International Journal of Shape Modeling

INTERACTIVE MODELLING OF SHAPES USING THE LEVEL–SET METHOD

J. ANDREAS BÆRENTZEN
Informatics and Mathematical Modelling, Technical University of Denmark, Building 321 Lyngby, DK–2800, Denmark

and

NIELS JØRGEN CHRISTENSEN
Informatics and Mathematical Modelling, Technical University of Denmark, Building 321 Lyngby, DK–2800, Denmark

Received (received date)
Revised (revised date)
Communicated by (Name of Editor)

In this paper, we propose a technique for intuitive, interactive modelling of 3D shapes. The technique is based on the Level–Set Method which has the virtue of easily handling changes to the topology of the represented solid. Furthermore, this method also leads to sculpting operations that are very simple and intuitive from a user perspective. A final virtue is that the LSM makes it easy to maintain a distance field representation of the represented solid. This has a number of benefits such as simplification of the rendering scheme and the curvature computation. A number of LSM speed functions which are suitable for shape modelling are proposed. However, normally these would result in tools that would affect the entire model. To facilitate local changes to the model, we introduce a windowing scheme which constrains the LSM to affect only a small part of the model. The LSM based sculpting tools have been incorporated in our sculpting system which also includes facilities for volumetric CSG and several techniques for visualization.

Keywords: Level–Set Method, Shape, Sculpting, Volume, Distance Field

1. Introduction

Interactive modelling of 3D shapes on a computer should be as simple and intuitive as doodling 2D shapes using pencil and paper. Simpler, in fact, since on a computer changes can always be undone, and the user is more free to explore and experiment.

Unfortunately, completely intuitive, interactive modelling of shapes seems to be an elusive goal. Some authors attack the problem of intuitive sculpting from a user interface perspective. Perhaps the best example is the well–known gesture–based system, Teddy, by Takeo Igarashi [21]. It seems likely that by a careful design of user friendly interfaces, we can come a long way towards intuitive sculpting, but, unfortunately, it also seems that the underlying representation will be an obstacle

– simply because effective modelling often requires the user to be aware of the parameters of the representation.

For instance, subdivision surfaces require the user to pay attention to the valency of vertices and the placement of extraordinary vertices. In a similar vein, modelling with implicit surfaces requires the user to be aware of the (often non–intuitive) parameters controlling the shape of the implicit. Supposedly, the user (or sculptor) thinks in terms of real world manipulations such as "squeezing", "bending", "pushing", and having to manually translate this into e.g. changes to control points or grid connectivity is bothersome. In other words, the desirable goal is to completely decouple the user interface from the underlying representation.

When it comes to implicit surfaces, some efforts have been made in this direction. Witkin and Heckbert proposed a technique whereby the user can indirectly control an implicit surface by moving control particles distributed on the surface [41]. This work has very recently been extended by Turk and O'Brien who use interpolating implicit surfaces as the underlying implicit representation [37].

While this work seems promising, we are considering a different approach based on the volumetric representation which does allow the desired decoupling: When a shape is represented by a volume (i.e. a regular 3D grid of voxels), it is feasible to create shape manipulation tools where the user only specifies what to do (e.g. make a dent, create a protrusion, make something smoother) without any direct control over voxel values. A number of techniques have been used for sculpting systems based on the volume representation. In this paper, we propose the use of the Level-Set Method (LSM). The LSM is a promising choice because it allows for generic deformations. The LSM is usually a global method, but we have adapted it for local manipulations and defined LSM based tools for sculpting. These tools have been implemented in our volume sculpting system which also supports volumetric CSG operations and several techniques for visualization.

In the next section, we discuss related work and motivate our departure from previous volume based methods. In Section 3 we discuss the Level–Set Method and our particular implementation. In Section 4 our interactive sculpting system is presented. Results are found in Section 5, and finally we draw conclusions and point to future work in Section 6.

2. Related Work

The past ten years have seen a number of publications pertaining to volume sculpting [3, 16, 31, 12, 39, 32, 7, 2, 30, 15, 4] as well as a commercial system from SensAble Technologies. The proposals are diverse, and a number of the systems support advanced 3D input and output facilities. However, the systems are similar with respect to the tools they support. This is true at least if we focus on the systems based on the grey–level volume representation (see next section). In this case all manipulations are block manipulations where a region of the volume is traversed and some operation performed on each voxel therein. This mode of

operation has some drawbacks. In particular, it does not lead to generic technique for deformations, and it is impossible to assign a precise significance to a voxel.

As a remedy, we propose to use the Level–Set Method as the basic technology of a sculpting system. This approach lends itself well to any sort of deformative manipulation of volumetric solids, and it maintains a "cleaner" volume representation where voxels have (and retain) the property that their value is the signed shortest distance to the boundary of the represented solid.

Recently, Museth et al. [22] proposed a surface editing system also based on the Level–Set Method. Their (independent) work is similar to ours in this respect, but differs in aim since they focus on editing existing shapes whereas we are concerned with free–form sculpting.

2.1. *Background and Motivation*

Existing volume sculpting systems can, roughly, be divided into three categories: Systems that employ the *binary* volume representation, e.g. [31, 12, 32], and systems which employ the grey–level or *scalar* volume representation [3, 16, 39, 7, 30, 15]. In addition a number of systems are related to volume sculpting systems, but differ in significant ways: For instance, some authors have investigated *Adaptive Distance Fields* [18, 29] or volumes where the voxels are linked [17].

In this paper, we focus on the systems based on the scalar volume representation, since the binary representation does not lend itself well to the sculpting of solids with smooth surfaces. The alternative approaches solve certain problems, but they also introduce new difficulties. For instance, Adaptive Distance Fields allow for higher resolution features, but seem to be suitable only for volumetric CSG and not manipulations that deform the solid. The linked volume representation is an augmented binary volume representation, and like binary volumes probably not suitable for the sculpting of solids with smooth surfaces.

