Dynamic implicit surface tesselation

Max Froumentin, Eric Varlet
Laboratoire d'Informatique Fondamentale de Lille
Université de Lille I
59655 Villeneuve d'Ascq Cedex
Email: froumentin@lifl.fr, varlet@lifl.fr

Abstract

This paper presents a method for quickly computing polygonal approximations of dynamic implicit surfaces: when the shape of a surface is changed, its polygonal approximation is updated in real-time. The algorithm relies on the fact that moving a control point only affects the part of the surface within that point's radius of influence. A real-time environment for our algorithm is provided by a medical simulator where human organs modelled using implicit surfaces are constantly deformed and have to be displayed at an interactive frame rate.

1 Introduction

Implicit surfaces are a well-known tool for conveniently designing closed continuous shapes (figure 1). They also provide simple solutions for creating blends or ramifications. An appropriate field of application where this kind of surface is useful is biological structures modelling, in particular human organs (figure 2).

The computer graphics group at LIFL is concerned with designing medical simulators that allow novice physicians and surgeons to learn complex medical procedures harmlessly. In particular project SPEED ("Simulateur d'Echo-Endoscopie Digestive") focuses on ultrasound endoscopy [12], which consists in observing organs around the digestive tracts using ultrasound images.

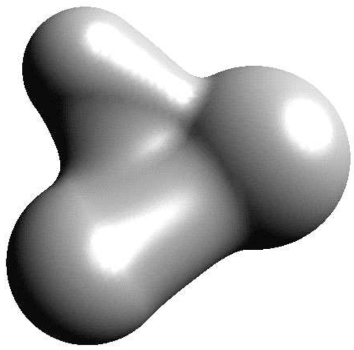

Figure 1: Implicit surface defined by 3 control points

The physician gets those images by introducing a supple endoscope via the mouth into the digestive tracts. We designed a computer system to make physicians learn the complex handling of the endoscope inside the duodenum, which is located in the first 20 cm after the stomach. At the end of the endoscope, the ultrasound probe is surrounded by a water balloon that is pressed against the duodenum and widens it.

The simulator is composed of a mechanical interface which retrieves the movements of the physician and sends positions to the computer. The computer calculates the position of the probe in the digestive track and displays the corresponding simulated ultrasound image. In order to achieve this, we designed a 3D model of the organs. We chose implicit surfaces in order to get a realistic model that allows us to get a good approximation of a real echographic image.

As the goal of the simulator is initial training, we added a 3D view of the organs to help the beginner locate the probe inside the duodenum. To display this view in real time, it is necessary to tesselate our implicit surface model of the organs. Moreover, as the balloon surrounding the probe deforms the duodenum, the surface has to be tesselated at frame rate. The algorithm we present here allows us to achieve this goal, but can also be used in many other applications using implicit surfaces.

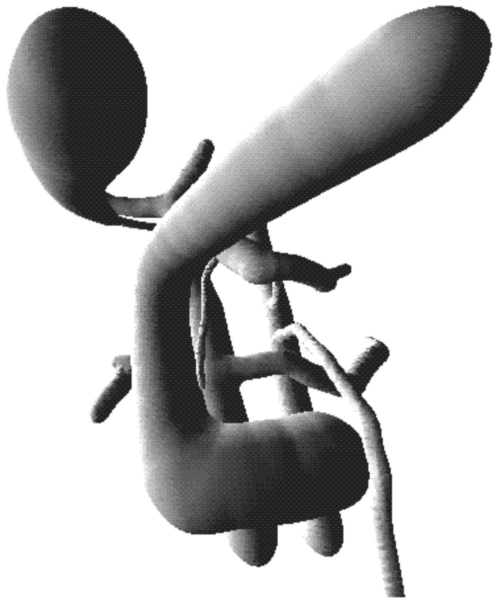

Figure 2: Implicit surfaces modelling a duodenum among other abdominal organs

Section 2 will define implicit surfaces and, in particular, the type of implicit surfaces we use, as well as tesselation techniques. Section 3 will describe our dynamic tesselation method and section 4 will discuss the results.

