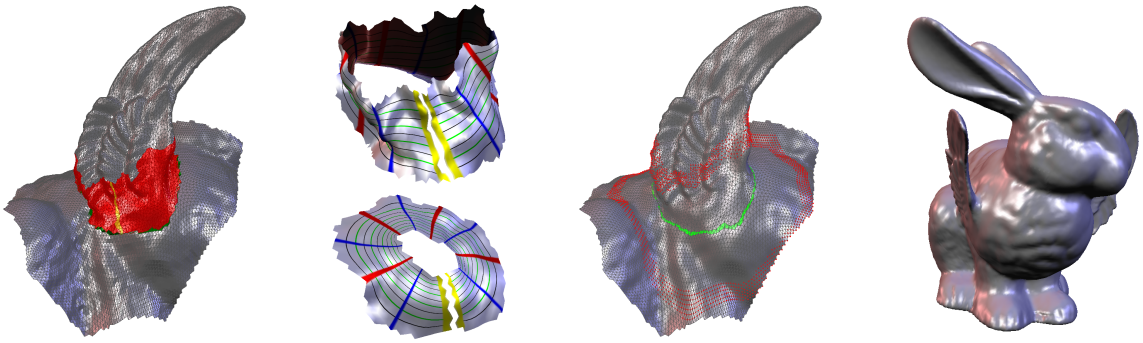
8 Implementation details

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.



(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)).

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

ALEXA, M. 2003. Differential coordinates for local mesh morphing and deformation. *The Visual Computer 19*, 2, 105–114.

ALEXA, M. 2003. Differential coordinates for local mesh morphing and deformation. *The Visual Computer 19*, 2, 105–114.

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

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

BOTSCH, M., AND KOBBELT, L. 2004. An intuitive framework for real-time freeform modeling. *ACM Trans. Graph. 23*, 3, 630–634.

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

DESBRUN, M., MEYER, M., SCHRÖDER, P., AND BARR, A. H. 1999. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of ACM SIGGRAPH 99*, 317–324.

FATTAL, R., LISCHINSKI, D., AND WERMAN, M. 2002. Gradient domain high dynamic range compression. In *Proceedings of ACM SIGGRAPH 2002*, 249–256.

FLEISHMAN, S., DRORI, I., AND COHEN-OR, D. 2003. Bilateral mesh denoising. In *Proceedings of ACM SIGGRAPH 2003*, 950–953.

FLOATER, M. S. 2003. Mean-value coordinates. *Computer Aided Geometric Design 20*, 19–27.

FORSEY, D., AND BARTELS, R. 1988. Hierarchical b-spline refinement. In *Proceedings of ACM SIGGRAPH 88*, 205–212.

GUSKOV, I., SWELDENS, W., AND SCHRÖDER, P. 1999. Multiresolution signal processing for meshes. In *Proceedings of ACM SIGGRAPH 99*, 325–334.

KANAI, T., SUZUKI, H., MITANI, J., AND KIMURA, F. 1999. Interactive mesh fusion based on local 3D metamorphosis. In *Graphics Interface '99*, 148–156.

KOBBELT, L., CAMPAGNA, S., VORSATZ, J., AND SEIDEL, H.-P. 1998. Interactive multi-resolution modeling on arbitrary meshes. In *Proceedings of ACM SIGGRAPH 98*, 105–114.

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

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

MEYER, M., DESBRUN, M., SCHRDER, P., AND BARR, A. H. 2003. Discrete differential-geometry operators for triangulated 2-manifolds. *Visualization and Mathematics III*, pages 35–57.

PÉREZ, P., GANGNET, M., AND BLAKE, A. 2003. Poisson image editing. In *Proceedings of ACM SIGGRAPH 2003*, 313–318.

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

SEDERBERG, T. W., AND PARRY, S. R. 1986. Free-form deformation of solid geometric models. In *Proceedings of SIGGRAPH 86*, 151–160.

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

TAUBIN, G. 1995. A signal processing approach to fair surface design. In *Proceedings of ACM SIGGRAPH 95*, 351–358.

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

ZORIN, D., SCHRDER, P., AND SWELDENS, W. 1997. Interactive multiresolution mesh editing. In *Proceedings of ACM SIGGRAPH 97*, 259–268.

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 α, β, γ.

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 δ_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 [Sorkine et al. 2004] 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 δ_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

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

or explicitly for, e.g., x:

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

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.

Designing with Distance Fields

Sarah F. Frisken
Tufts University
frisken@cs.tufts.edu

Ronald N. Perry
MERL
perry@merl.com

1. Introduction

Distance fields provide an implicit representation of shape that has advantages in many application areas; in this overview, we focus on their use in digital design. Distance fields have been used in Computer Aided Design since the 1970's (e.g., for computing offset surfaces and for generating rounds and filets). More recently, distance fields have been used for freeform design where their dual nature of providing both a volumetric representation and a high-quality surface representation provides a medium that has some of the properties of real clay. Modern computer systems coupled with efficient representations and methods for processing distance fields have made it possible to use distance fields in interactive design systems. This overview reviews previous work in distance fields, discusses the properties and advantages of distance fields that make them suitable for digital design, and describes Adaptively Sampled Distance Fields (ADFs), a distance field representation capable of representing detailed, high quality, and expressive shapes. ADFs are both efficient to process and have a relatively small memory footprint.

2. Distance Fields

An object's distance field specifies, for any point in space, the distance from that point to the boundary of the object. The distance can be signed to distinguish between the inside and outside of an object (see Figure 1a). Distance fields are a specific example of implicit functions, which have a long history of use and study (e.g., see [Bloomenthal 1997]). A distance field can be represented by a scalar function $dist(\mathbf{x})$ which maps $\mathbf{x} \in \Re^n$ onto $\Re$. Typically, the boundary Ω of an object represented by a distance field is located at the zero-valued iso-surface of the distance function, i.e., Ω is the set of all points where $dist(\mathbf{x}) = 0$.

The general form of a distance function is $dist(\mathbf{x}) = Norm(\mathbf{x} - S(\mathbf{x}))$, where $Norm(\mathbf{u})$ is a metric that decreases monotonically with $\|\mathbf{u}\|$ and $S(\mathbf{x})$ is a point on the boundary Ω. A *minimum* distance function is such that $S(\mathbf{x}) = \mathbf{s}^*$, where $\mathbf{s}^*$ is on Ω and $|Norm(\mathbf{s}^*)| \leq |Norm(\mathbf{s})| \ \forall \ \mathbf{s} \in \Omega$. Such general forms of the distance function have uses in various applications (e.g., distance fields with non-vanishing gradients are used in Computer Aided Design and Manufacturing (CAD/CAM) by [Biswas and Shapiro 2004]), but Euclidean distance (i.e., $dist(\mathbf{x}) = \pm\|\mathbf{x} - \mathbf{s}^*\|$) is frequently used because of its utility in a number of applications (e.g., collision detection and surface offsetting).

2.1 Properties of Distance Fields

Distance fields have a number of useful properties. Unlike boundary representations, a distance field representing an object is defined everywhere in space and not just on the object's surface. With a distance field representation, it is trivial to determine whether a point is inside, outside, or on the boundary of the represented shape; the distance function is simply evaluated at a query point and compared to the value of the iso-surface representing the boundary. The gradient of the distance field (i.e., $(\delta dist(\mathbf{x})/\delta x, \ \delta dist(\mathbf{x})/\delta y, \ \delta dist(\mathbf{x})/\delta z)$ in 3D) yields the surface normal if the point $\mathbf{x}$ lies on the boundary Ω and the direction to the closest point on the surface for points off of the boundary Ω.

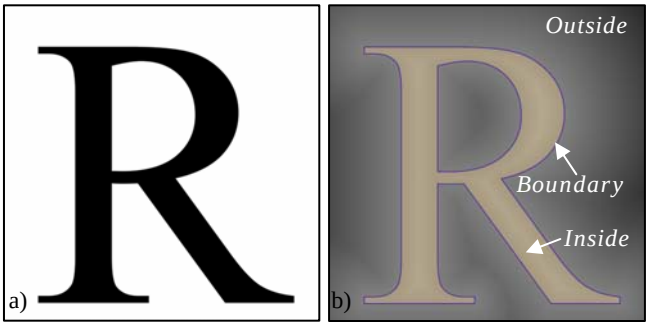


Figure 1. a) A 2D shape and b) its 2D distance field. The shape's distance field represents its boundary, its interior (tinted brown here for illustration) and the space in which it sits.

Euclidean distance fields are C^0 continuous everywhere in space and C^1 continuous except at boundaries of Voronoi regions (see Figure 2). Discontinuities in the gradient occur near sharp corners and along the medial axis of the shape and can be avoided 1) near the boundary Ω by filtering the boundary representation to avoid sharp corners (e.g., see [Sramek and Kaufman, 1999]) or 2) throughout the field by using alternatives to the Euclidean distance function (e.g., see [Biswas and Shapiro 2004]).

2.2 Operations on Distance Fields

Distance fields are particularly useful in design because they make it fast and simple to combine preexisting shapes using Boolean operations such as unioning, differencing, and intersection (see Figure 3). Such Boolean operations are used in Constructive Solid Geometry (CSG) to combine primitive solids such as spheres, cylinders, and rectangular boxes to form complex shapes. Boolean operations are often used in volumetric sculpting systems because they can be used to add or subtract material to the surface of an object along the swept path of a virtual sculpting tool.

When objects are represented as distance fields, Boolean

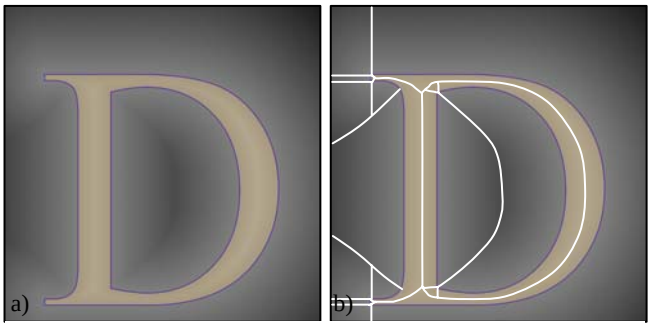
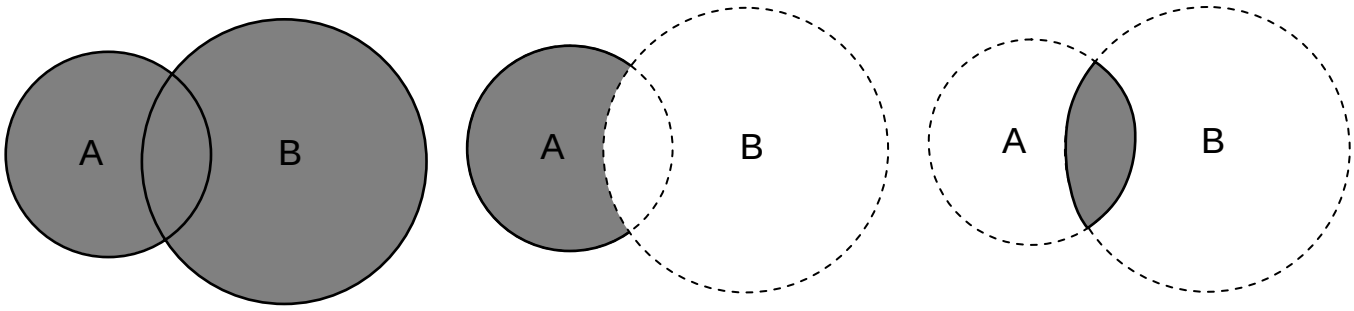


Figure 2. a) The signed 2D distance field of this letter 'D' is C^0 continuous everywhere and b) C^1 continuous everywhere except on the boundaries of Voronoi regions.



Union: $A \cup B$
$dist(A \cup B) = max(dist(A), dist(B))$

Difference: $A - B$
$dist(A - B) = min(dist(A), -dist(B))$

Intersection: $A \cap B$
$dist(A \cap B) = min(dist(A), dist(B))$

Figure 3. Distance fields can be trivially combined and edited using Boolean operations such as union, difference, and intersection. These Boolean operations can be expressed as simple min() and max() operators.

operations can be performed using simple min() and max() operators (see Table 1). Although the resultant fields are not strictly Euclidean (in particular, the combined field near sharp corners is non-Euclidean), the fields are often a reasonable approximation to the true Euclidean distance field close to the object boundary.

Operation Name	Symbolic Representation	Combined Distance
Intersection	dist(A ∩ B)	min(dist(A), dist(B))
Union	dist(A ∪ B)	max(dist(A), dist(B))
Difference	dist(A – B)	min(dist(A), –dist(B))

Table 1. Boolean operations can be performed on objects represented by distance fields using simple min() max() operators. The functions listed in this table assume a signed distance field with the object surface lying at the zero-valued iso-surface and a sign convention that uses positive distances for points inside the shape and negative distances for points outside of the shape.

2.3 Advantages of Distance Fields

Distance field have several advantages over boundary representations for representing and rendering shapes. First, distance fields represent more than just the boundary of the shape; they also provide a representation of the object's interior and the space in which the object sits. This additional information is what makes it easy to perform CSG on distance fields and also provides important information for physical simulation (e.g., it can be used to detect collisions and, if a collision occurs, to determine penetration depth and the direction from the intersecting point to the closest surface point).

Second, distance fields represent more than just a single boundary. By changing the iso-surface value, we can obtain an infinite number of offset surfaces. In contrast to boundary representations, surface offsetting with distance fields handles changes of topology robustly. This feature plays an important role in the utility and success of Level Set approaches (e.g., see Osher and Fedkiw 2002, and Sethian 1996) which use distance functions to represent evolving boundaries.

3. Applications of Distance Fields

Distance fields have been used in many fields including CAD/CAM, medical imaging and surgical simulation, modeling deformation and animating deformable models, level set methods, simulating fluid dynamics for modeling smoke and fluids, scan conversion or 'voxelization', reconstructing shape from range data, and robotics. See [Frisken and Perry, 2002] and [Jones et al., 2006] for summaries of the use of distance fields in computer graphics and computer vision.

3.1 Distance Fields in Digital Design

Early work using distance fields for digital design was done in CAD/CAM for offsetting (e.g., Ricci 1973 and Breen 1991), tolerancing (e.g., Requicha 1983), and generating rounds and filets (Rockwood 1989). Freeform design using distance fields has been done in the context of implicit surface modeling (e.g., Bloomenthal and Wyville 1990, Cani Gascuel 1993) and volume graphics (e.g., Galyean and Hughes 1991, Wang and Kaufman 1995, and Avila and Sobierajski 1996). These early freeform modeling systems typically produced 'blobby' models, i.e., organic models without sharp edges, corners, or other fine detail, thereby limiting the utility of such systems. More powerful computers coupled with the use of spatial data structures for reducing the memory requirements of sampled distance fields have recently enabled the development of systems that can produce higher resolution models (e.g., Sensable Technologies' Freeform modeling system, Baerentzen 1998, Perry and Frisken 2001, Museth et al. 2002, and Blanch et al. 2004).