In the case of the scalar volume representation, it is generally assumed that voxels are placed at the points of an isotropic 3D lattice. The distance between two adjacent voxels (one voxel unit – *vu*) is often a convenient unit. A scalar value is associated with each voxel. We can see this value as a sample of a *V–model* [38] also called a *characteristic function*. A V–model is, essentially, an *implicit surface* representation of a solid. More precisely, given a solid S, an associated V–model

$$\mathcal{V}(S) : \mathbb{R}^3 \rightarrow [a, b] \subset \mathbb{R}$$

should have the property that $\mathcal{V}(S)(\mathbf{p}) > \tau$ if $\mathbf{p}$ is outside the solid, $\mathcal{V}(S)(\mathbf{p}) = \tau$ if $\mathbf{p} \in \partial\mathbf{S}$ and $\mathbf{p} < \tau$ if $\mathbf{p}$ is inside the solid. The *iso–value* τ is arbitrary, and in the following we always assume that $\tau = 0$. In the context of scalar volumes, the process of sampling a V–model is called *voxelization*. In other words, in the context of scalar volume, voxelization denotes the conversion from some representation to the volumetric by way of a V–model representation.

It is known that $\mathcal{V}(S)$ should be smooth and vary slowly with respect to the

voxel grid. It would seem logical to use a V–model that jumps from, say, zero to one on the boundary of a solid. However, such a function is, of course, rich in high frequency components and when it is sampled (at the voxel positions), the reconstruction (the value is reconstructed using interpolation between voxel values) will exhibit artifacts. See also [19, 26, 9].

In the following, we will assume that the V–model is simply the signed shortest distance to the boundary of the solid clamped to a certain range, i.e.

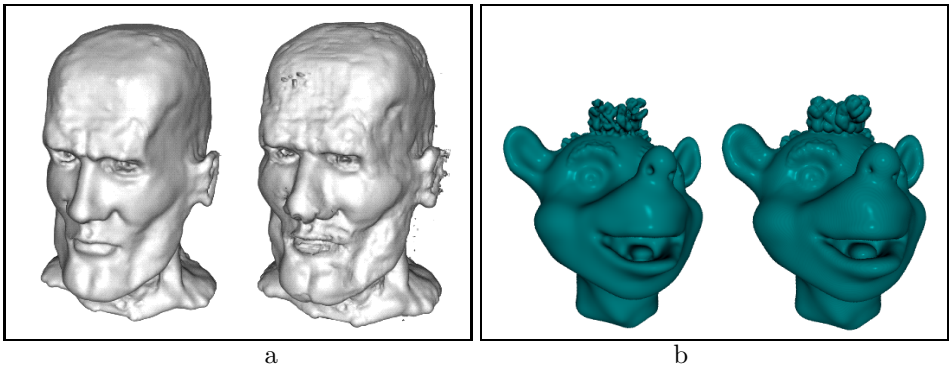
$$\mathcal{V}(S)(\mathbf{p}) = \min(\max(-\mathbf{r}, \mathbf{d_S}(\mathbf{p})), \mathbf{r}) \tag{1}$$

where r is the width of the *transition region* and

$$d_S(\mathbf{p}) = \begin{cases} -\inf_{\forall q \in \partial S} ||\mathbf{p} - \mathbf{q}|| & \mathbf{p} \in \mathbf{S} \\ \inf_{\forall q \in \partial S} ||\mathbf{p} - \mathbf{q}|| & \mathbf{p} \notin \mathbf{S} \end{cases} \tag{2}$$

Voxels in the transition region are called *transition voxels*, and voxels outside the transition region are called interior or exterior depending on their sign. To avoid artifacts in reconstruction, it is best if r is about 2.5 *vu* or larger [9, 38]. This value has been used in the work presented here.

The scalar volume representation will be called the *distance field volume* representation (DFV) when voxels are sampled from a function of the type (1). Distance field volumes hold a number of advantages. First of all, finding the distance to a solid is a common operation in computer graphics. Hence DFVs can be generated using technologies also used for e.g. collision detection. Secondly, the value of a transition voxel now has a clear geometric significance. Finally, certain operations are simplified. In this paper, we shall see that it is easier to compute curvature and find points on the boundary of a solid if the volume is a DFV than it is in general for scalar volumes.



a b

Fig. 1. a: Model created using old sculpting system. b: New system.

Sculpting systems based on the scalar volume representation are generally similar in the way manipulations work (a notable exception being the method proposed

by Arata et al. in [2] where voxels represent cellular automata that exchange material.). The user positions a tool somewhere inside the volume, and the tool affects a box shaped *Region of Influence* (ROI). For each voxel of value v and position $\mathbf{p}$ in the ROI, a simple operation is carried out. Typically either

1. v is replaced by a weighted average of v and the values of neighboring voxels.

2. v is replaced by a combination of v and the value of a V–model evaluated at the voxel position. $v \leftarrow g(v, \mathcal{V}_{\text{tool}}(\mathbf{p}))$ where $\mathcal{V}_{\text{tool}}$ is the V–model of a tool.

1. is the simplest to explain. The averaging corresponds to convolving the volume with a blurring kernel, and the result is that the represented solid becomes smoother. 2. is really volumetric CSG. Many implementations are possible. If the volume is a DFV, $g(a, b) = \min(a, b)$ is sometimes used since for most (but not all!) voxels the correct signed shortest distance is the minimum of the shortest distance [8, 27]. However, many other per–voxel operations have been proposed.

The motivation for the work presented here is twofold. First of all, the two operations above allow for smoothing and volumetric CSG but do not provide a general method for deforming volumetric solids. Secondly, the above method does not preserve the distance field representation. This easily leads to noise at other iso–values than $\tau = 0$ as illustrated in Figure 1. The model shown in Figure 1a is sculpted using the system discussed in [7] whereas the methods presented in this paper are used for Figure 1b. In both figures, the left hand image shows $\tau = 0$ whereas the right hand image shows a value of τ near the maximum. Notice that the right hand picture in Figure 1a exhibits considerable noise.

In the following, we will discuss the Level–Set Method which has been adapted to volume sculpting. Using this method it is possible to perform more general deformations. Moreover, the Level–Set Method preserves the distance field representation. In practical terms, we rebuild the transition region for each manipulation to ensure that voxel values correspond to distances with reasonable precision.

The Level–Set Method is a flexible tool which has found diverse applications. In the context of volume graphics, LSM has recently been used for segmentation problems [40], and the metamorphosis of volumetric solids [6].

3. The Level–Set Method

The Level–Set method [34] is a technique for tracking the evolution of a deforming interface or surface. The aim of this section is to inform the reader about how it works and how it is implemented. For the finer details on e.g. upwinding, stability and convergence, the reader is referred to [34, 24]. Assume that we are dealing with a surface $X(t) \subset \mathbb{R}^3$ where t is the time parameterization. X is assumed to change according to some *speed function* that pushes X in the normal direction.