2 Implicit Surfaces

2.1 Definition

Implicit surfaces are defined as the locus of points in space which satisfy a given equation: $f(x, y, z) = 0$. When f is a polynom, the surface is said to be *algebraic* [1]. Algebraic surfaces provide an intersting alternative to the well-known parametric surfaces as they allow fast intersection computation or boolean operator modelling, although they lack good shape control. In 1982, Jim Blinn [2] introduced a more general concept of implicit surface where f is no longer a polynom, but which provides good shape and continuity control. Blinn's idea was to model electron density maps of molecular structures: a set of control points (the electrons) define a density function of space which is maximum at the electron positions and converges to 0 towards infinity. The implicit surface S is then defined as the set of points where the density function takes a given threshold value T:

$$S = \{(x, y, z) \quad | \quad f(x, y, z) = T\} \qquad (1)$$

The surface has the shape of a smooth bond around the control points (figure 1 shows a surface defined by three control points).

Alternate definitions include using polynomial density functions (Blinn originally used a Gaussian distribution) for faster computation [15] and generalized control primitives such as lines or planes for better shape control [4]. For our purposes, we chose a model using control points with a polynomial density function. This allows fast density computation and the creation of not-so smooth shapes (i.e. G^1 but not G^2, see figure 3 left) that really look like organic tissues. A control point is specified as:

- a point P_i in space;
- a *radius* R_i: the radius of the sphere defined by that single point;
- a *radius of influence* RI_i: the distance beyond which the point's density is zero.

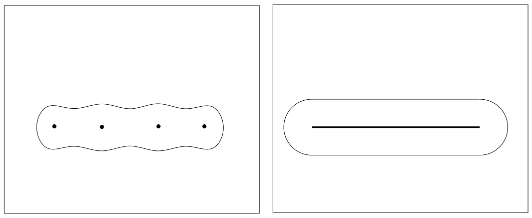

Figure 3: Point primitives (left) vs. line primitives (right) – 2D projection

The density function for a single control point is then defined as:

$$D_i(x, y, z) = \begin{cases} \left(\frac{d_i^2 - RI_i^2}{R_i^2 - RI_i^2}\right)^2 & \text{if} \quad d_i < RI_i \\ 0 \quad \text{otherwise} \end{cases} \qquad (2)$$

where d_i is the distance from (x, y, z) to P_i (figure 4).

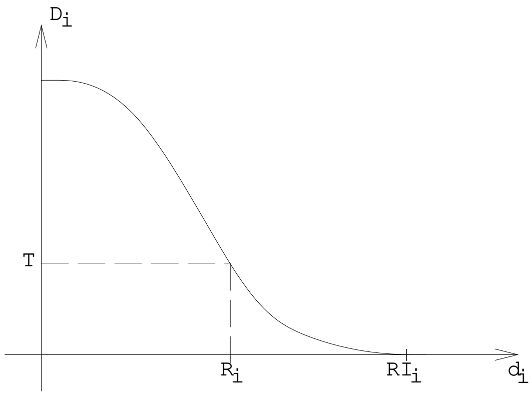

Figure 4: Density function

The global implicit function D is then defined as the sum of all individual density functions:

$$D(x, y, z) = \sum_{i=0}^{n} D_i(x, y, z) \tag{3}$$

In order for the R_i values to actually represent the radii of the spheres defined by individual control points, the threshold T must be set to 1. Therefore a surface is defined as :

$$S = \{(x, y, z) \quad | \quad D(x, y, z) = 1\} \tag{4}$$

This definition has the following properties:

1. The global implicit function is a piecewise quartic polynom in x, y and z.
2. The surfaces created using this function are tangent continuous (G^1, but not G^2) for any number of control points in any configuration.
3. Each control point influences the shape of the surface only within its radius of influence.
4. As with most other implicit surface definitions, surfaces have a "blobby" aspect, suitable for human organ modelling.