4. Representing Distance Fields

4.1 Implicit vs. Sampled Representations

The distance field of simple geometric shapes such as spheres, rectangular boxes, conics, and ellipsoids can be represented implicitly. For example, the distance field of a sphere centered at the origin can be written using the implicit expression $distSphere(x,y,z) = R - (x^2 + y^2 + z^2)^{½}$. Processing implicit shape representations (e.g., for rendering, modeling via CSG operations, or performing collision tests) requires evaluating the implicit expression at query points as needed.

Implicit functions for more complex shapes are often very difficult to specify and/or too costly to evaluate, thus making an implicit representation of an object's distance field impractical. For this reason, distance fields are often represented as sampled volumes, where each sample in the volume measures the distance from the corresponding sample point to the object. The distance from an arbitrary point to the object is reconstructed from local sampled values using an interpolation function. For example, in a regularly sampled rectilinear volume, tri-linear interpolation is often used to reconstruct the distance at an arbitrary point from the 8 nearest sampled values of the volume. Figure 4 illustrates

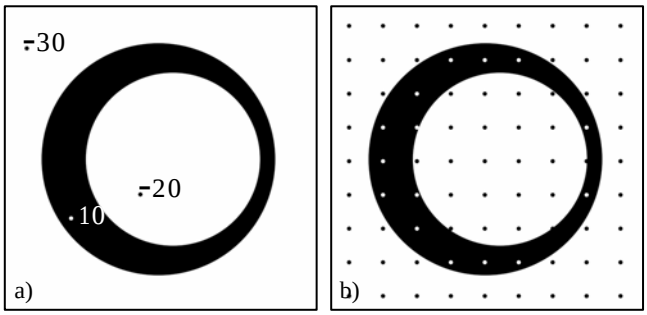


Figure 4. a) A 2D shape and 3 signed sampled distance values. b) A regular sampling of the distance field.

sampled distances to a 2D shape.

As long as the maximum curvature of an object is not too high, a sampled distance field can provide a reasonably good representation of the object's surface. As was shown in [Gibson 1998a], the surface of a sphere can be represented with a very small volume of samples, especially when both the distance and the gradient of the distance field are stored for each sample point (see Figure 5). However, for detailed models, the distance field must be sampled at high enough rates to avoid aliasing during reconstruction and rendering. Large models that have even small regions with high detail have very high memory requirements and/or limited resolution when the distance field is stored in a regularly sampled volume. Because generating the sampled representation requires evaluating the distance function at every sample point in the volume, regularly sampled volumes are also slow to generate and process.

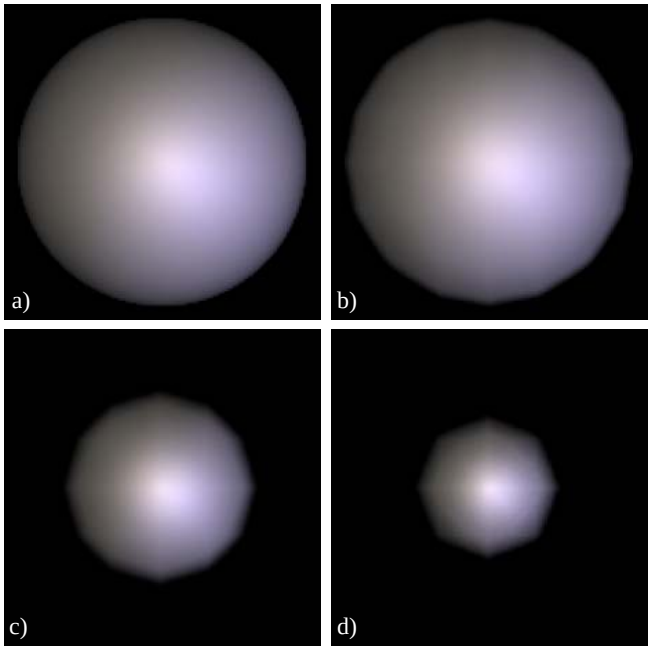


Figure 5. The surface of a sphere is well represented by a sampled distance field even at very low resolution. a) radius = 30 sample points, b) radius = 3 sample points, c) radius = 2 sample points, d) radius = 1.5 sample points.

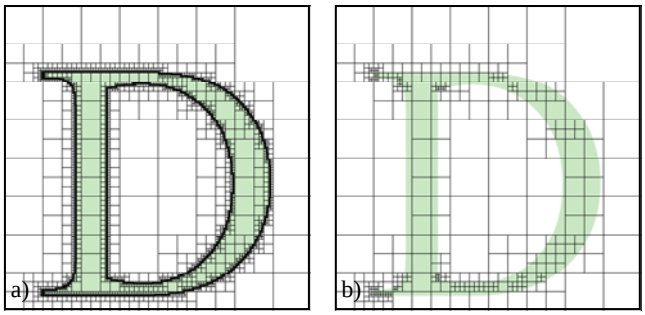


Figure 6. Quadtree representations for storing a sampled distance field of a 2D shape. a) is a boundary-limited (i.e., 3-color) quadtree in which cells are subdivided to their maximum level if they contain the shape's boundary. a) is an ADF with a biquadratic reconstruction function in which cells are subdivided according to local detail in the distance field. The ADF requires significantly fewer distance samples to achieve the same representation quality.

4.2 Improving Efficiency

There has been a significant amount of effort made to speed up the generation of regularly sampled distance fields. Many of these approaches are summarized in [Jones et al. 2006]. Researchers at the University of North Carolina [Hoff et al. 1999, Hoff et al. 2001, and Sud et al. 2004] have used graphics hardware to speed up the distance computation in 2D and later in 3D. Others reduce processing by restricting evaluation of the distance field to a 'shell' or 'narrow band' around the object surface [Curless 1996, Jones 1996, Desbrun and Cani-Gascuel 1998, and Whitaker 1998]. In some cases, accurate distance values evaluated in the shell are then propagated to voxels outside the shell using fast distance transforms [Jones and Satherley 2001, Zhao et al. 2001] or fast marching methods from level sets [Kimmel and Sethian 1996, Breen et al. 1998, Whitaker 1998, and Fisher 2001]. [Szeliski and Lavalle 1996, Wheeler 1998, and Strain 1999] evaluate distance values at cell vertices of a classic or '3-color' octree (i.e., an octree where all cells containing the surface are subdivided to the maximum octree level) to reduce the number of distance evaluations over regular sampling.

4.3 Adaptively Sampled Distance Fields

More recently, it was observed that substantial savings both in memory requirements and in the number of distance evaluations required to represent an object could be made by adaptively sampling the object's distance field according to the local complexity of the distance field rather than whether or not a surface of the object was present. [Gibson 1998a] noted that the distance field near planar surfaces can be reconstructed exactly from a small number of sample points using trilinear interpolation. This observation led to Adaptively Sampled Distance Fields (ADFs) [Frisken et al. 2000], which use detail-directed sampling, i.e., high sampling rates where there are high frequencies in the distance field and low sampling rates where the distance field varies smoothly. As illustrated in Figure 6, this approach results in a substantial reduction in the number of distance evaluations and significantly fewer stored distance values than would be required by a 3-color quadtree. ADFs are a practical representation of distance fields that provide high quality surfaces, efficient processing, and a reasonable memory footprint. [Perry and Frisken 2001] demonstrate the practical utility of

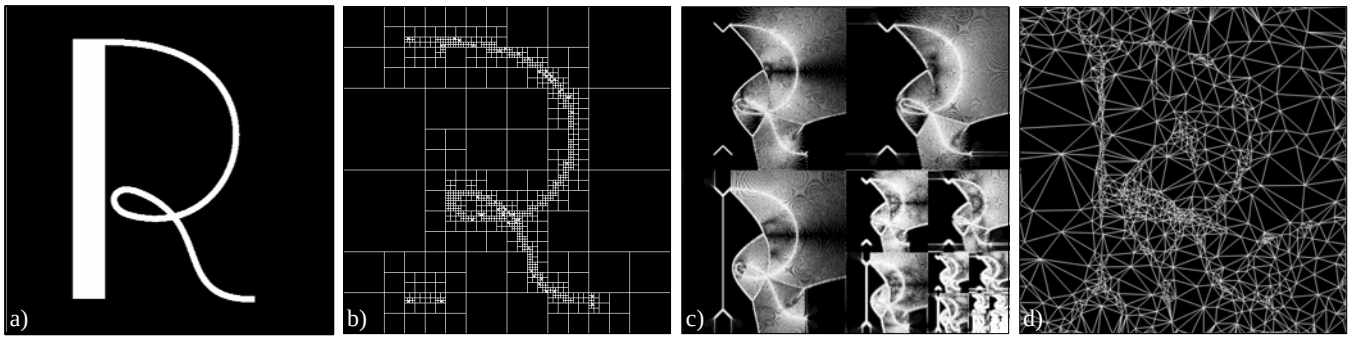


Figure 7. Various ADF instantiations: a) a 2D shape and its b) quadtree-based ADF, c) wavelet-based ADF, and d) multi-resolution triangulation-based ADF.

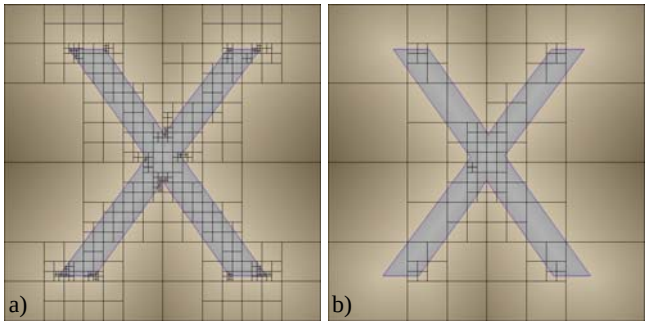


Figure 8. Improved ADFs for more accurate and efficient 2D shape representation. a) a 2D ADF with a biquadratic interpolation function for reconstructing distance values and b) an ADF with special cell representations for corners and thin sections of the shape.

ADFs in a 3D sculpting system that provides real time volume editing and interactive ray casting on a desktop PC (Pentium IV processor) for volumetric models that have a resolution equivalent to a 2048x2048x2048 volume.

While there are various instantiations of ADFs (see Figure 7 for some examples) [Frisken et al. 2002], this paper is primarily focused on quadtree and octree-based ADFs which subdivide the space enclosing an object into rectilinear cells whose size depends on the local detail of the distance field (see Figure 6b). A set of sampled distance values are stored for each leaf cell of the quadtree or octree. Distances and gradients of the distance field at arbitrary points within a cell can be reconstructed by interpolating the sampled values stored for the cell (and possibly neighboring cells). We currently use trilinear interpolation for reconstructing 3D distance fields from distances sampled at the eight corners of 3D ADF leaf cells and biquadratic interpolation for reconstructing 2D distance fields from nine sample points stored in 2D ADF leaf cells. Note that ADFs essentially subdivide space into small regions over which we have a local implicit function that is defined by the sample points associated with that region and the interpolation function. This subdivision of the globally implicit distance field into spatially-limited local implicit fields provides efficient querying and processing of the field.

Recently, we have implemented an improved 2D ADF representation that uses a biquadratic interpolation function for better quality and more efficient representation of curved edges (see Figures 6b and 8a) and specialized ADF cells that provide a compact and exact representation of the distance field near corners and thin sections of a 2D shape (see Figure 8b) [Perry and Frisken 2003, Frisken and Perry 2004].

5. Processing Adaptively Sampled Distance Fields

5.1 ADF Generation

Octree-based ADFs can be generated using a top-down tiled generation algorithm described in [Perry and Frisken 2001]. Starting with a geometric description of an object (e.g., a triangle model) and the root cell of the ADF, cells of the ADF are recursively subdivided until the field within a cell is well represented by the cell's sampled distance values and its reconstruction function. For example, for an octree-based ADF using trilinear interpolation, distances from the object to each cell vertex and distances to a set of test points within the cell are computed. The distances at cell vertices are used to reconstruct estimates of the distances at the test points; if the estimates do not match the computed distances at the test points, the cell is further subdivided. Additional data structures are used to avoid recomputing distances whenever possible and to ensure that shared distances (i.e., the distance value of a vertex that is shared by several cells) are only stored once.

5.2 Direct Rendering

3D ADFs can be rendered in several ways: directly via ray tracing and indirectly by first generating a surface representation (e.g., points or triangles) that can be rendered via a traditional graphics

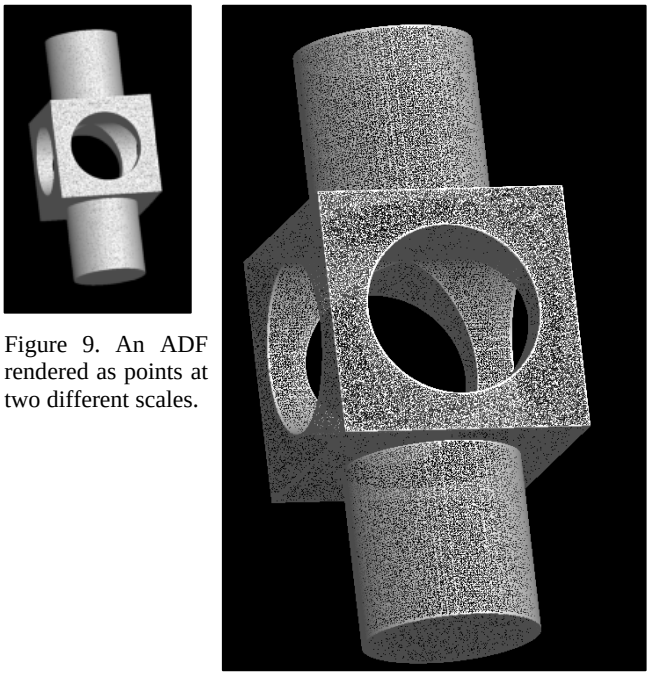


Figure 9. An ADF rendered as points at two different scales.

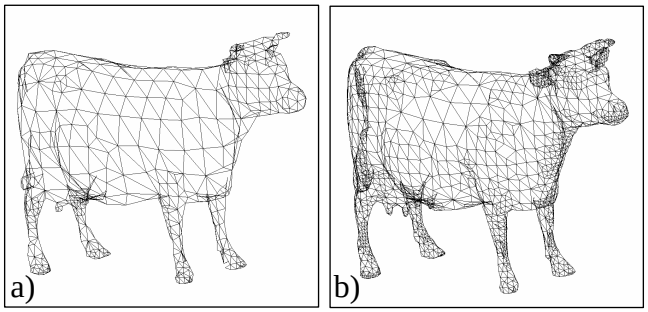


Figure 10. The octree data structure can be exploited for generating level-of-detail triangle models. a) a low resolution triangle model and b) a medium resolution model generated from an ADF.

pipeline (e.g., OpenGL). For direct rendering, a ray is cast into the ADF in the view direction for each pixel. Cells that might contain the surface (as indicated by the cell's distance values) are tested in front to back order for ray-surface intersections. If an intersection occurs, the intersection point and the gradient of the distance field at the intersection point are determined and used to compute the color of the pixel. Secondary rays (e.g., shadow rays or reflection rays) can be spawned at each intersection point for higher quality rendering. An adaptive ray casting approach can be used to achieve reasonable full-image rendering rates and fast local updates of regions that are being interactively edited [see Perry and Frisken 2001 for details].