The motion of X is expressed through a relationship with an embedding function $\Phi : R^3 \times R^+ \rightarrow R$. For all points on X the value of Φ must be zero. This leads to

the equation

$$\Phi(X(t), t) = 0 \tag{3}$$

where $X(t)$ denotes the set of points belonging to X at time t. (3) simply says that $X(t)$ is an isosurface (here called a level–set) of $\Phi(\cdot, t)$. Because this holds for any point in time, both X and Φ may evolve but the Level–Set equation continues to hold implying that

$$d\Phi(X(t), t)/dt = 0 \tag{4}$$

To see how the change of Φ and X are coupled, we compute the derivative using the chain rule

$$d\Phi(X(t), t)/dt = \frac{\partial \Phi}{\partial t} + \nabla\Phi \cdot \frac{dX}{dt} \tag{5}$$

where $\nabla\Phi = \left[\frac{\partial\Phi}{\partial x}\frac{\partial\Phi}{\partial y}\frac{\partial\Phi}{\partial z}\right]$. Because all motion is in the normal direction, we can write the change of X in terms of a speed function F times the normal $\frac{\nabla\Phi}{||\nabla\Phi||}$

$$\frac{dX(t)}{dt} = F\frac{\nabla\Phi}{||\nabla\Phi||} \tag{6}$$

Thus $F(\mathbf{p})$ where $\mathbf{p} \in \mathbf{X}$ is a voxel position is literally the speed at which that point on X moves in the normal direction. Plugging (6) back into (5), we obtain the Level–Set equation

$$\frac{\partial \Phi}{\partial t} + F||\nabla\Phi|| = 0 \tag{7}$$

The Level–Set Method works on a discrete grid representation of Φ, that is (assuming below that unit time step is used and that the grid spacing is also unit)

$$\Phi^n[i, j, k] = \Phi(i\Delta x, j\Delta y, k\Delta z, n\Delta t)$$

This is a 4D discrete function, but, in general, only one time step is stored. In other words, Φ is really represented by a 3D voxel grid. Moreover, the initial value, Φ^0 is typically a distance field. In other words, the voxel grids that are used throughout this paper are precisely the same type of representation as the discretized embedding function Φ which the Level–Set Method works on. The time derivative of Φ is approximated using

$$\frac{\partial \Phi}{\partial t} \approx D^{+t}\Phi = \Phi^{n+1}[i, j, k] - \Phi^n[i, j, k] \tag{8}$$

and if that estimate of the derivative is plugged into (7), we obtain a method for computing one time step:

$$\Phi^{n+1}[i, j, k] = \Phi^n[i, j, k] - F||\nabla\Phi^n|| \tag{9}$$

where the gradient $||\nabla\Phi^n||$ must be computed using one sided derivatives in the upwind direction [34]. The reason is that the solution otherwise has a tendency to become unstable in the presence of discontinuities in the evolving surface. However,

based on the observation that $||\nabla\Phi|| = 1$ everywhere (except at singularities) in a distance field, we have simplified the formula to

$$\Phi^{n+1}[i,j,k] = \Phi^n[i,j,k] - F \tag{10}$$

It might be thought that this could introduce numerical problems, but we have not observed ill effects.

An important question is how to define F. Adalsteinsson et al. have shown that if the speed function fulfills $\nabla F \cdot \nabla\Phi = 0$ then Φ remains a distance field [1]. In other words, the speed function should be constant along the gradient direction. To achieve this, F is always evaluated at the closest surface point – i.e. the point we reach by following the gradient towards the surface. Since we are dealing with distance fields, it is easy to find the closest surface point to a point $\mathbf{p}$ using the *boundary mapping*

$$B(\mathbf{p}) = \mathbf{p} - \nabla\mathbf{\Phi}\mathbf{\Phi}(\mathbf{p}) \tag{11}$$

In the following, it is understood that to evaluate the speed function, the boundary mapping is first used to find the closest surface point (the *foot point*), and then the speed function is evaluated there.

3.1. *Alternative Technique: CIR*

The CIR (Courant Isaacson Rees) scheme has recently been used to solve the Level–Set equation by John Strain [36]. Say we are following the characteristic curve $\mathbf{s}(\mathbf{t})$ defined by

$$\mathbf{s}'(\mathbf{t}) = \mathbf{F}\nabla\mathbf{\Phi} \qquad\qquad \mathbf{s}(\mathbf{0}) = \mathbf{p} \tag{12}$$

for some point $\mathbf{p}$, then

$$\frac{d}{dt}\Phi(\mathbf{s}(\mathbf{t}),\mathbf{t}) = \frac{\partial\mathbf{\Phi}}{\partial\mathbf{t}} + \nabla\mathbf{\Phi}\cdot\mathbf{s}' = \frac{\partial\mathbf{\Phi}}{\partial\mathbf{t}} + \mathbf{F}||\nabla\mathbf{\Phi}|| = \mathbf{0} \tag{13}$$

In other words, Φ is constant along $\mathbf{s}$. At any given point, we can approximate a step along $\mathbf{s}$ by the speed function times the gradient, and that leads to the CIR scheme which is, essentially, to track the characteristic curve from a voxel position one time–step back and then assign the value at that point.

The algorithm as implemented by Strain consists of three steps carried out for all grid points. Let the grid point be $\mathbf{p}$. First we evaluate the speed function $F(\mathbf{p})$. A step back along the characteristic is approximated by $\mathbf{s} = \mathbf{p} - \mathbf{F}(\mathbf{p})\nabla\mathbf{\Phi}$ where (as usual) unit time step is assumed. The value of Φ at $\mathbf{s}$ is computed. Strain uses the so–called ENO scheme [24] to find the value at $\mathbf{s}$ (which is not in general a grid point) – we use trilinear interpolation. Finally, the interpolated value $\Phi(\mathbf{s})$ is assigned to the grid point $\mathbf{p}$.

3.2. *Mean Curvature Flow*

Mean curvature [13] constitutes a very useful, geometry dependent speed function

$$F_{\text{curv}}(\mathbf{p}) = -\kappa_{\mathbf{m}} \tag{14}$$

where κ_m denotes the mean curvature. The sign of the curvature is defined to be positive at a convex point and negative at a concave point. The result is that all regions of high curvature are made smoother, protrusions shrink, and cavities are filled in. This process is known as mean curvature flow and it is a well known and explored application of the Level–Set Method [14].