2.2 Tesselation Techniques

Although implicit surfaces easily lend themselves to direct rendering using ray tracing methods [2, 8], these methods do not, at this time, allow real-time surface display. Currently only tesselation algorithms allow fast rendering using graphics accelerators. But whereas parametric surfaces possess surface coordinates, making real-time tesselation possible, implicit surfaces have to rely on completely different and less efficient techniques.

Nearly all tesselating algorithms for implicit surfaces use space subdivision methods, introduced by Wyvill *et al.* in 1986 [15, 10]: a given volume bounding the surface is subdivided into voxels (usually cubes or tetrahedra). For each of the voxels that intersect the surface, a set of facets that approximate the actual intersection is computed by interpolating density values at the voxel corners (figure 5). All the facets computed in this way form a polygonal approximation of the surface, the approximation being better as the voxels are smaller. This algorithm has the advantage of requiring only implicit function evaluations; it is also independant of the topology of the surface, although ambiguities can be introduced due to large voxel size [11]. The main issues that have been studied in later papers are speed [3] and adaptive tesselation [13]. Algorithms that have very recently been published use a new approach: displaying a surface as a set of "particles" that move on it [5, 14, 6].

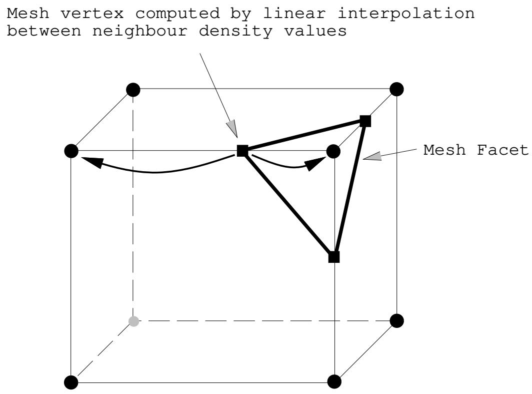

Figure 5: Computing a facet inside a voxel

A particular aspect of tesselation algorithms that has received less attention is that of surfaces that change shape: real-time displaying a moving implicit surface requires real-time tesselation. None of the space subdivision methods presented above seems fast enough to be executed at a high speed rate (typically 25 times per second).

In 1988, Jevans *et al.* made the first attempt at speeding up space subdivision algorithms for implicit surface animation [9], and have introduced some of the principles we exploit here. More recently, methods based on *particle systems* [14, 6] have explored this issue, but they suffer from other problems: while Witkin's method does not produce a tesselation of the surface (only vertices giving a global idea of the shape), Desbrun's algorithm introduces visual artifacts where control skeletons blend that make his method difficult to apply to a control point-only defined implicit surface.

As our application required a method for fast tesselation update, we have been adapting former algorithms (based on space subdivision) as well as designing new schemes to reach that goal.

3 Dynamic tesselation

We define an elementary shape change of a given surface as the displacement of a single control point. More complex shape changes will for the moment be considered as moving each control point individually.

The obvious method to handle our problem given any tesselating algorithm is to completely tesselate the surface each time the surface changes shape. This is of course very slow in practise but we will see that this method can be greatly enhanced.

The first property from our surface definition that is useful for dynamic tesselation is that control points have a radius of influence, which means that each of them has only a *local* influence on the shape. Therefore only the part of the surface that was modified should be re-tesselated at each step. This implies designing an efficient data structure to store density values that do not change and the polygonal model, as well as algorithms to keep it up-to-date.

3.1 Data structure for moving objects

The data structure we use for storing relevant information is an addition to the standard density value lattice used in *marching cubes* algorithms [10, 7]. We use three different structures:

- *a 3D array of density values at the lattice vertices.* This standard structure allows us to minimize implicit function evaluation (figure 6). Any density along an edge of the lattice can be evaluated by linear interpolation between two values of this array (as in figure 5).
- *a 3D array of gradient values at the lattice vertices.* Computing surface gradients are necessary to obtain the surface normal at vertices of the mesh. Storing gradient values at lattice vertices allows us to compute a surface normal anywhere on the lattice edges by linear interpolation, in the same way as density values (figure 7).