5.3 Point-based Rendering

The octree data structure lends itself well to point-based rendering approaches [Perry and Frisken 2001]. To generate a point-based model of the surface, leaf cells of the octree that contain the surface are seeded with a set of randomly generated points. A uniform distribution of points over the surface can be achieved by seeding leaf cells with a number of points that is proportional to the size of the leaf cell (i.e., large leaf cells are seeded with more points than small leaf cells). Once the seeded points are placed in each leaf cell, they are relaxed onto the surface by following the gradient of the distance field until they reach the surface. The points can be optionally shaded using the gradient of the distance field at their final locations. This approach is quite fast, allowing 800,000 Phong-shaded points to be generated in 1/5 of a second on a Pentium II processor in 2001. Figure 9 shows a point-base model rendered via OpenGL at two different sizes.

5.4 Tessellation

ADFs can also be converted to triangle models which can be rendered interactively using graphics hardware. We use a modified SurfaceNets triangulation algorithm [Gibson 1998b, Perry and Frisken 2001] (later relabeled as Dual Contouring in [Ju et al. 2002]) to create topologically consistent, high quality triangle models on the fly. The octree data structure of the ADF can be exploited for creating Level-of-Detail triangle models (see Figure 10). The tesselation algorithm is very fast and handles adjacent octree cells whose sizes differ by greater than a factor of two. The method was able to generate 200,000 triangles in 0.37 seconds on a Pentium II processor in 2001 and is considerably faster on today's workstations.

5.5 Concept Modeling

Building on prior work in implicit modeling (see e.g., [Bloomenthal 1997]), modeling with generalized cylinders (e.g., [Crespin et al. 1996] and [Aguado et al. 1999]), and sketched-based input (e.g., [Cohen et al. 2001] and [Grimm 1999]), we have implemented a prototype system for creating expressive and detailed 3D creatures and other organic models via a simple and intuitive interaction method. Leveraging off of traditional 2D drawing, this system incorporates three design stages: 1) free-hand sketching of skeleton curves that rough out the basic shape of the object, 2) fleshing out the geometry of the creature by specifying a set of 2D cross-sectional profiles that are lofted along the skeleton, and 3) editing the lofted surface to add high resolution geometric detail via a brush-based carving metaphor. These three stages are illustrated in Figure 11.

In the second design stage, the user fleshes out the geometry by lofting 2D cross-sectional profiles along the skeleton. The profiles are represented as 2D ADFs and are edited using a new 2D profile editor that provides a seamless interface between pixel-based (painting) and vector-based (curve drawing) metaphors. Because lofting is performed as an implicit blend, the cross sections can have arbitrary topology. A new robust lofting method that exploits ADFs is used to produce high resolution models that accurately reflect the detailed shape of the 2D profiles.

5.6 Detailed Carving

ADFs provide a significant improvement over regularly sampled distance fields and distance fields stored in 3-color octrees (i.e., octrees subdivided based on the presence of an object's surface

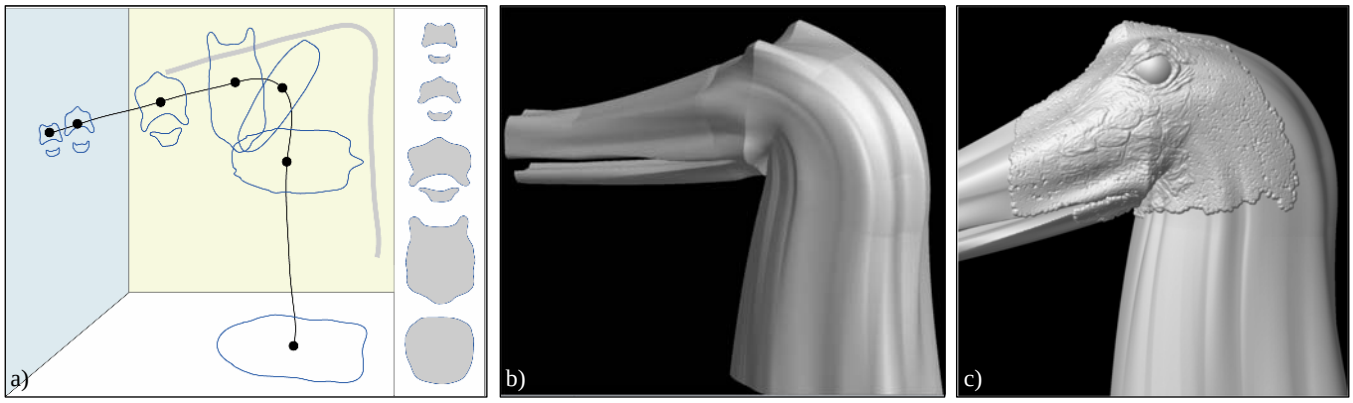


Figure 11. a) A skeleton curve defining the basic shape and a set of 2D profiles that are placed perpendicular to the skeleton to define the surface of the shape. Note that the profiles can have arbitrary topology. In b) the profiles have been lofted along the skeleton producing an expressive concept model. In c) detail has been added to the surface of the shape using a brush-based carving tool.

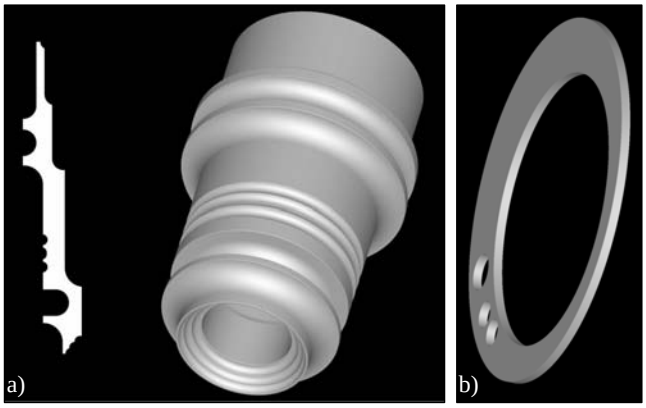


Figure 12. Detailed ADFs created using the Kizamu sculpting system. a) a 3D surface of revolution created from a sculpted 2D ADF and b) a part extruded from a 2D ADF with 3 punched holes.

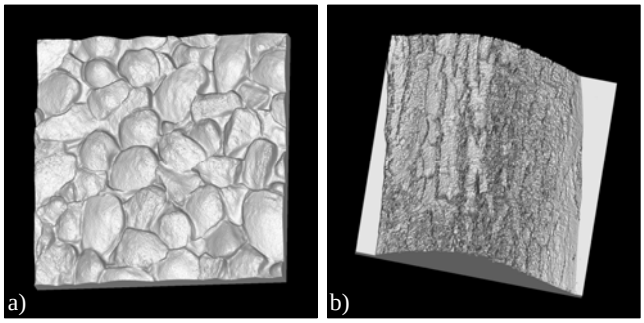


Figure 13. Highly detailed ADFs created from range data using the Kizamu sculpting system. a) stones and sand and b) tree bark.

rather than on detail in the distance field) because the smaller memory size and faster processing times of ADFs enable interactive carving at very high resolution. Carving is accomplished by performing Boolean operations (e.g., differencing or unioning) between the carving tool and the object being carved. For practical purposes, the effect of the carving tool is limited to a bounding region surrounding the tool. ADF cells of the object that lie within this bounding region are regenerated; the distance field in the regenerated cells is computed by applying the appropriate Boolean operation to the distance field of the tool and the distance field of the object.

[Perry and Frisken 2001] describe Kizamu, a system for sculpting detailed characters that uses ADFs. This system provides a means for generating ADF models from various sources such as stock distance functions (e.g., spheres, rectilinear boxes, cones, and cylinders), CSG combinations of stock distance functions, height fields and range data, extrusion and revolution of 2D ADFs, lathing of existing ADFs, and triangle models. Kizamu (i.e., "to carve" in Japanese) allows users to perform detailed carving of the surfaces of these ADF models using a pressure sensitive pen and a brush-based metaphor. The carving tool can be applied perpendicular to the viewing direction or in a direction normal to the local object surface. The system maintains a history of operations during carving and provides infinite undo and redo operations. Figures 11c, 12, and 13 show several parts generated using Kizamu, illustrating that ADFs can be used to produce smooth, organic surfaces with high quality edges and corners and intricate geometric detail.

6. Summary

The use of distance fields for representing and processing shape has application in many fields. In particular, distance fields provide an intuitive representation for digital design because they can be intuitively and efficiently combined using Boolean operations and they can be edited and manipulated in ways that resemble real clay. More efficient algorithms and efficient representations of distance fields (such as ADFs) have facilitated several systems that use distance fields for design. In this overview, we have discussed properties and advantages of distance fields for representing shape, reviewed previous work using distance fields in digital design, and described methods for representing and processing ADFs together with two systems that use ADFs for concept modeling and detailed carving.

7. References

AGUADO, A., MONTIEL, E., AND ZALUSKA, E., 1999. Modeling Generalized Cylinders via Fourier Morphing. ACM Transactions on Graphics, 18(4), pp. 293-315.

AVILA R. AND SOBIERAJSKI L. 1996. A Haptic Interaction Method for Volume Visualization. Proc. IEEE Visualization, pp. 197-204.

BAERENTZEN J., 1998. Octree-based volume sculpting. Proc. Late Breaking Hot Topics, IEEE Visualization, pp. 9–12.

BISWAS, A. AND SHAPIRO, V. 2004. Approximate Distance Fields with Non-Vanishing Gradients. Graphical Models, 66(3), pp. 133-159.

BLANCH, R., FERLEY, E., CANI, M-P., AND GASCUEL, J., 2004. Non-Realistic Haptic Feedback for Virtual Sculpture. Research report 5090, INRIA, France.

BLOOMENTHAL, J. AND WYVILLE, B. 1990. Interactive Techniques for Implicit Modeling. Computer Graphics, 24(2), pp. 109-116.

BLOOMENTHAL, J. (ED.), 1997. Introduction to Implicit Surfaces. Morgan Kaufman Publishers.

BREEN, D., 1991. Constructive Cubes: CSG Evaluation for Display Using Discrete 3D Scalar Data Sets. Proc. Eurographics, pp. 127-141.

BREEN, D., MAUCH, S., AND WHITAKER, R., 1998. 3D Scan Conversion of CSG Models into Distance Volumes. Symp. on Volume Visualization, pp. 7-14.

COHEN, J., MARKOSIAN, L., ZELEZNIK, R., AND HUGHES, J., 1999. An Interface for Sketching 3D Curves. Proc. Interactive 3D Graphics, pp. 17-21.

CRESPIN, B., BLANC, C., AND SCHLICK, C., 1996. Implicit Swept Objects. Proc. Eurographics, pp. 165-174.

CURLESS, B. AND LEVOY, M., 1996. A Volumetric Method for Building Complex Models from Range Images. ACM SIGGRAPH, pp. 303-312.

DESBRUN, M. AND CANNI-GASCUEL, M-P., 1998. Active Implicit Surface for Animation. Graphics Interface, pp. 143-150.

FISHER, S. AND LIN, M., 2001. Fast Penetration Depth Estimation for Elastic Bodies Using Deformed Distance Fields. IEEE/RSJ International Conference on Intelligent Robots and Systems.

FRISKEN, S., PERRY, R., ROCKWOOD, A., AND JONES, T., 2000. Adaptively Sampled Distance Fields: a General Representation of Shape for Computer Graphics. ACM SIGGRAPH, pp. 249-254.

FRISKEN, S. AND PERRY, R., 2002 Efficient Estimation of 3D Euclidean Distance Fields from 2D Range Images. Proc. IEEE Symposium on Volume Visualization, pp. 81-89.

FRISKEN, S., PERRY, R., AND JONES, T., 2002 Detail-Directed Hierarchical Distance Fields. U.S. Patent 6,396,492.

FRISKEN, S. AND PERRY, R., 2004. Method for Generating an Adaptively Sampled Distance Field of an Object with Specialized Cells. U.S. Patent Pending.

GASCUEL, M-P. 1993. An implicit Formulation for Precise Contact Modeling between Flexible Solids. ACM SIGGRAPH, pp. 313-320.

GALYEAN T. AND HUGHES J., 1991. Sculpting: an Interactive Volumetric Modeling Technique. ACM SIGGRAPH, pp. 267-274.

GIBSON, S. 1998a. Using DistanceMaps for Smooth Surface Representation in Sampled Volumes. Symp. Volume Visualization, pp. 23-30.

GIBSON, S. 1998b. Constrained Elastic SurfaceNets: Generating Smooth Surfaces from Binary Segmented Data. Proceedings MICCAI.

GRIMM, C., 1999. Implicit Generalized Cylinders using Profile Curves. Proc. Implicit Surfaces.

HOFF III, K., CULVER, T., KEYSER, J., LIN, M. AND MANOCHA, D., 1999. Fast Computation of Generalized Voronoi Diagrams Using Graphics Hardware. ACM SIGGRAPH, pp. 277-285.

HOFF III, K., ZAFERAKIS, A., LIN, M., AND MANOCHA, D., 2001. Fast and Simple 2D Geometric Proximity Queries Using Graphics Hardware. Symp. Interactive 3D Graphics.

JONES, M., 1996. The Production of Volume Data from Triangular Meshes Using Voxelization. Computer Graphics Forum, 15(5), pp. 311-318.

JONES, M. AND SATHERLEY, R., 2001. Shape Representation Using Space Filled Sub-Voxel Distance Fields. Int. Conf. Shape Modeling and Applications, pp. 316-325.

JONES, M., BAERENTZEN, J. A., AND SRAMEK, M., 2006. 3D Distance Fields: A Survey of Techniques and Applications, accepted for IEEE Transactions on Visualization and Computer Graphics.

JU, T., LOSASSO, F., SCHAEFER, S., AND WARREN, J., 2002. Dual Contouring of Hermite Data. ACM SIGGRAPH, pp. 339-346.

KIMMEL, R. AND SETHIAN, J. 1996. Fast Marching Methods for Computing Distance Maps and Shortest Paths. CPAM Report 669, Univ. of California, Berkeley.

MUSETH, K., BREEN, D., WHITAKER, R., AND BARR, A., 2002. Level set surface editing operators. ACM SIGGRAPH, pp. 330-338.

PERRY R. AND FRISKEN, S. 2001. Kizamu: A System for Sculpting Digital Characters. ACM SIGGRAPH, pp. 47-56.

PERRY R. AND FRISKEN S., 2003. Method for Generating a Two-Dimensional Distance Field within a Cell Associated with a Corner of a Two-Dimensional Object. U.S. Patent 7,034,830.

RICCI, A., 1973. A Constructive Geometry for Computer Graphics. Computer Journal, 16(2), pp. 157-160.

REQUICHA, A., 1983. Toward a theory of geometric tolerancing. International Journal of Robotics Research, 2(4), pp. 45-60.

ROCKWOOD, A., 1989. The Displacement Method for Implicit Blending in Solid Models. ACM Trans. Graphics, 8(4), pp. 279-297.

SRAMEK, M. AND KAUFMAN, A., 1999. Alias-free voxelization of geometric objects. IEEE Trans. on Visualization and Computer Graphics, 3(5), pp. 251-266.