The formula typically (see e.g. [34]) used to compute mean curvature is

$$\kappa_m = \frac{1}{2} \frac{\begin{pmatrix} (\Phi_{yy} + \Phi_{zz})\Phi_x^2 \\ +(\Phi_{xx} + \Phi_{zz})\Phi_y^2 \\ +(\Phi_{xx} + \Phi_{yy})\Phi_z^2 \\ -2(\Phi_x\Phi_y\Phi_{xy} + \Phi_x\Phi_z\Phi_{xz} + \Phi_y\Phi_z\Phi_{yz}) \end{pmatrix}}{(\Phi_x^2 + \Phi_y^2 + \Phi_z^2)^{3/2}} \tag{15}$$

but, based on the observation that Φ is a distance field, we can use a much simpler formula [20]

$$\kappa_m = \frac{\text{trace(H)}}{2} = \frac{\Phi_{xx} + \Phi_{yy} + \Phi_{zz}}{2} \tag{16}$$

where H is the *Hessian* (i.e. the matrix of second order derivatives.) of Φ. A common way of computing the second order partial derivatives is using the following discrete operator

$$\frac{\partial^2\Phi}{\partial x^2} \approx \Phi(x+1, y, z) + \Phi(x-1, y, z) - 2\Phi(x, y, z)$$

However, we store gradients in the volume and it is simple to compute the second order derivatives by applying central differences to the gradients. This method is a bit unusual, but it is fast and very stable.

3.3. *Rebuilding the Transition Region*

The Level–Set Method is a technique for computing the evolution of surfaces that may expand and contract. If we assume that the speed function is always positive, the so–called Fast Marching Method [33] may be used instead. Applied to a voxel grid, the FMM computes the arrival time of an evolving front. If the front evolves at unit speed, the result is a distance field. The FMM requires that a set of voxels, whose distance values are known, are *frozen* initially. By solving a quadratic polynomial, the distances are then computed at the neighbors of the frozen voxels.

After that, a loop ensues. For each iteration of the loop, the non–frozen voxel having the smallest distance value is frozen, and distances are computed at its neighbors. The distance value of a frozen voxel is never recomputed. Thus, we can

see the FMM as an expanding front. A band of voxels along the front are being recomputed, and voxels behind the front are known and their values frozen.

Because, the FMM can be used to compute distance fields, it can be used to build or rebuild the transition region of a DFV – provided that we know the distance values of a thin band of voxels. A second order version of the algorithm is possible. This version is called the High Accuracy Fast Marching Method (FMMHA), and this method has been used in the work presented here. For more details about how the Fast Marching Methods are implemented, the reader is referred to [34, 33, 10].

3.4. *Implementation*

The volume is stored in a two level hierarchical grid. More precisely, an $N \times N \times N$ grid is represented by an $N' \times N' \times N'$ super–grid where each cell contains an $M \times M \times M$ sub–grid so that $N = MN'$. Because the V–model is clamped to the $[-r, r]$ range only transition voxels need to have an explicit voxel value stored. Interior and exterior voxels all have values of $-r$ and r, respectively. Consequently, a sub–grid is stored only if it contains at least one transition voxels. Otherwise, all its voxels must be either interior or exterior, and only this information is stored for the entire sub-grid.

In its simplest form, the Level–Set Method consists of visiting all transition voxels and replacing each voxel with the result of (10):

$$\Phi[\mathbf{p}] \leftarrow \mathbf{\Phi}[\mathbf{p}] - \mathbf{F}(\mathbf{p}_{\text{foot}}) \tag{17}$$

where F is the speed function evaluated at the foot point $\mathbf{p}_{\text{foot}} = \mathbf{p} - \mathbf{\Phi}\nabla\mathbf{\Phi}$. Note that (17) is really the same as (10) with a slight change of notation. The updating procedure can quite easily be changed to update the voxels using the CIR approach suggested by Strain [36]

$$\Phi[\mathbf{p}] \leftarrow \mathbf{\Phi}(\mathbf{p} - \mathbf{gF}(\mathbf{p}_{\text{foot}})) \tag{18}$$

where $\Phi(\cdot)$ denotes the value of the volume interpolated at a given location. Exactly the same fundamental loop is used in conjunction with both (17) and (18). The only difference lies in how the voxels are updated.

The basic approach is to update all voxels in the transition region using either (17) or (18). However, it is not enough to simply update the voxels. As the surface deforms, some voxels should be added to the transition region, and other voxels should be removed. Recall that voxels are in the transition region if their distance values fall in the range $]-r, r[$ where r is the width of the transition region. If the distance value after updating falls outside this range, it becomes an interior/exterior voxel as appropriate. This does not pose a problem, but it also happens that voxels outside the transition region come closer to the surface than r. In this case the distance needs to be recomputed. This problem could be solved by freezing all transition voxels and then running the fast marching method. However, our experience is that even when evaluating the speed function only at foot points, the voxels in the outer layers of the transition region have a tendency to become less

precise. Consequently, a better idea seems to be to retain only the voxels in the immediate neighborhood of the surface and rebuild the rest using the Fast Marching Method. To concretize "immediate neighborhood" only voxels at $1/2vu$ distance or less from the surface are retained and the rest are rebuilt. This is illustrated in Figure 2.

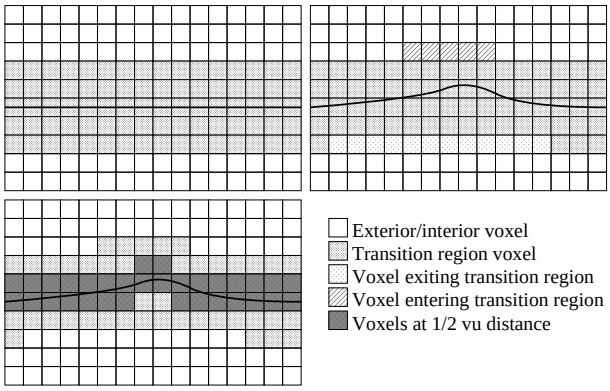


Fig. 2. Level–Set Method

The complete procedure is as follows:

1. Compute new distance value for all transition voxels using (17) or (18).
2. Freeze all voxels at $1/2\ vu$ distance from the surface.
3. Rebuild transition region using the high accuracy Fast Marching Method.