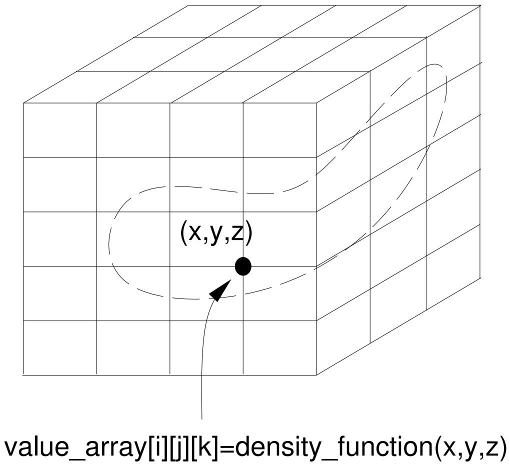

Figure 6: Stored density values

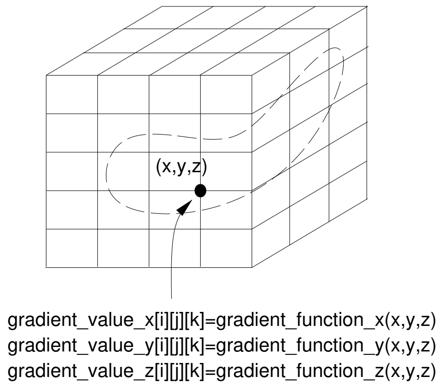

Figure 7: Stored gradient values

- *a 3D array of colour values.* The colours at the array vertices are computed by an appropriate function of space, allowing the colour at any point to be obtained by linear interpolation, in the same way as gradients (figure 9).
- *a 3D array of voxels containing, for voxels that intersect the surface, a chained list of the triangles inside that voxel*: this data structure holds the

polygonal mesh, so the part of the surface that does not change needs not be tesselated again (figure 8).

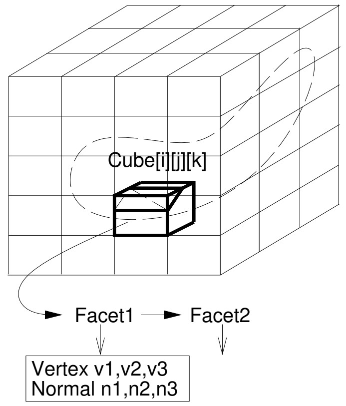

Figure 8: Polygonal mesh storage

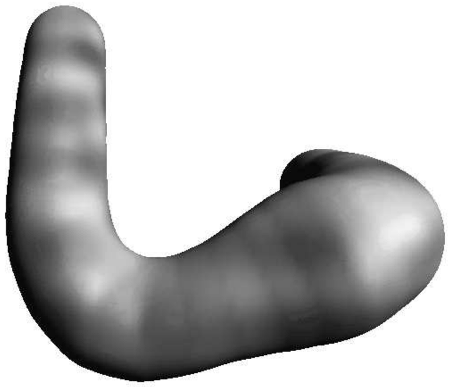

Figure 9: Textured duodenum

3.2 Moving a sphere

As the algorithm we propose handles fast tesselation *update* when a control point is moved, we have to assume that the surface is correctly tesselated in the first place, i.e. the data structure above has been correctly initialized. Speed in the initialization part is not critical to us as it only needs to be done at the beginning. Therefore we use a standard non-adaptive tesselating algorithm [15, 10].

The 3D model is displayed by examining each voxel in the 3D array (figure 8). If a voxel contains facets then these are sent to the graphics engine.

The principle for keeping the tesselation up-to-date is to update only the two regions containing the starting and ending positions of the moving control point's sphere of influence (actually, the smallest cube bounding the sphere). The rest of the surface does not change by definition so nothing has to be done outside these two regions (figure 10 shows a 2D example of a control point moving, with the starting and ending regions).