STRAIN, J., 1999. Fast Tree-based Redistancing for Level Set Computations. J. Comp. Physics, 152, pp. 648-666.

SUD, A., OTADUY, M. A., AND MANOCHA, D., 2004 DiFi: Fast 3D Distance Field Computation Using Graphics Hardware. Computer Graphics Forum 23(3), pp.557-566.

SZELISKI, R. AND LAVALLE, S., 1996. Matching 3-D anatomical surfaces with non-rigid deformations using octree-splines. Int. J. Computer Vision, 18(2), pp. 171-186.

WANG S. AND KAUFMAN A., 1995. Volume sculpting, Symposium on Interactive 3D Graphics, pp. 151-156.

WHEELER, M., SATO, Y., AND IKEUCHI, K., 1998. Consensus surfaces for Modeling 3D Objects from Multiple Range Images. Int. Conf. Computer Vision.

WHITAKER, R., 1998. A Level-Set Approach to 3D Reconstruction from Range Data. Int. J. Computer Vision, pp. 203-231.

ZHAO, H-K., OSHER, S., AND FEDKIW, R., 2001. Fast Surface Reconstruction using the Level Set Method. 1st IEEE Workshop on Variational and Level Set Methods, pp. 194-202.

Towards Virtual Clay

Marie-Paule Cani
GRAVIR lab (IMAG-INRIA) and INP Grenoble

Alexis Angelidis*
Dynamics Graphics Project, University of Toronto

Abstract

Providing the user with an intuitive sculpting system similar to real clay is one of the most challenging goals of interactive shape modeling. A user should ideally be able to deform, add and remove material freely in real time, without any geometric or topological constrains on the modeled shape.

This chapter reviews and compares three techniques that bring virtual shape modeling closer to this objective. The two first ones rely on a specific representation of the sculpted shape, namely the iso-surface of a scalar field stored in a grid. Because this representation conveniently captures topological changes, adding and removing material is straightforward. Based on this representation, we compare a geometric versus a physically-based method for handling local and global shape deformations that make the model closer to virtual clay. We also discuss solutions for providing users with an intuitive interface with haptic feedback. A third and very different approach is to define the deformations of the sculpted model as spatial deformations, such that the operation is applicable to a wide range of shape representations. We show that the extension of sweepers (presented in the space deformation chapter of this tutorial) to constant volume "swirling-sweepers" produces an intuitive clay-like behavior of the modeled shape. This technique provides a very good alternative to physically-based virtual clay when preserving the shape's topology is desirable.

1 Introduction

Making the creation and edition of a virtual shape as straightforward as the manipulation of real clay is an unsolved challenge. Compared to standard tools for virtual shape modeling, real clay is a simple and often familiar way to create complex shapes: Play-Doh® is introduced to children since kindergarten. And many artists still prefer to express themselves by interacting with real clay rather than using a computer, even if the resulting shape needs to be digitalized for subsequent use. This occurs in a variety of fields such as automotive design or the modeling of 3D characters for the film industry. The attraction towards real clay for drafting a shape is due to several reasons: In standard digital modeling systems, hand manipulation and the sense of touch are replaced by a more complex and indirect manipulation interface. Also, the operations that the user performs on the shape are intuited by his understanding of the underlying mathematics describing the shape: he cannot merely push or pull the object's surface. This is the case, for instance, when manipulating the control points of a NURBS or a subdivision surface. In addition, making holes or connecting separated parts of the shape is not always a straightforward operation.

If one could get the advantages of real clay in a computer-based modeling system, one could have the best of both worlds. As opposed to real clay, virtual clay is not affected by drying nor cracking, it may be mutated back and forth between soft and dry states as needed, and the artist may take a pause at any time without worrying about the material changing in the meanwhile. Furthermore, gravity may be switched off, and the artist would not have to worry about the shape collapsing under its own weight. Finally, all the advantages of standard shape 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, 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 and compares different approaches towards *virtual clay* for interactive modeling. After a quick review of related work, we first detail a solution based on purely geometric and volumetric modeling [Ferley et al. 2000; Ferley et al. 2002; Blanch et al. 2004]. The modeled shape is defined as an iso-surface of a scalar field stored in a virtual multi-grid, enabling quick prototyping of arbitrary shapes with no limitation in space extent or in level of detail. Solutions for enabling local deformations in this framework and for generating haptic feedback are provided. The second model [Dewaele and Cani 2004a; Dewaele and Cani 2004b] introduces more general local and global deformations using a similar representation. This layered physically-based model captures in real-time important properties of real clay: plasticity, mass preservation and surface tension. Intuitive user interaction in this context is discussed. Volume being probably the most important property of real clay, we finally present a deformation technique that preserves the volume of a shape. This is done implicitly, by preserving the volume locally, at every point in space. These deformations also preserves the topology of the shape, which may or may not be a desirable property. The operations are applied in real-time to a surface represented with an adaptive mesh. User action is intuitive and produces clay-like deformations of the shape [Angelidis et al. 2004a].

2 Related work

2.1 Implicit surfaces

Implicit representations have been identified for long as a very good model for representing clay-like objects [Bloomenthal et al. 1997]: in addition to their well known ability to represent smooth, deformable shapes, they ensure a *coherent* definition of a closed surface with a well defined interior and exterior. They also handle conveniently any kind of topological change, such as digging a hole in an object, splitting

*On leave from Graphics & Vision lab, U. of Otago.

it into several parts or merging several disconnected components.

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. Interactive sculpting with implicit surfaces has thus been tackled using 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. These approaches are presented next.

2.2 Volumetric sculpting

Interactive modeling based on discrete scalar field representation was first introduced in 1991 by T. Galyean and J. Hughes [Galyean and Hughes 1991]. 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 [Wang and Kaufman 1995] 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 [Avila and Sobierajski 1996b] 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.

Several extensions enabling the multi-resolution visualization or edition of the sculpted shape were introduced in the next few years: J. Bærentzen [Bærentzen 1998] proposed an octree-based representation of the field, aimed at accelerating ray-casting. S. Frisken [Frisken et al. 2000; Perry and Frisken 2001] presented the Adaptively Sampled Distance Field (ADF) approach detailed in the previous chapter of this tutorial. A. Raviv and G. Elber [Raviv and Elber 2000] proposed a different hierarchical approach based on the combination of trivariate B-Spline volumes to represent the field.

2.3 Physically-based modeling

The volumetric models we just reviewed were restricted to simple operations such as adding material or carving it, but contrary to real clay, did not provide any mechanism for deforming an existing shape. K. Mc Donnel and H. Qin [McDonnell et al. 2001; McDonnell and Qin 2002] explored the introduction of physically-based deformations within such a volumetric sculpting framework. They addressed more specifically the cases when the field function is defined by either a trivariate B-spline function of by a subdivision solid. Masses and spring networks attached to the control polygon defining the field were used to generate deformations of the sculpted shape. Although indirect, this interaction with the structure storing the field achieved interesting results in terms of interactive deformations with haptic feedback.

Closer to the physically-based modelling of clay, cellular automata were introduced in a volumetric, discrete field representation to allow free-form modeling with volume conservation and topological changes [Arata et al. 1999; Druon et al. 2003]. However, these models did not capture large-scale deformations such as bending the limbs of a clay model.

More generally speaking, real clay (see Figure 1) lies between plastic solids and viscous fluids, so one could think of extending either of these models for setting up a physically-based representation. However, the existing real-time models for elastic or plastic-solids rely on pre-computations or on a hierarchical structure for achieving a high frame rate. This prevents from using them in a framework where the objects topology changes over time. Both Eulerian simulations and particle systems (now often called *mesh-less models*) have been used for representing viscous fluids. They have been rendered using an implicit surface (or a level-set) or through point-based rendering, both approaches conveniently capturing the resulting topological changes. In addition to ensuring real-time performances, the difficulty with these models is to set up internal cohesion forces that make the model behave as clay. In particular, a piece of material lying on the ground should not spread as a viscous liquid would do. A more detailed discussion of physically-based deformable models is over the scope of this tutorial, but a very good survey can be found in [Nealen et al. 2005]. No mention can be found of a real-time, physically-based model convenient for clay.

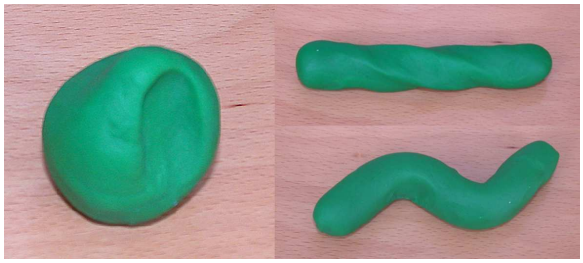


Figure 1: Real clay lies between plastic solids and viscous liquids. As such, it can undergo local and global deformations, including as weel as changes of toology such as splitting and merging.

3 Implicit sculpting with local deformations

This section details the implementation of a sculpting system which enables, in addition to the addition and removal of material, the application of local deformations to the shape [Ferley et al. 2000]. The shape representation is 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 is depicted on Figure 2.

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).

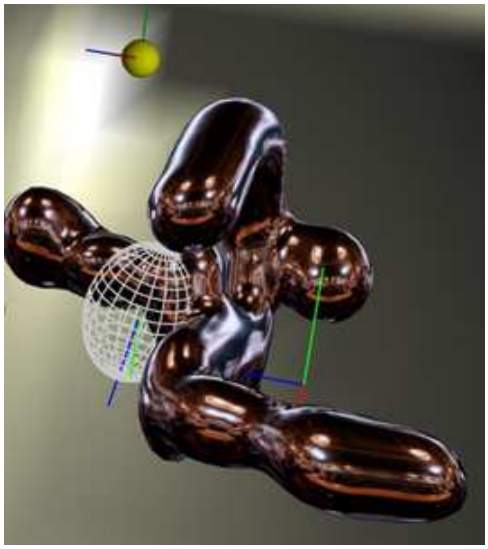


Figure 2: Sample snapshot of our sculpting application. The object being modeled is environment mapped so that the user can better appreciate its shape. The sculpting tool is displayed in wireframe. The yellow spheres represent the lights that the user can move around while sculpting.

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 saves computational time.

The simplest analytical primitives that can be implemented are spheres and ellipsoids. We use Wyvill's field function [Wyvill et al. 1986] to generate an isotropic (spherical) field around the tool center. General ellipsoids can be obtained by scaling the tool along its three axes.

The user can also sculpt freeform tool shapes within our application. The shape displayed for the tool corresponds to the same iso-surface as the one visualized during its design process. The tool can also be scaled along the three axes of its local frame before being re-used. 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 [Galyean and Hughes 1991; Avila and Sobierajski 1996b]:

- 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 un-smoothly 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 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 [Cani and Desbrun 1997].

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. Finally the contribution vanishes to zero as we reach the border of the tools region of influence. These combined negative and positive actions imitate the displacement of matter that occurs when a real tool collides with a block of clay. 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 [Ferley et al. 2000] for details). A local deformation tool and the deformation it produces are depicted in Figures 3. Figures 4 shows an inprint onto a sculpted surface made by a tool sculpted within our application.

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

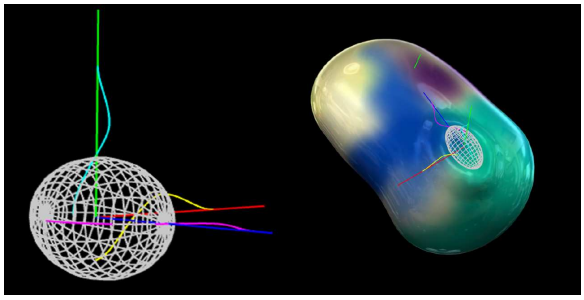


Figure 3: (left) Ellipsoidal local deformation tool (right) Deformation made by sweeping this tool over a block of virtual clay.

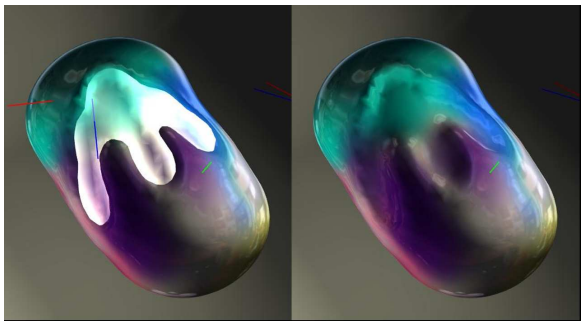


Figure 4: Local deformation created by a tool sculpted within our system.

arbitrary `minVal` and `maxVal`, 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`, which samples the region of space where some material has been deposited. 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 called `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 standard step in the Marching-cube algorithm; see [Bloomenthal et al. 1997] for details).
- 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, an `Edge` structure is created to store the intersections of a Cells edge with the iso-surface.

We tried two different implementations for the above dynamical 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 [Ferley et al. 2000]).

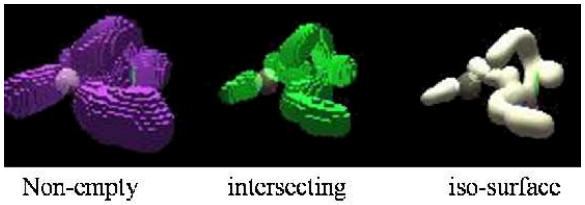


Figure 5: Data structures for the field representation. Left: the Cell-Tree. Middle: the cross-list. Right: the sculpted surface

3.3 Applying a tool: data structures update

Each time a tool is applied all the dynamical structures we just defined need to be quickly, locally updated. To this end, we first compute the axis-aligned bounding box surrounding the local tool bounding box. Then, we walk through this box by:

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 Practical use of the system

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.

An example of complex object created with our sculpting system is depicted in Figure 6. 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 6: 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 [Blanch et al. 2004]

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 [Massie and Salisbury 1994]. Figure 3 shows the use of the *Phantom desktop* to model a character.

Figure 7: 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 [Avila and Sobierajski 1996a; Avila 1998] 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 motion.

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

The parameters in equation (1) are: α, 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.

$$\vec{f_c} = -f_{c0} \frac{\vec{grad}(V)}{\|\vec{grad}(V)\|} \left(\frac{V}{V_0}\right)^e \quad (2)$$

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 [Huang et al. 1998]. 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-tacto-realistic" 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 $\beta_p < \beta_a$.

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

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

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 [Gillespie and Cutkosky 1996].

The original solution presented here is a particular case of virtual coupling introduced in [Colgate et al. 1995] without using complex linear circuit theory as in [Adams and Hannaford 1999].

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 $\vec{p_r}$, τ being the time constant of the filter.

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

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 γ 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 λ characterizes the amplitude of movement we want to cut.

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

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 relations above. 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 λ, 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 λ. 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. This ensures the good immersion of the user, making artistic work possible.

5 Multi-resolution implicit 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 un-divides 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 [Ferley et al. 2002]

5.1 Hierarchical scalar field

We extend the idea of dynamic structures to store the field function described 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 9. In order to allow a multiresolution representation, a cell isn't replaced by a set of sub-cells, 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 9), 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;

Figure 8: Use of haptic feedback: This sculpture was achieved within less than one hour, to be compared with the three hours needed to sculpt the bust in Figure 6, without feedback forces.