4. An Interactive Volume Sculpting System

We have implemented LSM based tools in our volume sculpting system which will be described in the following.

On start-up, the user is presented with a graphics window and a control panel. All sculpting operations take place in the graphics window, and the control panel is used to select various visualization parameters, the tool, and tool parameters. For instance, the user can select the smoothing tool, the amount of smoothing and the size of the smoothed region in the control panel and then apply the smoothing tool in the graphics window.

In the graphics window, the user simply places the mouse cursor above the place where a change is desired and clicks to invoke the tool. Apart from sculpting operations, the graphics window also allows the user to zoom in on the model, pan or rotate the view using a virtual trackball.

The system does not only support Level–Set based tools but also tools based on volumetric CSG (Constructive Solid Geometry) [8]. These tools allow the user to

add or subtract shape primitives such as spheres and polyhedra. In addition CSG has been used to implement a simple cut and paste facility.

Like the LSM based tools, the CSG tools are invoked by pointing and clicking. However, in the case of CSG, the orientation of the tool is important since the effect of adding or subtracting a shape depends on its orientation. The user can switch between three modes, one where the orientation is locked to the normal of the shape being sculpted and another mode where the orientation is locked to the view direction. Finally, a second virtual trackball can be used to orient the CSG tool.

The system has been written in C++ using FLTK as the GUI toolkit, and it runs on Linux and Windows. For the timings below, an 800 MHz Athlon based system (running Linux) equipped with 256 MB RAM and a GeForce2 GTS graphics card was employed.

4.1. *Level–Set Sculpting Tools*

LSM based sculpting tools differ only by their associated speed functions. The simplest possible speed function is a constant speed function. A constant speed function

$$F_{\text{const}}(\mathbf{p}) = \mathbf{k} \tag{19}$$

pushes the boundary uniformly outwards and thus results in a dilation. A speed function which may be used to add a small protrusion (or dent) to the surface is the 3D Gaußian

$$F_{\text{bump}}(\mathbf{p}) = \exp^{-||\mathbf{p}-\mathbf{p_0}||/\mathbf{2}\sigma^{\mathbf{2}}} \tag{20}$$

The already mentioned mean curvature speed function is used to smoothen the surface.

$$F_{\text{curv}}(\mathbf{p}) = -\kappa_{\mathbf{m}}$$

An important piece is missing. We want the user to be able to make local changes. Locality means that the entire Level–Set Method is used only in a ROI around the center of the tool, and the value of the speed function should be 0 on the boundary of the ROI. To achieve this, a radially symmetric windowing function is used. This function can be seen as a speed function that is controlled by four parameters: A scaling factor α, a window radius r, a window transition region thickness k, a center point $\mathbf{p_0}$, and another speed function F. The definition is

$$F_{\alpha r k \mathbf{p_0 F}}(\mathbf{p}) = \alpha \mathbf{F}(\mathbf{p}) \mathbf{w_{rk}}(||\mathbf{p} - \mathbf{p_0}||) \tag{21}$$

where

$$w_{rk}(t) = \begin{cases} 1 & 0 \le t < r \\ 1 - 3(\frac{t-r}{k})^2 - 2(\frac{t-r}{k})^3 & r \le t \le r+k \\ 0 & t > r+k \end{cases} \tag{22}$$

Notice that w_{rk} is a C^1 function. This ensures that the speed function decreases smoothly to 0. The scaling factor α is used to scale the effect of the tool.

The following concrete sculpting tools have been implemented:
1. Add blob: F_{bump} used in conjunction with the scaling–windowing speed function. This tool is local only. 2. Remove blob: Same as above, but with negative scaling.
3. Smooth: F_{curv} used either in conjunction with the scaling–windowing speed function or without, depending on whether a global or a local smoothing is desired. Scaling is used to determine the degree of smoothing.
4. Un–smooth: Same as above but with a negative scaling.
5. Dilate: F_{const} used with scaling but usually not windowing since a dilation of a part of an object is rarely desirable.
6. Erode: Same as above but with negative scaling.

4.2. *CSG Tools*

While Level–Set based tools suffice for many purposes, they are not ideal if the user wishes to include, say, a torus, sphere, cube or some other specific shape in the design. Therefore, the system also provides tools based on volumetric CSG [23].

Volumetric CSG is very simple in the case of binary volumes where we simply perform a boolean operation between the volume containing our sculpture and a tool volume. For grey–level volumes Perlin's operators [28] are sometimes used [39]. In the case of distance fields, most authors use min or max to produce union and intersection, respectively [19, 5]. In other words, for each voxel, we simply compute the new voxel value as the minimum of the old distance value and the value of a tool volume. It is well known that this operation does not produce correct distances for all voxels. Fortunately, it is possible to find and correct these voxels quite easily [11].

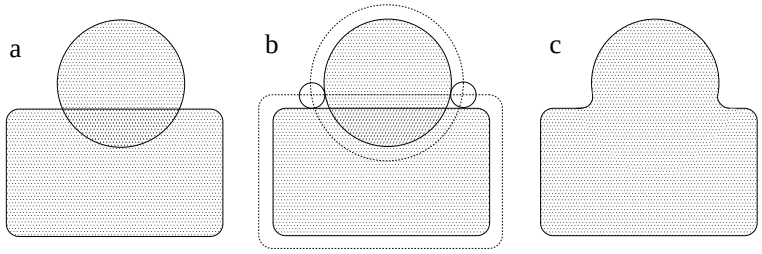


Figure 3: Illustration of a CSG operation with a rolling ball blend between the old and the added shape.

However, CSG based on the min operator does introduce sharp edges which are not well represented in volumes due to the high frequency content of such features. To ameliorate this problem, we have previously proposed a method for volumetric CSG which performs a fixed radius rolling ball blend between the tool and the model [8] (see Figure 3). In some cases, the rolling ball blend itself introduces small, ill–represented features. However, in most cases it produces a much improved result.

4.3. *Visualization*

A fast method for visualizing volume data is very important in volume sculpting. The methods typically employed are either ray casting [39, 3, 7] or a variation of the well–known Marching Cubes algorithm [16, 15, 30] by Lorensen et al. [25].