The procedure used to update the data structure in the starting and ending regions is:

1. (a) Substract the single-point density (eq. 2) and gradient values of the moving control point in the area of its starting position (figure 10 left).
   (b) Update the mesh data structure by applying a standard tesselating method in the considered area.
2. (a) Add the single-point density and gradient values of the moving sphere in the area of its ending position (figure 10 right).
   (b) Update the mesh data structure by applying a standard tesselating method in the considered area.

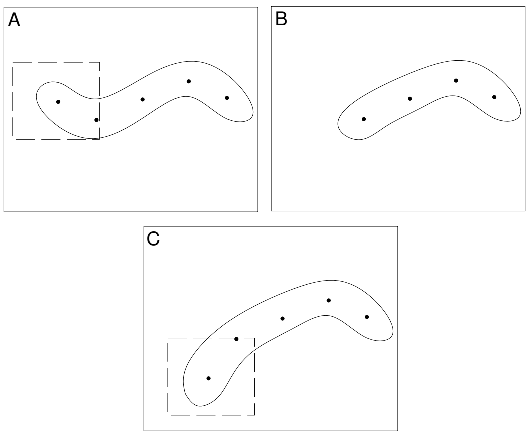

Figure 10: Moving a control point. A: before step 1, B: after step 1, C: after step 2

The obvious benefit of this method is that storing the polygonal mesh avoids having to tesselate the whole surface at each step. But it also has the important advantage of requiring only density function evaluation *for single spheres*, which means that the sum of densities (eq. 3) is never evaluated for updating the mesh. This is very important as, contrary to Javans' method [9], we do not need a data structure to speed-up density summations. Our algorithm can be therefore considered independant of the number of control points, i.e. complexity of the surface [1].

Computing surface normals by interpolating gradients at lattice vertices while only adding or substracting single sphere gradients for animation again makes our algorithm independant of the number of control points.

3.3 Further optimization

Other schemes were implemented in order to get an even faster update algorithm. Although these are not quite as fundamental as the basic procedure described above they are necessary to reach real-time update.

- *Small moves*: If the displacement of a control point is small, the two boxes whithin which the density values are updated can greatly overlap (figure 11 right). This situation can be exploited by joining steps 1 and 2 above in a single procedure updating the bounding box of the starting and ending position (grey region in figure).

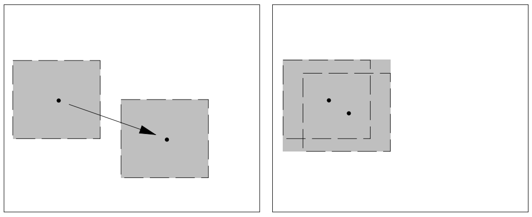

Figure 11: Large and small moves (2D projection)

- *Moving groups of spheres*: in a general dynamic scheme, usually more than a single sphere could be moved at a time. The simple way of achieving it is simply to use the method described above by moving the spheres one by one. But for a large number (n) of spheres, this can be quite unefficient, as density values will be updated n times whereas they could be computed once for all. A faster method is to join the update areas of the moving control points into a single paralellepiped (figure 12).

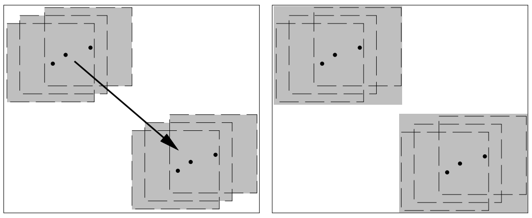

Figure 12: Moving several control points individually (left) and grouped (right)

- *"Simple" surfaces*: acurate time measuring shows that for implicit surfaces defined by less than about 100 control points and that are polygonised at a convenient size, it is faster not to store gradient information as updating all the gradient values at voxel vertices becomes slower than computing gradients at surface vertices (voxel vertices being much more numerous that surface vertices in this case).

Some of the most "obvious" schemes for making our algorithm faster in practise happened to be slower than the algorithm we described. Nevertheless we consider it instructive to describe here two of these schemes and explain why they fail to make the tesselation update faster.