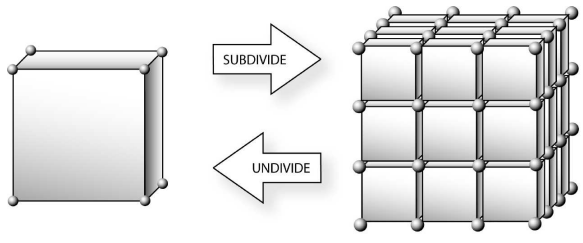


Figure 9: Subdivision principle. Several choices are possible such as replacing the left *Cell* by the subdivided *Cells* on the right or referencing the subdivided *Cells* as children of the left *Cell* (e.g. enriching the left *Cell*), duplicating or not the *Vertex* elements that have the same position, etc.

(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 9 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 10, where the sampling rate increases (i.e. the cell size decreases) in regions where the tool has sharp features.

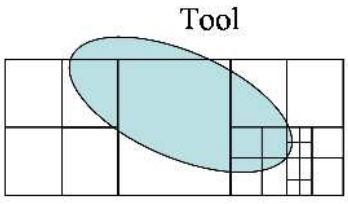


Figure 10: Sampling an ellipsoidal tool. (a). Only 2 consecutive levels. (b). All the levels hierarchically created.

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.

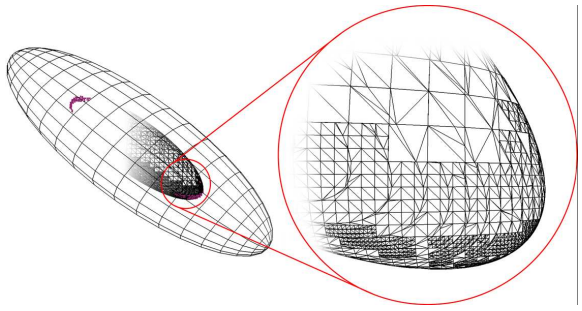


Figure 11: Sampling an ellipsoidal tool: the large ellipsoid is the tool, and the small one inside it is the surface created. The figure shows the maximum resolution reached in highly curved areas.

As stated above, using a subtractive tool can cause discon-

tinuities 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() > acceptableDis-
pcrepancy) {
                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 9) 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 subcells 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 12), it is thus *a posteriori* not worth the effort. An offline global polygonization is computed when the sculpted object needs to be exported.

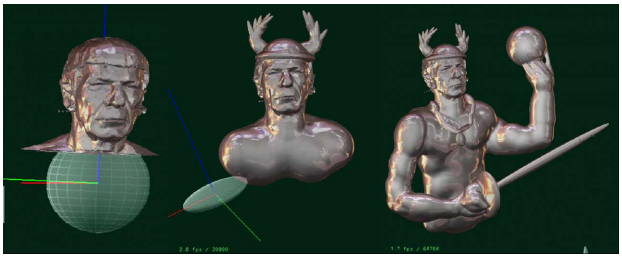


Figure 12: Three steps of a character's modelling through the editing of an imported polygonal mesh. This example illustrate multi-resolution sculpting, since large tools were used for creating a smooth body for the character while very small ones were needed to create the wings of the helmet and the chain.

6 Virtual Clay with local and global deformations

Although the previously described sculpting techniques offer real-time interaction and several interesting features such as force feedback or multi-resolution editing, the user's action is mostly restricted to carving and adding material. With a very limited kind of local deformation, this method fails to fulfill the versatile modeling interface we are looking for. Several of the essential features of real clay are not simulated, such as the ability to be globally bent or twisted, the preservation of volume during global and local deformations and surface tension.

This section presents a volumetric, real-time virtual clay model which can be both sculpted by adding or removing material and deformed through the interaction of rigid tools. The model mimics the global and local effects of plasticity, mass preservation and surface tension found in real clay. Our method enables the user to specify 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 discussed in conclusion. This work was first presented in [Dewaele and Cani 2004a; Dewaele and Cani 2004b]

In this section, the clay surface is defined as the iso-surface of iso-value 0.5 of a scalar field that represents 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 which, 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 first layer allows the user to bend or twist 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:** This 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 neighboring cells. This results into intuitive folds and local bulges when the user deforms or presses the clay.

3. **Surface tension:** This 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 reacts very similarly to real clay, as our results show. Figure 13 gives a schematic representation of the three layers.

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 δ to apply to the 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

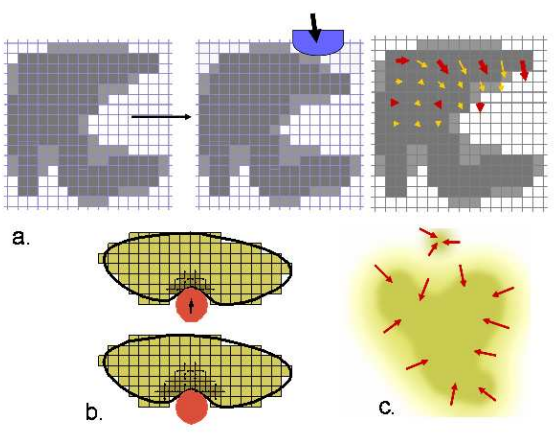


Figure 13: A layered model for virtual clay. (a) Large-scale deformations modeling plasticity. (b) Local deformations insuring constant volume. (c) Surface tension avoiding the spreading of clay over space.

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 δ of the clay in a cell as:

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

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:

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

where ρ_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 δ_O is the translation of point O (the center of the tool) and ω, the screw of the tool. To take into account the rotation of the tool, we simply replace the previous δ_i in equation (13) by the δ_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 the contraction of the clay, this layer tries to avoid expansion. It will separate the object in two different compact parts if the user stretches it too far, due to the decrease of density in the central region.

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 ρ_i represent the amount of clay being transported to the destination cell and ν_i be the vector of associated local properties. Let ρ_j and ν_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:

$$\nu_{destination} = \frac{\rho_j \nu_j + \rho_i \nu_i}{\rho_j + \rho_i} \tag{16}$$

For instance, in the case of fluidity, ν 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:

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

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

6.6 Results and discussion

Figure 14 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

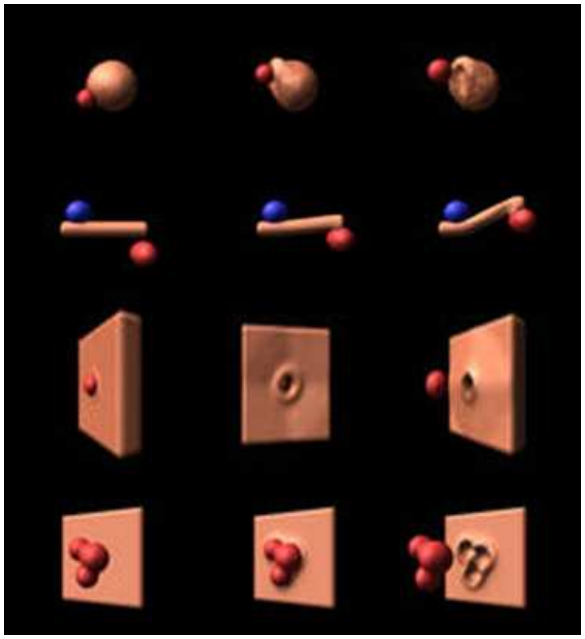


Figure 14: Snapshops of the virtual clay sculpting system.

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.

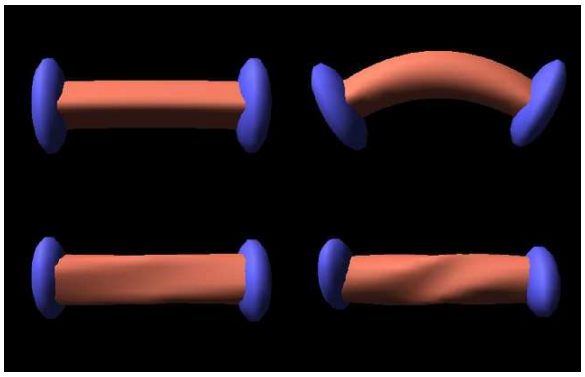


Figure 15: Globally bending and twisting a piece of clay.

Figure 15 shows the ability of the model to take into acound both translations and rotational motion of the tools in contact with the clay.

Figure 16 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.

Comparing these results with the real images depicted in figure 1 shows that our model achieved capturing the main features of real clay. However, there is a good reason why this physically-based system was only used, up to now, to perform very simple, proof-of-concept deformations: the closer a model is to real clay, the most difficult interaction is. Although possible, sculpting a complex shape using a single, rigid tool controlled with the mouse is difficult and time-consuming. Indeed, most people would use their ten fingers

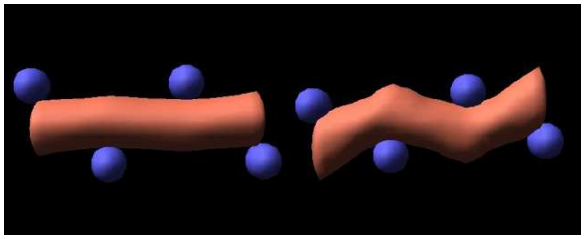


Figure 16: Deforming a piece of virtual clay through the simultaneous action of four tools, to be compared to the real-clay deformation at the bottom right of figure 1.

and may-be the palm of their hand for modeling with real clay. Although interaction with an arbitrary number of tools is made possible by our model, a good way for the real-time control of these multiple tools still needs to be defined. The best solution would be the real-time control of virtual hands interacting with the clay. This could be done by either using a data glove or a system based on cameras for capturing the motion of the users hands. However, finding a method for providing a convincing force feedback in such a framework would be very difficult, since the sense of touch is very different from the kind of feedback an exo-skeleton glove can provide.

This leads to the question: how far should a digital model for clay try to mimic the behavior of real material? Gestured-based deformations as the sweepers technique presented in the space deformation chapter provide a much easier control, since the user can place the deformation tool either inside, outside or partly inside the model to control the part of the shape he wants to deform. Could we add constraints to generated deformations, so that they become as natural to us as real clay deformations, while keeping the same easiness of use? This is the approach we are studying next.

7 Swirling-sweepers: constant volume space deformations

In a non-virtual modeling context, one of the most important factors that affects the artist's technique is the amount of available material. This notion is not only familiar to professional artists, but also to children who play at kindergarten with Play-Doh® and to adults through everyday life experience. A shape modeling technique that preserves volume will take advantage of this familiarity, and will hopefully be genuinely intuitive to use.

The model presented in this section is an extension to constant volume deformations of the sweepers technique presented in the chapter on space deformation of this tutorial. As such, it combines the advantages of a very simple gesture-based interaction with characteristics similar to real clay.

7.1 Swirl

We introduce a particular case of sweeper, a *swirl*, as a rotation whose magnitude decreases away from its center. Formally, a swirl is defined by a center point $\mathbf{c}$, together with a rotation of angle θ around an axis $\mathbf{v}$ (see Figure 17). A radial function φ, defines how the amount of rotation decreases

away from $\mathbf{c}$, within a sphere of radius λ around $\mathbf{c}$:

$$\varphi(\mathbf{p},\lambda) = \mu(\frac{\|\mathbf{p}-\mathbf{c}\|}{\lambda}) \quad (18)$$

$$\text{where } \mu(d) = \begin{cases} 0 & \text{if } 1 \leq d \\ 1 + d^3(d(15-6d)-10) & \text{if } 1 > d \end{cases}$$

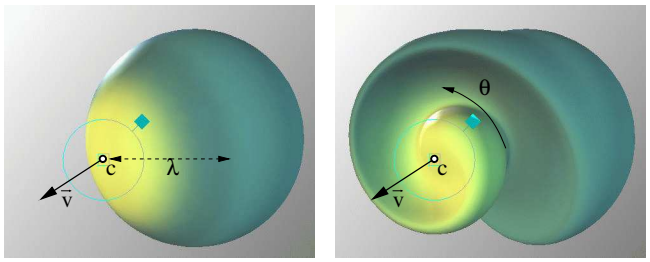


Figure 17: The effect on a sphere of a swirl centered at c, with a rotation angle θ around $\vec{v}$. The two shapes have the same volume.

The function μ is a picewise C^2 polynomial, antisymmetric about 0.5. Although in this section we will develop our technique on this choice for μ, alternative functions may also be used. Informally, a swirl twists space locally around axis $\mathbf{v}$ without compression or dilation (see proof in [Angelidis et al. 2004b]), thus it preserves implicitly[1] the volume of any shape embedded in the deformed space. The equation defining the effect of a swirl on space may be defined in several equivalent ways, for example using quaternions or an algebraic formula. For convenience reasons that will become apparent in the next section, we chose to define the effect a swirl using the exponential and logarithm of matrices (see a complete overview in [Alexa 2002]), for which we will give *closed-forms* in Section 7.3):

$$f(\mathbf{p}) = \exp(\varphi(\mathbf{p},\lambda)\log R)\cdot\mathbf{p} \quad (19)$$

where R denote the 4×4 matrix of a rotation of center $\mathbf{c}$, axis $\mathbf{v}$ and angle θ. The exponential or logarithm of a 4×4 matrix is a 4×4 matrix, and are formally defined as the limit series $\exp M = \sum_{k=0}^{\infty}\frac{M^k}{k!}$ and $\log N = -\sum_{k=1}^{\infty}\frac{(\mathrm{I}-N)^k}{k}$.

7.2 Ring of Swirls

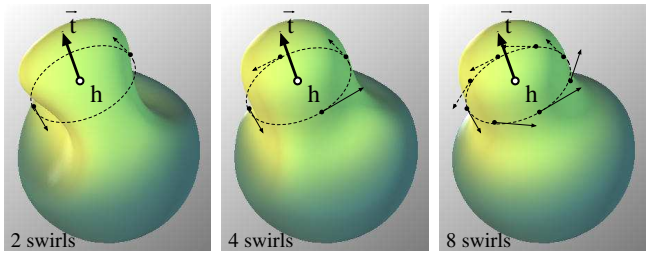


Figure 18: By arranging n basic swirls in a circle, a more complex deformation is achieved. In the rightmost image: with 8 swirls, there are no visible artifacts due to the discrete number of swirls.

With the exp and log formulation, it is very convenient to define the effect of simultaneous swirls, simply by summing the weighted matrix logarithms:

$$f(\mathbf{p}) = \left(\exp(\sum_{i=0}^{n-1}(\varphi_i(\mathbf{p},\lambda)\ \log R_i))\right)\cdot\mathbf{p} \quad (20)$$

It is important here to remark that the above blending is *not* the blending formula of simultaneous tools defined in [Angelidis et al. 2004b], and uses simple weights. The reason for using the above simple blending equation as opposed to the one of [Angelidis et al. 2004b] is that the latter modulates the amount of individual transformations locally, and attempting to control the volume with it would be inappropriate. Using swirls as building blocks, we now provide a useful way for the artist to input n swirls by specifying a single translation $\mathbf{t}$. Let us consider n points, $\mathbf{c}_i$, on the circle of center $\mathbf{h}$, and radius r lying in a plane perpendicular to $\mathbf{t}$. To these points correspond n consistently-oriented unit tangent vectors $\mathbf{v}_i$ (see Figure 18). Each pair, $(\mathbf{c}_i,\mathbf{v}_i)$, together with an angle, θ_i, define a rotation. Along with radii of influence $\lambda = 2r$, we can define n swirls. The radius of the circle r, is left to the user to choose. The following value for θ_i will transform $\mathbf{h}$ exactly into $\mathbf{h}+\mathbf{t}$ (see justification in [Angelidis et al. 2004b]):

$$\theta_i = \frac{2\|\mathbf{t}\|}{nr} \quad (21)$$

With such defined swirls, the deformation of Equation (20) is a deformation tool capable of transforming a selected point to a desired location. We show in Figure 18 the effect of the tool for different values of n; in practice, 8 swirls are sufficient.