Two methods have been implemented: Marching Cubes and a point rendering method inspired by [35]. Figure 5 (right) illustrates both methods. Both methods were implemented using OpenGL, and in the following, we briefly discuss the point rendering method. For each transition voxel within a given distance of the boundary, the boundary mapping (11) is used to produce a foot point. Together with the normal, this point can be rendered using the OpenGL point primitive. To facilitate perspective projection, points are scaled according to the distance to the surface. Not all surface points are recomputed each time the volume is changed. A point–bin is associated with each sub–grid of the hierarchical grid. When the volume is changed, the points of a sub–grid are recomputed only if at least one voxel has been changed.

One strength of the point rendering method lies in its simplicity, and our tests indicate that point rendering is almost always faster than triangle rendering. However, the biggest advantage of point rendering lies in the fact that it is much faster to generate the primitives (i.e. points). Also, efficient triangle rendering requires that the triangles are stored in triangle strips, but stripping adds additional overhead. To isolate the effect of primitive generation, we performed a test where primitives were regenerated for a number of random cells each frame, but no actual sculpting was performed. This resulted in 50 % slower rendering when marching cubes was used instead of point rendering.

The weakness of point rendering is that at low resolutions or when zooming in close, points become visible and the quality of MC visualization is better in these cases. When zooming in on a small model, point rendering might also be slower than triangle rendering, since the points overlap and triangles do not. This means that if the program is fill–limited, triangle rendering becomes faster. In practice, though, this is rarely the case.

5. Results

The interactive system has been used to create a number of sculptures. The effects of some of the sculpting tools are shown isolated in the top row of Figure 4. The add and remove blob tools were used to bore a hole through and create a handle on the cube, respectively. The smoothing tool was used to smoothen one corner of the cube. More elaborate sculptures are shown in Figure 5. The bear model is a $256 \times 256 \times 256$ volume whereas the head is stored in $1024 \times 1024 \times 1024$ volume (which would take up far too much storage except for the hierarchical grid). The LSM based tools discussed in this paper are the primary tools that have been used to create the models. However, two other techniques have also been used: For the eyes of the head (Figure 5 left), volumetric CSG was used to create the eyeballs.

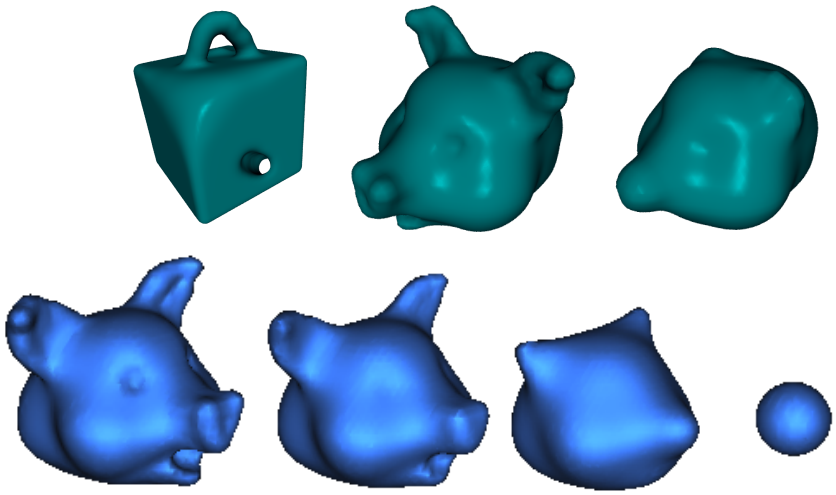


Fig. 4. Top: Effect of add/remove blob and smoothing (left), a "marzipan pig" (center), and marzipan pig after open with a sphere or radius 3 (right). Bottom: Volume Sculpture of a "marzipan pig" under mean curvature flow.

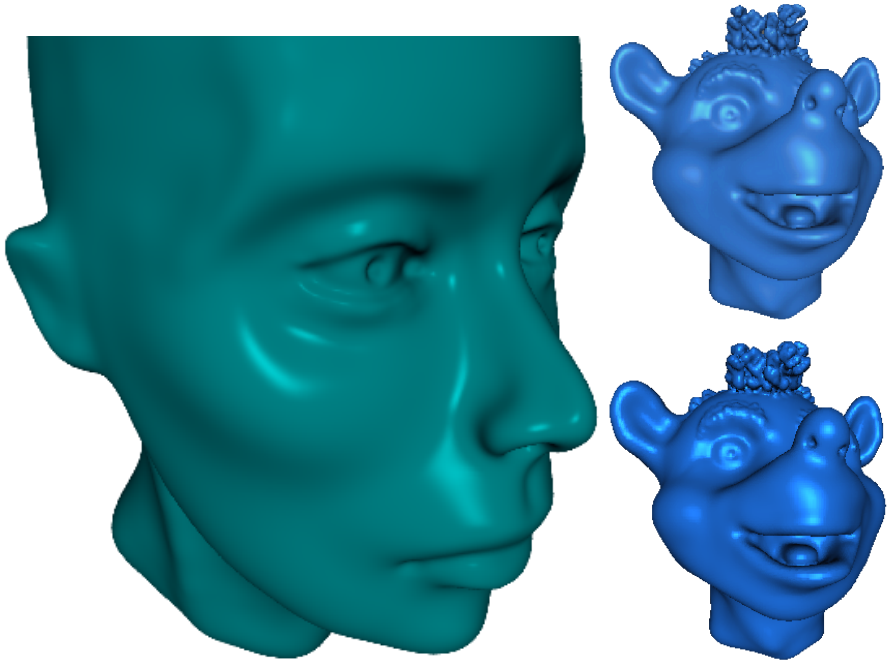


Fig. 5. Left: Head model visualized using ray casting. Right: Bear model rendered using Marching Cubes (top) and point rendered (bottom).

Secondly, the resolution was changed during sculpting. In the case of both models, very crude resolutions (e.g. $32 \times 32 \times 32$) were used during the initial work. The resolution was then increased by a factor of two while finer details were added. Doubling the resolution entails an interpolation of the values of new voxels which turned out to introduce slight artifacts. These were easily removed using global smoothing. Figure 6 illustrates the multiresolution sculpting process.