- *1D list of nonempty cubes* [9]: using a list (vector or linked list) could make the process of displaying the mesh faster by not having to check for voxel emptiness. Unfortunately the cost of updating such a structure is a lot greater than the perfomance gained.
- *Boundary representation of the mesh*: storing a full boundary representation of the polygonal mesh, allowing vertex normal vector computation by averaging surrounding facet normals, in the usual Gouraud-shading way, could save the cost of storing and updating gradients. Unfortunately, visual results obtained in this way are poor: as the polygonal models generated by any implicit surface tesselating method exhibit large differences in triangle size (figure 13), averaged normal are often too distant from "true" gradient normals. Moreover maintaining and reading the structure again loses from the overhead generated by allocating/freeing memory as well as additional pointer indirection.

[1] unless one considers the number of triangles a direct consequence of the number of control points.

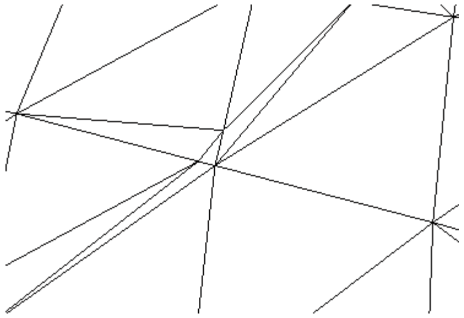

Figure 13: Tesselation of an implicit surface by space subdivision (closeup)

4 Results

The data structure and algorithms presented above have proved to be a useful tool for modelling and animating objects made of implicit surfaces (see below). Unlike most algorithms that tesselate the whole surface, our method is fast enough for real-time shape modification, while keeping a correct level of approximation. On a standard graphics workstation (SGI O2), a single control point move on a surface tesselated in a convenient $30\times30\times30$ lattice takes between 10 and 100 ms depending on its radius of influence. This provides more than convenient frame rates, even when moving more than single control points at a time.

Nevertheless, two disadvantages that could be attributed to our method are:

- *The algorithm is dependant on the type of implicit surface we use*: we necessarily need a surface model that has radii of influence. However nearly all of the implicit surface schemes used at this time are polynomial and do have radii of influence.
- *The compensation for the gain in speed is memory size.* A lattice of $30\times30\times30$ vertices takes 4 floating point numbers per vertex (1 for the density value and 3 for the gradient) leading to a memory occupation of more than 800 Kb. Although it is difficult to give a accurate estimate to the number of facets, an order of 1 Mb is reached by the display list data structure for a standard surface. This gives a broad estimate of less than 2 Mb for an average scene at standard resolution, the most complex examples we could produce never went past 5 Mb. This is well below the memory limit of a standard graphics workstation, avoiding time-consuming overhead caused by memory management (swapping etc.)

Application

The first application for which the algorithm was implemented is the endoscopy simulator we presented above. A 3D model of a duodenum and the surrounding organs were modelled using the implicit surface scheme we presented in section 2. To simulate the deformation of the duodenum by the balloon, we added a control point centred on the balloon to the duodenum model. It is therefore the only control point that moves during the simulation. This is why our algorithm is particularly efficient in this case.

The simulated ultrasound image is computed in real-time from the polygonal structure our algorithm creates and updates, using a 2D ray tracing algorithm. An sample ultrasound image and the corresponding 3D view of the organs are shown in figures 14 and 15.

Figure 14: A view of the 3D model during simulation: the balloon creates a bulge on the duodenum. The black surface through it represents the plane where the echographic image is displayed (below)

5 Conclusion

Mainly by trading memory usage for speed, we managed to develop a method that suits our initial application needs, but that can also be used in more general implicit surface modelling. Future developments include making the algorithm even faster by implementing adaptive tesselation methods and by adapting it to more general implicit surface definitions, in particular more complex control primitives (lines, planes, etc.)