Preserving Coherency and Volume If the magnitude of the input vector $\mathbf{t}$ is too large, the deformation of Equation (20) will produce a self-intersecting surface, and will not preserve volume accurately. The reason for self-intersection is explained in [Angelidis et al. 2004b]. The volume is not accurately preserved because the blending of Equation (20) blends the transformation matrices, and not the *real* deformations. To correct this, it is necessary to subdivide $\mathbf{t}$ into smaller vectors for the same reasons that applies to solving discretely a first order differential equations. Thus foldovers and volume preservation are healed with the same strategy. The number of steps must be proportional to the speed of the translation, and inversely proportional to the size of the tool. We use:

$$s = \max(1, \lceil 4\|\mathbf{t}\|/r\rceil) \quad (22)$$

As the circle sweeps space, it defines a cylinder. Thus the swirling-sweeper is made of ns basic deformations. Figure 19 illustrates this decomposition of the deformation applied to a shape.

7.3 Swirling-Sweepers Algorithm & Implementation

We summarize here the swirling-sweepers algorithm:

Input point $\mathbf{h}$, translation $\mathbf{t}$, and radius r
Compute the number of required steps s
Compute the angle of each step, $\theta_i = \frac{2\|\mathbf{t}\|}{nrs}$
for each step k from 0 to $s-1$ **do**
  for each point $\mathbf{p}$ in the tool's bounding box **do**
    $M = 0$
    for each swirl i from 0 to $n-1$ **do**
      M += $\varphi_k^i(\mathbf{p},\lambda)\log R_{i,k}$

[1] The term implicit refers to volume preservation, and is *not* related to the implicit surface representation of previous section.

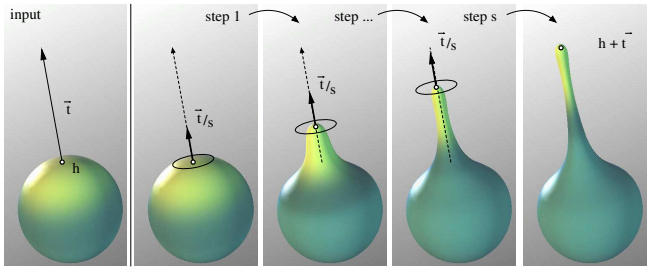


Figure 19: A volume preserving deformation is obtained by decomposing a translation into circles of swirls. 3 steps have been used for this illustration. As the artist pulls the surface, the shape gets thinner. The selected point's transformation is precisely controlled.

end for
$\mathbf{p} = (\exp M) \cdot \mathbf{p}$
end for
end for

The point $\mathbf{c}_{ik}$ denotes the center of the i^{th} swirl of the k^{th} ring of swirls. For efficiency, a table of the basic-swirl centers, $\mathbf{c}_{ik}$, and a table of the rotation matrices, $\log R_{i,k}$, are precomputed. We have a closed-form for the logarithm of the involved matrix, given in Equations (23) and (24), saving an otherwise expensive numerical approximation:

$$\begin{aligned} \mathbf{n} &= \theta_i \mathbf{v}_i \\ \mathbf{m} &= \mathbf{c}_{i,k} \times \mathbf{n} \end{aligned} \tag{23}$$

$$\log R_{i,k} = \begin{pmatrix} 0 & -n_z & n_y & m_x \\ n_z & 0 & -n_x & m_y \\ -n_y & n_x & 0 & m_z \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{24}$$

Note that this matrix is almost antisymmetric, thus we can handle it with a pair of vectors, $(\mathbf{n}, \mathbf{m})$. Once M is computed, we use a closed-form for computing $\exp M$. Since the matrix M is a weighted sum of matrices $\log R_{i,k}$, the matrix M is of the form of Equation (24) as well, and can be represented with a pair $(\mathbf{n}_M, \mathbf{m}_M)$. If $\mathbf{n}_M = 0$, then $\exp M$ is a translation of vector $\mathbf{m}_M$. Else, if the dot product $\mathbf{m}_M \cdot \mathbf{n}_M = 0$, then $\exp M$ is a rotation of center $\mathbf{c}$, angle θ axis $\mathbf{v}$, as given by Equation (25):

$$\begin{aligned} \mathbf{c} &= \frac{\mathbf{!} \times \mathbf{m}}{\|\mathbf{!}\|^2} \\ \theta &= \|\mathbf{n}_M\| \\ \mathbf{v} &= \mathbf{n}_M / \theta \end{aligned} \tag{25}$$

Finally, in the remaining cases, we denote $l = \|\mathbf{n}_M\|$, and we use Equation (26):

$$\exp M = \mathrm{I} + M + \tfrac{1-\cos l}{l^2} M^2 + \tfrac{l - \sin l}{l^3} M^3 \tag{26}$$

Symmetrical objects can be easily modeled by introducing a plane of symmetry about which the tool is reflected (see Figure 21).

Efficiency: Applying the exponential of the matrix to a point does not require to compute the exponential of the matrix explicitely. Let us define the matrix M with a pair of vectors, $(\mathbf{n}, \mathbf{m})$.

$$\begin{aligned} \exp(M) \cdot \mathbf{p} &= \mathbf{p} + (\mathbf{m} + \mathbf{n} \times \mathbf{p}) b + (\tfrac{\mathbf{n} \times \mathbf{m}}{l^2} - \mathbf{p}) a \\ &\quad + \mathbf{n}((\mathbf{n} * \mathbf{p}) a + (\mathbf{n} * \mathbf{m})(1 - b)) \tfrac{1}{l^2} \\ \text{where } l &= \|\mathbf{n}\| \\ a &= 1 - \cos(l) \\ b &= \tfrac{\sin(l)}{l} \end{aligned} \tag{27}$$

For even faster performances, the exponential may be replaced with a first order approximation:

$$\exp(M) \cdot \mathbf{p} \approx (\mathrm{I} + M) \cdot \mathbf{p} = \mathbf{p} + \mathbf{n} \times \mathbf{p} + \mathbf{m} \tag{28}$$

7.4 Results

Figure 20 shows how much the deformations created by swirling sweepers give a clay-like behaviour to the scupted shape. In particular, the material seems to be swept along with the tool, as soft, real-clay would.

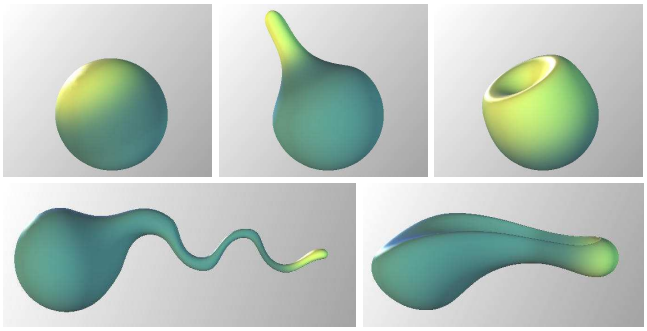


Figure 20: When pushed or pulled, a sphere will inflate or deflate elsewhere.

In Figure 21, 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. Note that 80 swirling-sweepers have been used to model the alien, thus the error of each steps is of no consequence to the user. The small errors are due to the finite number of steps, and to our choice of shape representation, and can be reduced by increasing the number of steps s and the number of samples on the deformed shape. The shapes shown in Figure 21 were modeled in real-time in *half an hour* at most, not including the design phase. They were all made starting with a sphere, thus all the feature were genuinely created with our method.

8 Discussion and conclusion

We have presented three different approaches towards virtual clay, all enabling shape creation and editing at interactive framerates. The first one is an extension of volumetric implicit sculpting with no limitation of the shape extent in space or resolution, thanks to the use of a virtual grid for storing the field samples. Local deformations mimicking a displacement of matter are added to enhance realism, but no mechanism is provided for applying more global deformations to the shape. Closer to real-clay, the second model incorporates physically-based deformations within the previous shape representation, yielding visually realistic deformations. At the other end of the spectrum, the last model incorporates a constraint from the physical world, constant volume deformations, within a geometric modeling framework. It provides a very practical system for intuitively deforming a shape when no change of topology needs to be applied.

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 and last sculpting systems are well suited to sculpting with a single tool (controlled for instance via a 3D mouse or an haptic device), sculpting a complex shape with the second model, much closer to real

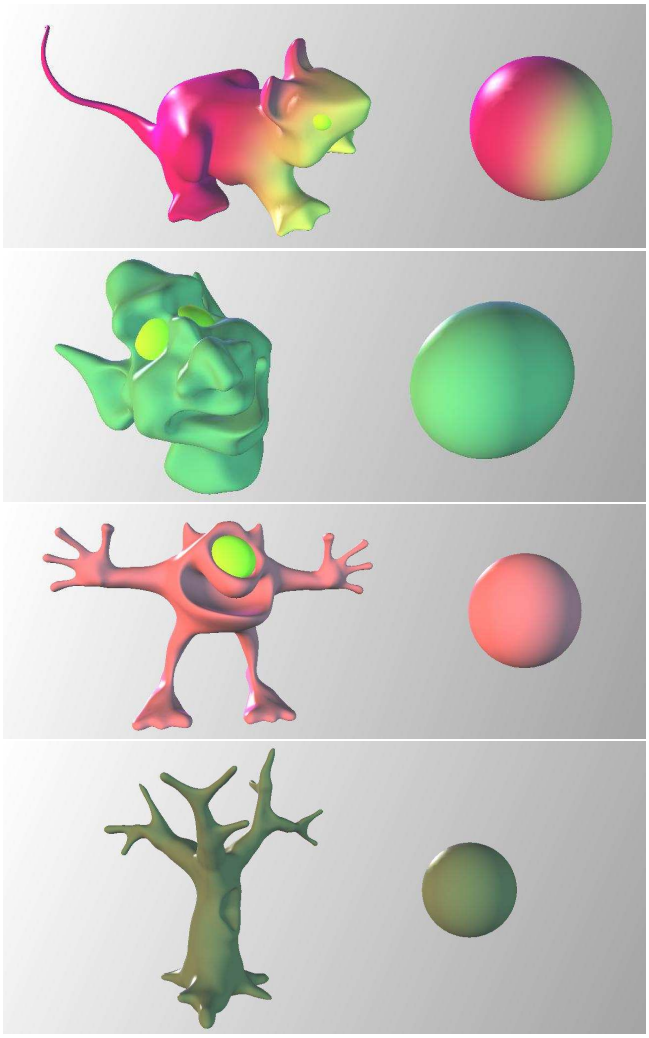


Figure 21: Examples of models modeled with swirling-sweepers. The mouse, the goblin, the alien and the tree have respectively 27607, 25509, 40495 and 38420 vertices. These objects were modeled in less than 30 min by one of the authors. Eyeballs have been added.

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. A mechanism for interactively controlling the motion of virtual hands is thus required.

Lastly, haptic interaction proved to be a great aid in a sculpting process. In the first system we presented, feeling the model 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 second 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 22). His hand gestures would be captured by cameras and the reconstructed gestures be used to deform the virtual

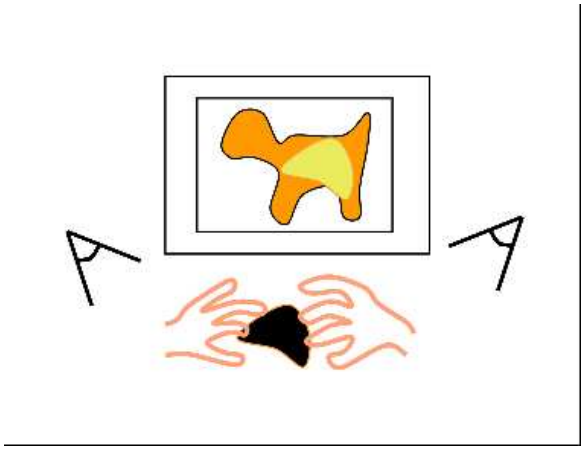


Figure 22: Our concept for direct hand manipulation for virtul clay.

sculpture. We believe that such an 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, Guillaume Dewaele, Geoff Wyvill and Scott King for their respective contributions to the different virtual clay models presented in this chapter.

References

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

Alexa, M. 2002. Linear combination of transformations. In *Proceedings of SIGGRAPH'02*, ACM Press / ACM SIGGRAPH, vol. 21(3) of *ACM Transactions on Graphics, Annual Conference Series*, ACM, 380–387.

Angelidis, A., Cani, M.-P., Wyvill, G., and King, S. 2004. Swirling-sweepers: Constant-volume modeling. In *Pacific Graphics 2004*, IEEE, 10–15. Best paper award.

Angelidis, A., Cani, M.-P., Wyvill, G., and King, S. 2004. Swirling-sweepers: Constant-volume modeling. In *Pacific Graphics 2004*, IEEE, 10–15. Best paper award at PG04.

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

Avila, R. S., and Sobierajski, L. M. 1996. A haptic interaction method for volume visualization. In *IEEE Visualization '96*, IEEE, 197–204.

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

AVILA, R. S. 1998. Volume haptics. *Computer Graphics*, 103–123. SIGGRAPH'98 Course Notes #01.

BÆRENTZEN, A. 1998. Octree-based volume sculpting. *Presented at IEEE Visualization '98*. `www.gk.dtu.dk/Andreas/publications.html`.

BLANCH, R., FERLEY, E., CANI, M.-P., AND GASCUEL, J.-D. 2004. Non-realistic haptic feedback for virtual sculpture. Tech. Rep. RR-5090, INRIA, U.R. Rhone-Alpes, january. Projet EVASION, theme 3.

BLOOMENTHAL, J., BAJAJ, C., BLINN, J., CANI, M.-P., ROCKWOOD, A., WYVILL, B., AND WYVILL, G. 1997. *Introduction to Implicit Surfaces*. Morgan Kaufman.

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

COLGATE, J. E., STANLEY, M. C., AND BROWN, J. M., 1995. Issues in the haptic display of tool use. IROS'95.

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

DEWAELE, G., AND CANI, M.-P. 2004. Virtual clay for direct hand manipulation. In *Eurographics '04 (short papers)*.

DRUON, S., A.CROSNIER, AND BRIGANDAT, L. 2003. Efficient cellular automata for 2d / 3d free-form modeling. *WSCG 11* (Feb.).

FERLEY, E., CANI, M.-P., AND GASCUEL, J.-D. 2000. Practical volumetric sculpting. *the Visual Computer 16*, 8 (dec), 469–480. A preliminary version of this paper appeared in Implicit Surfaces'99, Bordeaux, France, sept 1999.