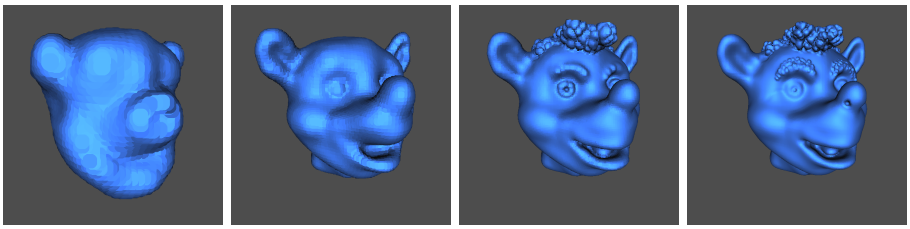


Fig. 6. These four images show the bear model at different levels of resolution ($32 \times 32 \times 32$ up to $256 \times 256 \times 256$) and corresponding stages of the sculpting process.

As an alternative to iteratively increasing the resolution, we can use volumetric CSG to create the initial workpiece. In Figure 7 the process is illustrated: The image on the left was created by adding several spheres and a tetrahedron and removing a cube. Note that the rolling–ball blend CSG method (see Section 4.2) was used to prevent sharp edges. On the right smoothing and add blob have been used to add details to the model.

The add blob and smoothing tools have been seen in previous sculpting systems. However, using the LSM, new possibilities emerge. The un–smooth tool which was used to create the hair imitation on the bear is one example. A more practically useful example is shown in the top row of Figure 4: The marzipan pig model on the left has been eroded and then dilated with a ball of radius three *vu* producing a morphological opening.

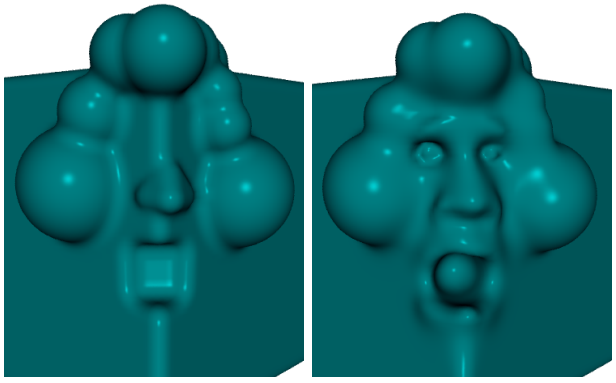


Fig. 7. The image on the left shows a crude model created using only CSG operations. The image on the right shows the same model after LSM based sculpting operations.

Table 1. Timings for the add blob and smoothing tools.

	Add blob		Smoothing	
ROI	applications	avg. time/sec	applications	avg. time/sec
10x10x10	612	0.044	1674	0.047
20x20x20	654	0.134	738	0.153
30x30x30	192	0.307	250	0.352
40x40x40	128	0.703	132	0.878
50x50x50	110	0.973	138	1.109
60x60x60	65	1.246	140	1.293
70x70x70	43	1.744	64	1.453

The figures discussed above give an idea of the scope and effectiveness of the sculpting tools. Another important concern is speed. The speed has been tested by a user experiment. The add blob and local smoothing tools were applied by a sculptor using ROIs ranging from $10 \times 10 \times 10$ voxels to $70 \times 70 \times 70$ voxels. For each tool and each size of ROI the tool was applied a number of times in a random fashion. The results are shown in Table 1. As the table indicates, the method is easily interactive for small tools. Large tools are clearly much slower, but the default tool size is $20 \times 20 \times 20$ which is reasonably fast at about 0.15 seconds per application.

It has been mentioned several times that our method preserves the distance field representation. This is ensured by the way the speed function is extended and by the fact that distances are recomputed for all voxels at more than $0.5vu$ distance from the boundary. However, it is hard to prove that the volume remains a distance field. Also, some numerical error must be allowed for. The best test seems to be to verify that the length of the gradient is unit, since the gradient of a distance field must be unit–length except at critical points of the distance field [19].

An experiment was carried out. The experiment consisted of 400 applications of the add blob tool interspersed with 400 applications of the smoothing tool. The tools were applied to random points on the side of the cube. Afterwards, gradients were computed for voxels incident on cells intersected by the boundary. As one would expect the error is quite low – nowhere higher than $0.07vu$. Moreover, the greatest error is near the edges where curvature is an important source of error.

Note also that the point rendering relies on the boundary mapping which works only for distance fields. Inaccuracies would translate into errors in the visualization.

6. Conclusions and Future Work

We have shown that it is feasible to use the Level–Set Method as the underlying technology of a volume sculpting system.

The method is generic. Any deformation which can be expressed through a speed function can be implemented using the Level–Set Method. By introducing the scaling–window in the context of the LSM, we have provided a way of using

the Level–Set Method also for local manipulations. This has led to very effective tools for smoothing and for adding or subtracting blobs of material from the model. Moreover, tools that were not previously possible have been implemented. For instance, the un–smooth tool used in the bear volume. Morphological operations have also been shown.

Another advantage of our method lies in the fact that the LSM maintains a cleaner volume representation than previous methods. The fact that a signed distance volume is maintained has been exploited to simplify the computation of curvature and the visualization.

For some sculptures, the LSM based tools suffice. However, in general, CSG tools [8] are also important. Thus, the LSM based tools should be seen as just one component of a complete sculpting system, but an important component. Moreover, the tools we have implemented are probably only the beginning, and others are envisioned. For instance, it would be possible to create shearing or warping speed functions.

It turned out to be very valuable to be able to begin sculpting at low resolutions and then gradually increase the resolution. This multiresolution approach could potentially be made more powerful by allowing the user to change resolution locally, during interactive sculpting. We believe this is an important goal, albeit difficult.