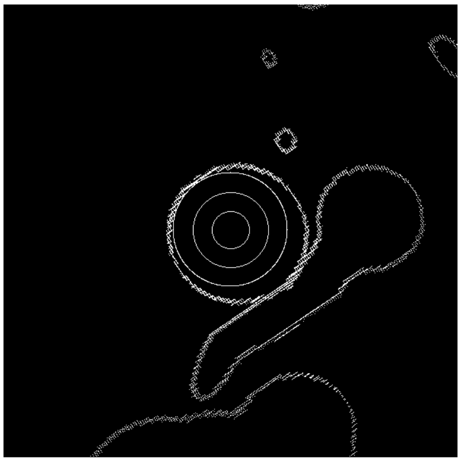

Figure 15: The corresponding echographic image: the outer of the three circles in the middle is the outline of the balloon. Around it is the duodenum and sections of the organs seen above

Among further perspectives for medical applications are surface texture (solid texturing is a simple and effective method for implicit surfaces, but unrealistic for deformable surfaces) and using our method for animation: adding a spring-mass system to the control points to simulate realistic deformation/animation. This is a very important step to simulate realistic organ response to surgical procedures.

References

[1] Chanderjit L. Bajaj. Geometric modeling with algebraic surfaces. In D. Handscomb, editor, *The mathematics of surfaces III*, pages 3–48. Oxford University Press, 1988.

[2] James F. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, 1(3):235–256, July 1982.

[3] Jules Bloomenthal. Polygonalization of implicit surfaces. *Computer Aided Geometric Design*, 5(4):341–355, 1988.

[4] Jules Bloomenthal and Brian Wyvill. Interactive techniques for implicit modeling. In Rich Riesenfeld and Carlo Sequin, editors, *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, volume 24, pages 109–116, March 1990.

[5] Luiz Henrique de Figueiredo, Jonas de Miranda Gomes, Demetri Terzopoulos, and Luiz Velho. Physically-based methods for polygonalization of implicit surfaces. In *Graphics Interface '92*, pages 250–257, 1994.

[6] Mathieu Desbrun, Nicolas Tsingos, and Marie-Paule Gascuel. Adaptive sampling of implicit surfaces for interactive modeling and animation. In *Implicit Surfaces '95 proceedings*, pages 171–185. Eurographics, 1995.

[7] Tinsley A. Galyean and John F. Hughes. Sculpting: an interactive volumetric modeling technique. In *Computer Graphics (Siggraph '91 Proceedings)*, volume 25, pages 267–274. ACM SIGGRAPH, 1991.

[8] Jean-Dominique Gascuel. Implicit patches: An optimized and powerful ray intersection algorithm for implicit surfaces. In *Implicit Surfaces '95 proceedings*, pages 143–159. Eurographics, 1995.

[9] David Jevans, Brian Wyvill, and Geoff Wyvill. Speeding Up 3D Animation For Simulation. *Proc. SCS Conference*, pages 94–100, 1988.

[10] William E. Lorensen and Harvey E. Cline. Marching cubes: a high resolution 3D surface construction algorithm. In *SIGGRAPH '87 Conference Proceedings*, pages 163–169. ACM, 1987.

[11] Paul Ning and Jules Bloomenthal. An evaluation of implicit surface tilers. *IEEE Computer Graphics & Applications*, pages 33–41, November 1993.

[12] E. Varlet, C. Chaillou, P. Dubois, and V. Maunoury. a study of a simulator for ultrasound endoscopy training. *Innovation and Technology in Biology and Medicine*, 18(1):59–69, February 1997.

[13] Luiz Velho. Adaptive polygonalization of implicit surfaces using simplicial decomposition and boundary constraints. In *Eurographics '90*, pages 125–136, 1990.

[14] Andrew P. Witkin and Paul S. Heckbert. Using particles to sample and control implicit surfaces. In *Siggraph '94 Conference Proceedings*, pages 269–277. ACM, 1994.

[15] Geoff Wyvill, Craig McPheeters, and Brian Wyvill. Data structure for soft objects. *The Visual Computer*, 2:227–234, 1986.