FERLEY, E., CANI, M.-P., AND GASCUEL, J.-D. 2002. Resolution adaptive volume sculpting. *Graphical Models (GMOD) 63* (march), 459–478. Special Issue on Volume Modelling.

FRISKEN, S. F., PERRY, R. N., ROCKWOOD, A. P., AND JONES, T. R. 2000. Adaptively sampled distance fields: A general representation of shape for computer graphics. *Proceedings of SIGGRAPH 2000* (July), 249–254. ISBN 1-58113-208-5.

GALYEAN, T., AND HUGHES, J. 1991. Sculpting: An interactive volumetric modeling technique. *Computer Graphics 25*, 4 (July), 267–274. Proceedings of SIGGRAPH'91 (Las Vegas, Nevada, July 1991).

GILLESPIE, R. B., AND CUTKOSKY, M. R., 1996. Stable user-specific haptic rendering of the virtual wall. Proceedings of the 1996 ASME International Mechanical Engineering Congress and Exhibition, DSC-Vol. 58.

HUANG, C., QU, H., AND KAUFMAN, A. 1998. Volume rendering with haptic interaction. In *Proceedings of the Third PHANTOM Users Group Workshop*, vol. 3, 14–18.

MASSIE, T. H., AND SALISBURY, J. K., 1994. The phantom haptic interface : A device for probing virtual objects. Proceedings of ASME'94.

MCDONNELL, K., AND QIN, H. 2002. Dynamic sculpting and animation of free-form subdivision solids. *The Visual Computer 18*, 2, 81–96.

MCDONNELL, K. T., QIN, H., AND WLODARCZYK, R. A. 2001. Virtual clay: A real-time sculpting system with haptic toolkits. *2001 ACM Symposium on Interactive 3D Graphics* (March), 179–190. ISBN 1-58113-292-1.

NEALEN, A., MUELLER, M., KEISER, R., BOXERMAN, E., AND CARLSON, M. 2005. Physically-based deformable models in computer graphics. In *State of the Art Report*, Eurographics 2005, 71–94.

PERRY, R. N., AND FRISKEN, S. F. 2001. Kizamu: A system for sculpting digital characters. *Proceedings of SIGGRAPH 2001*, 47–56.

RAVIV, A., AND ELBER, G. 2000. Three-dimensional freeform sculpting via zero sets of scalar trivariate functions. *Computer-Aided Design 32*, 8-9 (August), 513–526. ISSN 0010-4485.

WANG, S. W., AND KAUFMAN, A. E. 1995. Volume sculpting. *1995 Symposium on Interactive 3D Graphics* (April), 151–156. ISBN 0-89791-736-7.

WYVILL, B., MCPHEETERS, C., AND WYVILL, G. 1986. Data structure for soft objects. *Visual Computer 4*, 2 (Aug.), 227–234.

MAKING DIGITAL SHAPES BY HAND

Steven Schkolne
steven@schkolne.com
California Institute of the Arts

SIGGRAPH Courses 2006
Interactive Shape Editing

INTRODUCTION

We do many things with our hands, but when it comes to making digital shapes, we tend to use the mouse and the keyboard. The hand is the human's most versatile means of acting on the material world, yet we use a small fraction of its potential when we model using conventional interfaces. This presentation explores interfaces which translate properties of the hand into operations on digital geometry. In particular we focus on looking beyond the mouse to alternate forms of user input.

What does the hand offer? One might answer this question from an emotional point of view Hands are part of the body, they create a heightened sense of connection with a design space when richly connected to its constituents. There is a sense within our culture that the manual tradition is important, and vanishing. Objects carry a particular value when they absorb human touch during formation, and reflect that human touch in their completed form.

Alternately, we can view the hand from a technical perspective. The hand's skeleton has about 27 continuous degrees of freedom, and placed at the end of an arm there is a huge parameter space which can be expressed by the human hand. Mouse input is a continuous stream of two-dimensional coordinates punctuated by discrete events (clicks, double-clicks, and the like). Is there not a better interface which would allow users to control more simultaneous degrees of freedom?

There is a certain integration of mind and body that takes place when we intentionally move our bodies. A professional baseball player can reliably hit a baseball moving at a high velocity. Computing the bat position and velocity, as any first-year physics student can attest, is a non-trivial set of calculations. The player does this in a split second. Current modeling interfaces break 3-dimensional shape modeling into a series of disjoint displacements that users compute using high-level cognition. Can we not use the motor cortex as a kind of parallel processor, to facilitate the creation of models?

The ultimate challenge in shape creation is conceptual — linking motion and model conception into a paradigm that facilitates the creation of sophisticated forms.

CONTROL MAPS: BETWEEN THE BODY AND THE MODEL

As Axel Mulder [Mulder 1998] notes in his dissertation on sound control, many simultaneously sensed parameters do not necessarily afford rich control. Mulder considers controlling sound with a CyberGlove. This glove detects 18 DOF of the human hand. Consider an interface where joint angles are mapped to properties of sound — the angle of the first finger joint is pitch, the thumb controls volume, etc., for several simultaneous sounds. While a performer technically has control of many parameters, there is a mental difficulty in realistically controlling them.

In digital interface design, input devices offer a general, abstract access to parameters. The relationship between an input device and a digital model is a *control map*. A control map (also known as transfer function) is a mathematical relationship established between body/input device position and model state.

We will consider three classes of interface, each with a distinct flavor of control map.

MATERIAL — The model consists of atomic units which are arranged and manipulated by the user. Input directly manipulates this structure. The virtual clay interface presented by Marie-Paule Cani, earlier in this course, is an example. In this interface, a 2-dimensional mouse controls a sculpting tool which, on a monitor, affects the shape of a 3-dimensional virtual clay model. In material interfaces, there is a continuous map between motion of the input device and deformations of the model.

Many systems implemented in the virtual world are also material [Deering 1995, Keefe 2001]. In these interfaces, 3-dimensional direct manipulation creates shapes. Shapes made of discrete entities can also be made in this manner. Block-based systems (such as a child's toy blocks, or [Frazer 1980]) present a number of physical objects which are manually arranged to make a form.

MARK INTERPRETATION — Continuous paths in two or three dimensions are mapped to 3-dimensional geometry. These paths either add to, or modify an existing shape. In Takeo Igarashi's Teddy interface, also presented earlier in the course, strokes are used as input to an algorithm which produces 3-dimensional form.

In these methods, hand motions are captured, but do not directly affect form. Hand motion creates strokes which are inputs to functions which produce 3-dimensional geometry. The effectiveness of the control map is highly dependent on the design of these functions.

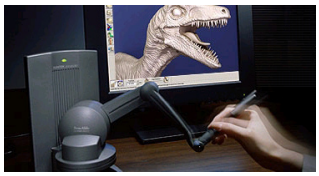
THINKING THROUGH STRUCTURE — In these interfaces, parameters of an underlying data structure are exposed to the user. The user edits parameters (often continuously), and views the resulting change in the data structure. In the subdivision modeling described by Denis Zorin, users pull control vertices. These modifications of single points cause a surface to deform. Before the information age, engineers used technical diagrams to numerically describe processes which, after manufacture, resulted in a three-dimensional form.

MATERIAL METAPHORS

We will consider material interfaces based on input devices ranging from mice to cameras. The mouse-driven ZBrush toolsuite allows modelers to add details to subdivision surfaces in a material manner. Dragging the mouse cursor across an area causes a region to be deformed along the surface normal (or other vector). By changing the method of calculating the deformation vector, users can tune the creation of detail.

The grooves in this character's skin were formed by ZBrush

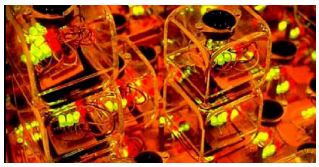
The Free-Form modeler, which utilizes the Phantom haptic device, adds a sense of touch to this process. The user is able to specify the normal vector of the deformation interactively by rotating the phantom around its pentip. The user can also more accurately depress the form due to the force feedback. This is especially useful for creating features which trace surface curvature lines.



The phantom input device controls the deformation of the onscreen model. Through haptic feedback, the user can feel the surface as it is deformed.

Note the phantom keeps the pen in front of a flat screen. There is a mapping from the 3-dimensional input space to the 2-dimensional display surface across some distance. While the input control map is between 3-dimensional spaces, and the haptic display is 3d, the visual display is lower-dimensional and across some distance. Thus it is not quite as immediate as the physical carving of clay where the mapping to the surface is direct.

While Free-Form offers haptic feedback & 6-DOF deformation, it does not allow users to fully grasp objects. Haptic devices that control the human skeleton to such a degree that grasping can be enforced are (sadly, but inevitably) bulky and awkward. There is another paradigm, one where instead of simulating material we use the material properties that already exist in nature.

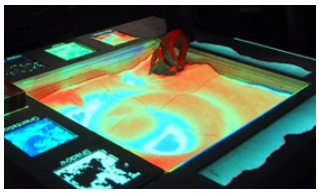


These sensed blocks are manipulated in Frazer's interface.

Sensed material interfaces detect physical objects and use this information to update a digital structure. The earliest known work is that of John Frazer et al., which was realized in the early 1980's [Frazer 1980, Frazer 1994]. These innovative physical/digital interfaces comprised of an array of cubic blocks. Each cube was the same size, and wired to a computer that detected the spatial configuration of the blocks (based on block topology). Frazer came from the architectural tradition, and was interested in designing tools to allow the quick evaluation of building programs. In some implementations, computations occurring on the CPU, such as the results of thermal analysis, were displayed on a screen as the user interacted.

One limitation of this approach is that the blocks themselves are not visually sophisticated. Anderson et al. developed a system [Anderson 2000] that displays a stylized rendition of the physical model on a 2d display. This sensed-block approach cannot create detailed geometry, as the resolution of the form is bounded by the resolution of the building blocks.

The SandScape system [Piper 2002] uses particles of sand as its building block. The user creates a form in a small sand box. A camera detects the shape, while a projector displays information on the sand surface. This increases model resolution, although users are limited to height fields, such as landscape models.



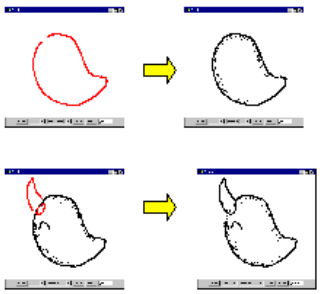
A user manipulates a landscape visualization with the SandScape interface.

These physical, tangible interfaces have many advantages. They are bimanual and multiuser, and they mimic the physical world, so interaction is immediate. The hand manipulates many degrees of freedom simultaneously. However, they are limited in the types of models they can create. As we will see elsewhere in this presentation, there is typically a trade-off between the simplicity of a metaphor and the sophistication of the results it produces.

MARK-INTERPRETATION

Tracing interfaces, based on the act of making a stroke, have their roots in traditional drawing. Computers interpret these strokes in a variety of manners to create shapes.

The Teddy interface [Igarashi 1999], described earlier in this course, creates 3-dimensional information from 2-dimensional strokes.



The Teddy system converts strokes, shown in red, into 3d geometry.

In Teddy, the control map is not a mathematical surface representation but a set of algorithms which takes 2d strokes as input. It is very challenging to make these algorithms so clear that a child can understand them (which is the case with Teddy).

Systems which interpret marks are a rare augmentation of material with algorithmic properties. The current challenge with this style of interface is to make the algorithms capable of very precise, expert-class surface modifications.

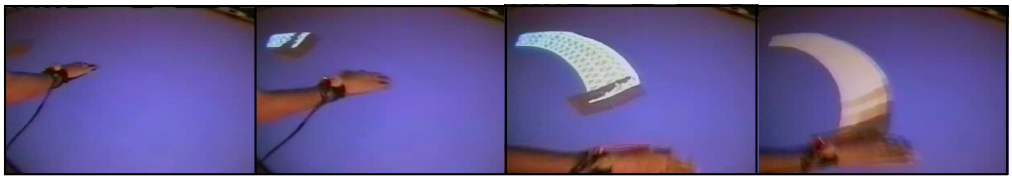
SURFACE DRAWING

As a case study we look at the surface drawing system, which I built with Michael Pruett and Peter Schröder [Schkolne 1999, 2001, 2003]. In this system, which utilizes 6-DOF electromagnetic trackers and a stereoscopic display, hand motions create 3-dimensional shapes in free space. This direct creation is material, although the strokes are merged to one another to form a coherent surface. This last step of mark interpretation makes it easier for users to build coherent surfaces.

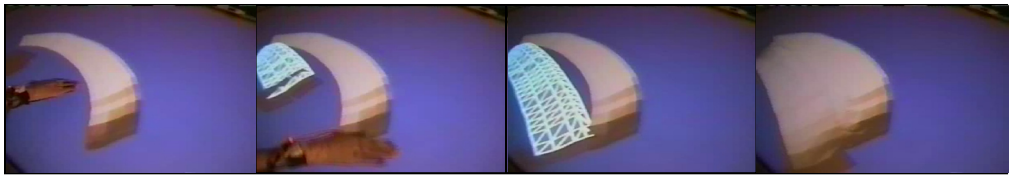
The path of the hand in space is rendered as a geometric object. The curvature of the hand defines the curvature of the stroke: a large amount of data is specified in each sweep of the hand.

Surface drawing is very hand-centric. The produced marks directly reflect the curvature of the hand as it bends. The resulting shapes reflect this organic nature. This interface also has limitations, and by studying it we can understand the shortcomings of transparent (so-called 'natural') control maps.

The images below show an implementation of this interaction. The second row shows strokes merging together to form a larger continuous surface due to a stroke-merging algorithm.

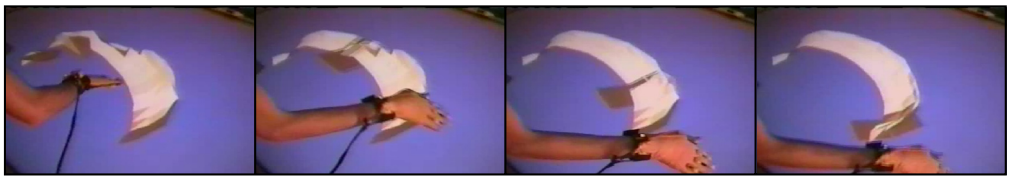


The hand paints a stroke in space.

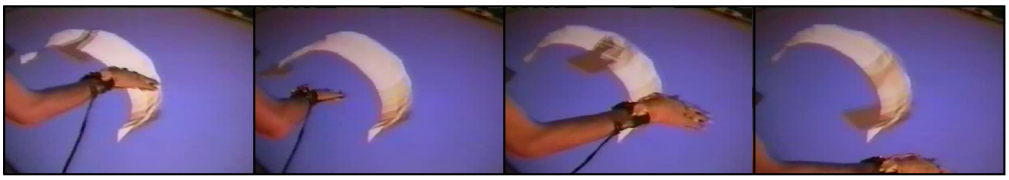


Two strokes automatically merge to make a continuous surface.

The tangent plane can be used as a variable to control a smoothing operation (where the surface moves closer to a flat surface whose normal is dictated by the hand's normal) and a deform operation (where the surface moves in the direction of the hand's normal).