References

1. D. Adalsteinsson and J.A. Sethian. The fast construction of extension velocities in level–set methods. *Journal of Computational Physics*, 148(1):2–22, 1999.
2. H. Arata, Y. Takai, N.K. Takai, and T. Yamamoto. Free-form shape modeling by 3D cellular automata. *Proceedings Shape Modeling International '99. International Conference on Shape Modeling and Applications*, pages 242–7, 1999.
3. Ricardo S. Avila and Lisa M. Sobierajski. A haptic interaction method for volume visualization. In Roni Yagel and Gregory M. Nielson, editors, *Visualization '96*. IEEE, 1996.
4. L. Barthe, B. Mora, N. Dodgson, and M. Sabin. Triquadratic reconstruction for interactive modelling of potential fields. *Shape Modeling International, 2002. Proceedings*, pages 145–153, 2002.
5. David E. Breen, Sean Mauch, and Ross T. Whitaker. 3D scan conversion of csg models into distance volumes. In Stephen Spencer, editor, *Proceedings of IEEE Symposium on Volume Visualization*, October 1998.
6. D.E. Breen and R.T. Whitaker. A level-set approach for the metamorphosis of solid models. *Visualization and Computer Graphics, IEEE Transactions on=20*, 7(2):173–192, 2001.
7. Andreas Bærentzen. Octree–based volume sculpting. In Craig M. Wittenbrink and Amitabh Varshney, editors, *LBHT Proceedings of IEEE Visualization '98*, October 1998.
8. Andreas Bærentzen and Niels Jørgen Christensen. A technique for volumetric csg based on morphology. In *Proceedings of International Workshop on Volume Graphics*, 2001.
9. Andreas Bærentzen, Miloš Šrámek, and Niels Jørgen Christensen. A morphological

approach to the voxelization of solids. In Vaclav Skala, editor, *Proceedings of WSCG 2000*, volume I, February 2000.
10. J. Andreas Bærentzen. On the implementation of fast marching methods for 3d lattices. Technical Report IMM-REP-2001-13, DTU.IMM, 2001. http://www.imm.dtu.dk/~jab/publications.html.
11. J. Andreas Bærentzen. *Volumetric Manipulations with Applications to Sculpting*. PhD thesis, IMM, Technical University of Denmark, 2001.
12. Jerome A. Broekhuijsen, Robert P. Burton, and William A. Barrett. Interactive editing of volumetric objects with 3D input and output devices. *Journal of Imaging Technology*, 17(6):269–274, December 1991.
13. Manfredo P. Do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice Hall, 1976.
14. D. L. Chopp and J. A. Sethian. Flow under curvature: Singularity formation, minimal surfaces, and geodesics. *Journal of Experimental Mathematics*, 2:235–255, 1993.
15. Eric Ferley, Marie-Paule Cani, and Jean-Dominique Gascuel. Practical volumetric sculpting. *the Visual Computer*, 16(8):211–221, December 2000.
16. Tinsley A. Galyean and John F. Hughes. Sculpting: An interactive volumetric modeling technique. *ACM Computer Graphics*, 25(4), July 1991.
17. Sarah F. Frisken Gibson. Using Linked Volumes to Model Object Collisions, Deformation, Cutting, Carving and Joining. *IEEE Transactions on Visualization and Computer Graphics*, 5(4), October–December 1999.
18. Sarah F. Frisken Gibson, Ronald N. Perry, Alyn P. Rockwood, and Thouis R. Jones. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proceedings of SIGGRAPH 2000*, pages 249–254, 2000.
19. Sarah F.F. Gibson. Using distance maps for accurate surface representation in sampled volumes. In Stephen Spencer, editor, *Proceedings of IEEE Symposium on Volume Visualization*, October 1998.
20. Erich Hartmann. On the curvature of curves and surfaces defined by normalforms. *Computer Aided Geometric Design*, 16(5):355–376, 1999.
21. Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: A sketching interface for 3D freeform design. In *Proceedings of SIGGRAPH 1999*.
22. R.T. Whitaker K. Museth, D.E. Breen and A.H. Barr. Level set surface editing operators. *ACM Transactions on Graphics(Proc. SIGGRAPH)*, 21(3):330–338, July 2002.
23. Arie Kaufman, Daniel Cohen, and Roni Yagel. Volume graphics. *IEEE Computer*, 26(7), July 1993.
24. Randall J. Leveque. *Numerical Methods for Conservation Laws*. Birkhäuser, 1992.
25. W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *ACM Computer Graphics*, July 1987.
26. S.R. Marschner and R.J. Lobb. An evaluation of reconstruction filters for volume rendering. *Proceedings. Visualization '94 (Cat. No.94CH35707)*, pages 100–7, CP10, 1994.
27. Bradley A. Payne and Arthur W. Toga. Distance field manipulation of surface models. *IEEE Computer Graphics & Applications*, 12(1), 1992.
28. K. Perlin and E.M. Hoffert. Hypertexture. *Computer Graphics*, 23(3):253–67, 1989.
29. Ronald N. Perry and Sarah F. Frisken. Kizamu: A system for sculpting digital characters. In *Proceedings of SIGGRAPH 2001*, 2001.
30. Alon Raviv and Gershon Elber. Three–dimensional freeform sculpting via zero sets of scalar trivariate functions. *Computer–Aided Desing*, 32:513–526, August 2000.
31. Alan E. Richardson, Robert P. Burton, and William A. Barrett. Sculptbox - a volumet-

ric environment for interactive design of 3D objects. In *Proceedings, 1990 SPIE/SPSE Symposium on Electronic Imaging Science and Technology*, pages 198–209, 1990.
32. Saurabh Sethia and S. Manohar. Minkowski Operator for Voxel Based Sculpting. *Computers and Graphics*, pages 593–600, 1998.
33. James A. Sethian. Fast marching methods. *SIAM Review*, 41(2):199–235, 1999.
34. James A. Sethian. *Level Set Methods and Fast Marching Methods*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, second edition, 1999.
35. Nilo Stolte and Ari Kaufman. Parallel spatial enumeration of implicit surfaces using interval arithmetic for octree generation and its direct visualization. In *Proceedings of Implicit Surfaces'98*, pages 81–87, 1998.
36. John Strain. Semi-Lagrangian methods for level set equations. *Journal of Computational Physics*, 151(2):498–533, 1999.
37. Greg Turk and James F. O'Brien. Modelling with implicit surfaces that interpolate. *ACM Transactions on Graphics*, 21(4):855–873, October 2002.
38. Miloš Šrámek and Arie Kaufman. Alias–free voxelization of geometric objects. *IEEE Transactions on Visualization and Computer Graphics*, 5(3), July/September 1999.
39. Sidney Wang and Arie E. Kaufman. Volume sculpting. In *1995 Symposium on Interactive Graphics*. ACM SIGGRAPH, 1995.
40. R. Whitaker, D. Breen, K. Museth, and N. Soni. A framework for level set segmentation of volume datasets. In *International Workshop on Volume Graphics 2001*, pages 159–168, 2001.
41. A.P. Witkin and P.S. Heckbert. Using particles to sample and control implicit surfaces. *Computer Graphics Proceedings. Annual Conference Series 1994. SIGGRAPH 94 Conference Proceedings*, pages 269–77, 1994.