Smooth: A surface is polished by rubbing the hand over it.



Deform: The surface is slightly altered by rubbing the hand over it.

These three-dimensional digital tongs contain a 6-DOF sensor in their base. A contact switch detects when they are closed. These tongs are used to move and stretch shapes in the surface drawing system.

Objects are moved with sensed tongs. The act of closing the tongs is a natural signal to begin moving a virtual object with the tongs themselves. Two sets of tongs used together stretch an object, increasing or decreasing its scale.

RESULTS

The shapes that can be produced with the system have an intrinsic roughness to them. This is a reflection of the wavering of the body, combined with sampling error introduced by sensors. Note how well the organic qualities of the leaves and petals below are modeled by this shakiness. While the final roughness can be removed by known methods [Desbrun 1999], it is more difficult to add such a meaningful roughness to surfaces. Although this process is non-haptic, the element of touch has significance. While not volumetric, surface drawing has many of the tactile elements of clay sculpture.

A variety of shapes created with the surface drawing system.

The two versions of a head model (above, at right) show the effects of the smoothing operation. Shown at left is an early model of the head. At right, the smooth/deform tool has been used to correct the proportions of the face and smooth the head's surface.

There is a level of spatial understanding which is unique to 3-dimensional interfaces. The images below are of the same shape. It is difficult to see this as one shape through 2d views. Working directly in three dimensions allows one to create highly artificial structures that are not easily made with 2d tools.

Three views of an interwoven form

one-minute gesture drawings

The gesture drawings shown above were created by an experienced user. Even for a user new to 3-dimensional interfaces, it is easy to produce a quick gestural rendering of a person.

An artist created this shape in 20 minutes, after 10 minutes of training.

Users responded enthusiastically to this system, making assessments such as the following:

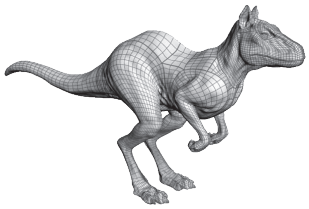
> *I was completely amazed at how quickly I interpreted and understood the canvas and model to be existing in space. It was immediate.*
>
> *To see abstract images pour like water from my fingertips is sensational... Even more amazing is to see what touch looks like!*

We can clearly see the emotional connection that such an interface enables. However, we must note the difference between enthusiasm and incorporation of a method into professional practice. It is one thing to feel connected to a form, and another to actually have a fine degree of control over a surface. Interfaces such as surface drawing are rich examples of bodily interfaces, yet they miss something crucial: the data-centric, structural view of digital models. The optimal modeling interfaces mixes the physical (motor cortex) thought of surface drawing with the data-driven (frontal lobe) analysis required by interfaces that expose their data structures more directly.

THINKING THROUGH STRUCTURE

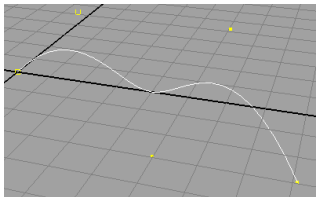
The key component that is missing from both the material and mark-interpreting interfaces presented above is a close relationship between the user and the underlying data structure. The majority of these works aim to liberate users from having to worry about data structures. However, these material interfaces are not very popular commercially. Even the methods that do not require specialized hardware have failed to become standards for digital modeling. Why is this the case?

I believe the root of this to be the utility of data structures. There is a level of control that can be achieved by being close to the data. In the most useful systems, users actually think in terms of a data structure that they are presented with. This is as unintuitive as it is powerful. This type of control is rarely afforded by material input mappings. In structural interfaces, networks of control points, curves, and interpolating surfaces/volumes are used to control objects. Users will put in a large amount of work for a high quality result, even if the interface is unpleasant



A subdivision model is controlled by placing vertices in a control mesh which is subdivided to produce a continuous surface.

and counter-intuitive. Traditional Computer-Aided Design (CAD) has long been criticized as unintuitive. In many of these systems, users place points on screen space planes which weight basis functions to form a 3d object. For example, the curve below is created by five coplanar control points which are interpolated by a spline. In typical CAD interfaces there are several steps to create curves, as users must specify three dimensions with a 2d input device.



A curve and its control points

Perhaps the first system to allow the direct input of 3d spatial data, 3-Draw [Sachs 1991] tries to make the process more straightforward by using a 3d stylus to place control points in space. 3-Draw can build points with constraints and interpolating curves. While it is similar to other stylus-based systems (such as HoloSketch [Deering 1995]) in outward appearance, its control map is fundamentally different. HoloSketch is a material interface where the path of the stylus creates strokes directly in 3-dimensional space. There is a continuous relationship between hand and form. In 3-Draw, users place points, and the form is decided by spline basis functions. More recently, Wesche and Seidel [Wesche 2001] have presented further developments 3d interfaces for the creation of spline surfaces.

The 2d metaphors required by mouse-based CAD applications are sometimes an advantage, not a limitation. Planes are essential constructs in engineering applications where the tools used for manufacture work in terms of planes and sweep directions. However, for freeform modeling this is more of a limitation than a feature.

CONCLUSION

We have covered a variety of input devices and output scenarios, in the hopes that the reader (and the author!) will begin to understand the control maps that tie the input to the output. Between mind and model, the hand plays a crucial role in delivering input to this control map. We have seen how some maps provide a level of direct control. These come closest to replicating the sense of touch. While haptics is a nice addition, it is the link between hand motion and model deformation which most strongly establishes this connection.

Other interfaces place a level of interpretation between user action and the resulting change in the model. In these interfaces, the challenge is to allow the hand to most readily manipulate and/or create elements (such as strokes or control vertices) of geometric data structures. A flexible, versatile, intuitive structure that can be richly manipulated by the hand has yet to be discovered. Currently, we see either rich structures which can only be manipulated awkwardly, or overly simple structures which can be richly manipulated. Satisfying both goals simultaneously is a major challenge for modeling interface research.

REFERENCES

[ANDERSON 2000] David Anderson, James L. Frankel, Joe Marks, Aseem Agarwala, Paul Beardsley, Jessica Hodgins, Darren Leigh, Kathy Ryall, Eddie Sullivan, and Jonathan S. Yedidia, Tangible Interaction + Graphical Interpretation: A New Approach to 3D Modeling, Proceedings of SIGGRAPH 2000, 393-402.

[ANGUS 1995] I. G. Angus and H. A. Sowizral, Embedding the 2D Interaction Metaphor in a Real 3D Virtual Environment, SPIE Proceedings Vol 2409: Proceedings of Stereoscopic Displays and Virtual Reality Systems, 282-293, 1995.

[BALAKRISHNAN 1999] Ravin Balakrishnan and Ken Hinckley, The Role of Kinesthetic Reference Frames in Two-Handed Input Performance, Proceedings of UIST 1999, 171-178.

[BRODY 1999] Bill Brody and Chris Hartman, BLUI, a Body Language User Interface for 3d Gestural Drawing, SPIE Proceedings Vol 3644: Human Vision and Electronic Imaging IV, 1999.

[CRUZ-NEIRA 1992] Carolina Cruz-Neira, Daniel J. Sandin, Thomas A. DeFanti, Robert V. Kenyon, and John C. Hart, The CAVE: Audio Visual Experience Automatic Virtual Environment. Communications of the ACM, 35:6, 67-62, June 1992.

[DEERING 1995] Michael F. Deering, HoloSketch: A Virtual Reality Sketching/Animation Tool, ACM Transactions on Computer-Human Interaction, 2:3, 220-238, September 1995.

[DESBRUN 1999] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr, Implicit Fairing of Irregular Meshes Using Diffusion and Curvature Flow, Proceedings of SIGGRAPH 1999, 317-324.

[DEWAELE 2004] Guillaume Dewaele and Marie-Paule Cani, Virtual clay for direct hand manipulation. In Eurographics ‘04 (short papers).

[FEINER 1993] Steven Feiner, Blaire MacIntyre, Marcus Haupt, and Eliot Solomon, Windows on the World: 2D Windows for 3D Augmented Reality, Proceedings of UIST 1993, 145-155.

[FITZMAURICE 1996] George W. Fitzmaurice, Graspable User Interfaces, PhD Thesis, University of Toronto, 1996.

[FRAZER 1980] J. H. Frazer, J. M. Frazer, and P. A. Frazer. Intelligent Physical Three-Dimensional Modelling System, Proceedings of Computer Graphics 80, 359–370.

[FRAZER 1994] J. H. Frazer, An Evolutionary Architecture, Architectural Association, London, 1994.

[FREEFORM] FreeForm, software application, http://sensable.com.

[FRÖHLICH 2000] Bernd Fröhlich and John Plate, The Cubic Mouse: A New Device for Three-Dimensional Input. Proceedings of CHI 2000, 526-521.

[GALYEAN 1991] Tinsley A. Galyean and John F. Hughes. Sculpting: An Interactive Volumetric Modeling Technique, Proceedings of SIGGRAPH 1991, 267-274.

[GORBET 1998] Matthew G. Gorbet, Maggie Orth, and Hiroshi Ishii. Triangles: Tangible Interface for Manipulation and Exploration of Digital Information Topography, Proceedings of CHI 1998, 49–56.

[GUIARD 1987] Yves Guiard, Asymmetric Division of Labor in Human Skilled Bimanual Action: The Kinematic Chain as a Model, Journal of Motor Behavior, 19:4, 486-517, 1987.

[HINCKLEY 1994] Ken Hinckley, Randy Pausch, John C Goble, and Neal F Kassell, Passive Real-World Interface Props for Neurosurgical Visualization. Proceedings of CHI 1994, 452-458.

[IGARASHI 1999] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka, Teddy: A Sketching Interface for 3D Freeform Design, Proceedings of SIGGRAPH 1999, 409-416.

[JACOB 1994] Robert J. K. Jacob, Linda E. Sibert, Daniel C. McFarlane, and M. Preston Mullen, Jr., Integrality and Separability of Input Devices, ACM Transactions on Computer-Human Interaction, 1:1, 3-26, 1994.

[KEEFE 2001] Dan F. Keefe, Daniel Acevedo Feliz, Tomer Moscovich, David H. Laidlaw, and Joseph J. LaViola, Jr., CavePainting: A Fully Immersive 3D Artistic Medium and Interactive Experience, Proceedings of the 2001 Symposium on Interactive 3D Graphics, 85-93.

[KRUEGER 1983] Myron Krueger, Artificial Reality, Addison-Wesley, 1983.

[KRÜGER 1994] Wolfgang Krüger and Bernd Fröhlich. The Responsive Workbench, IEEE Computer Graphics and Applications, 12-15, May 1994.

[LLAMAS 2003] Ignacio Llamas, Byungmoon Kim, Joshua Gargus, Jarek Rossignac and Chris D. Shaw, Twister: A Space-Warp Operator for the Two-Handed Editing of 3D Shapes, Proceedings of Siggraph 2003.

[LIANG 1993] Jiandong Liang and Mark Green, JDCAD: A Highly Interactive 3D Modeling System, 3rd International Conference on CAD and Computer Graphics, Beijing, China, 217-222, August 1993.

[MCDONNELL 2001] Kevin T. McDonnel, Hong Qin, and Robert A. Wlodarczyk, Virtual Clay: A Real-Time Sculpting System with Haptic Toolkits, 2001 Symposium on Interactive 3D Graphics, 179-190.

[MINE 1997] Mark Mine, Frederick Brooks, Carlo Sequin, Moving Objects in Space: Exploiting Proprioception In Virtual-Environment Interaction, Proceedings of SIGGRAPH 1997.

[MULDER 1998] Axel Mulder, Design of Virtual Three-Dimensional Instruments for Sound Control. PhD Thesis, Simon Fraser University. 1998.

[PIERCE 1997] Jeffrey S. Pierce, Andrew Forsberg, Matthew Conway, Seung Hong, and Robert Zeleznik, Image Plane Interaction Techniques in 3D Immersive Environments., 1997 Symposium on Interactive 3D Graphics, 39-43.

[PIERCE 1999] Jeffrey S. Pierce, Brian C. Stearns, amd Randy Pausch, Voodoo Dolls: Seamless Interaction at Multiple Scales in Virtual Environments, 1999 Symposium on Interactive 3D Graphics, 141-145.

[PIPER 2002] Ben Piper, Carlo Ratti, and Hiroshi Ishii, Illuminating Clay: A 3-D Tangible Interface for Landscape Analysis, Proceedings of CHI 2002, 355-362.

[POUPYREV 1997] Ivan Poupyrev, Mark Billinghurst, Suzanne Weghorst, and Tadao Ichikawa. Go-Go Interaction Technique: Non-Linear Mapping for Direct Manipulation in VR, Proceedings of UIST 1996, 79-80, 1996.

[SACHS 1991] Emanuel Sachs, Andrew Roberts, David Stoops, 3-Draw: A Tool for Designing 3D Shapes, IEEE Computer Graphics & Applications, Nov. 1991, 18-26.

[SCHKOLNE 1999] Steven Schkolne, Surface Drawing: The Perceptual Construction of Aesthetic Form, M.S. Thesis, Caltech, 1999.

[SCHKOLNE 2001] Steven Schkolne, Michael Pruett, and Peter Schröder, Surface Drawing: Creating Organic 3d Shapes with the Hand and Tangible Tools, Proceedings of CHI 2001, 261-268.

[SCHKOLNE 2003] Steven Schkolne, 3d Interfaces for Spatial Construction, PhD thesis, Caltech, 2003.

[SHNEIDERMAN 1983] Ben Shneiderman, Direct Manipulation: A Step Beyond Programming Languages, IEEE Computer, 16:8, 57–69, August 1983.

[SEDERBERG 1986] T. W. Sederberg and S. R Parry, Free-Form Deformation of Solid Geometric Models, Proceedings of SIGGRAPH 1986, 151-160.

[SHAW 1994] Chris Shaw and Mark Green, Two-Handed Polygonal Surface Design, Proceedings of UIST 1994, 205-212.

[ULLMER 1997] Brygg Ullmer, and Hiroshi Ishii, The metaDESK: Models and Prototypes for Tangible User Interfaces, Proceedings of UIST 1997, 223-232.

[UNDERKOFFLER 1999] John Underkoffler, Brygg Ullmer, and Hiroshi Ishii, Emancipated Pixels: Real-World Graphics in the Luminous Room, Proceedings of SIGGRAPH 1999, 385-392.

[WESCHE 2001] Gerold Wesche and Hans-Peter Seidel, FreeDrawer: A Free-Form Sketching System on the Responsive Workbench, Proceedings of the 2001 ACM Symposium on Virtual Reality Software and Technology, 167-174.

[WESCHE 2003] Gerold Wesche, The ToolFinger: Supporting Complex Direct Manipulation in Virtual Environments, Proceedings of the ACM/Eurographics Workshop on Virtual environments 2003, 39-45.

[ZBRUSH] ZBrush, software application, http://pixologic.com.

[ZELEZNIK 1996] Robert C. Zeleznik, Kenneth P. Herndon, and John F. Hughes, SKETCH: An Interface for Sketching 3D Scenes, Proceedings of SIGGRAPH 1996, 163–170.