Speeding Up Grid-Data Generation for Polygonisation of Implicit Surfaces

Agata Opalach
Steve Maddock

Department of Computer Science, The University of Sheffield,
Regent Court, 211 Portobello Street, Sheffield, S1 4DP, UK
Tel: +44 114 282 5577 Fax: +44 114 278 0972
E-mail: a.opalach@dcs.shef.ac.uk

Abstract

Implicit surfaces are particularly suitable for design and animation of smoothly blended objects that deform during motion. Such objects change their shape due to external forces, deform during collisions with other objects or conform to their environment. They are also able to squash and stretch, an effect essential in traditional animation yet often lacking in computer animation. However, visualisation of implicit surfaces is not yet achieved at interactive rates. This limits the practical application of implicit surfaces in the area of interactive design and animation. One way of displaying implicit surfaces is to polygonise them and use a fast hardware polygon renderer to create a high quality image. This paper proposes a way of speeding up grid-data generation, one of the phases of the polygonisation process.

1 Introduction

Implicit surfaces as a modelling technique were initially described by Blinn (blobby molecules) [2], Nishimura *et al.* (metaballs) [13] and Wyvill *et al.* (soft objects) [20, 21]. All three approaches used scalar fields around 3D points to build their models. The technique was later generalised for arbitrary skeletal elements [4, 5, 22] and has found practical applications in a wide variety of areas. For instance, Tatsumi *et al.* present the use of metaballs in the process of modelling the cod liver [17], Fujita *et al.* represent splashing water using metaballs [7], Muraki uses a blobby model for volumetric shape description (including human face modelling) [12], Max and Wyvill render corals as soft objects [10] and Payne and Toga model a rat's brain using distance fields [16].

Implicit surfaces have also been appreciated in the field of computer animation. In Japan, metaballs have been widely used for modelling and animation of shapes as complex as human anatomy parts [9]. Another use for the technique is in visualising deformable material modelled as particles [11, 19] or mass points [18]. Implicit surfaces have also been used for modelling precise contact between colliding implicitly defined solids [8] and simulating highly deformable material which can break into pieces [6].

Beier at Pacific Data Images commented on the usefulness of blobby primitives for character animation [1]. Our work [14] describes the ABC of implicit surfaces: appearance, blending and coherence. These three properties are important for character animation with implicit surfaces. In [15] we used implicit surfaces to implement cartoon effects (or traditional animation principles) in a character animation system. The growing popularity of implicit surfaces calls for their interactive visualisation.

First let us briefly introduce the terminology of implicit surfaces. Consider a set of *skeletal elements* (points, lines, polygons) with a scalar field function f_i: $\Re^3 \rightarrow \Re$ defined for them. An implicit surface can be defined as $\{P \in \Re^3 | F(P) - Iso = 0\}$ where *Iso* is the isovalue at which an isosurface is extracted and $F(P) = \sum_{i=1}^{n} f_i(P)$. Figure 1 shows an implicit surface created around two points. We will refer to skeletal elements as *primitives*.

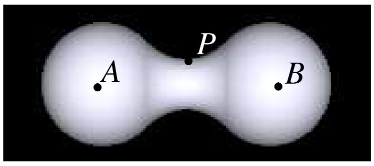

Figure 1: An implicit surface around two skeletal elements, or primitives, A and B.

Two main techniques are used to visualise implicit surfaces. The first is polygonisation, in which an implicit model is converted to a polygonal representation and then a standard polygon renderer is used to render the polygons. The second is ray tracing. Here, ray-surface intersections are computed using the implicit definition of a surface and the intersections are then used by a ray tracing program to render the surface.

Since fast polygon rendering hardware already exists, we shall consider techniques to speed up the polygonisation process. One standard way to polygonise an implicit surface is to divide a 3D space into grid cells. There are three stages in this method: (i) generating grid-data, (ii) identifying grid cells intersecting the implicit surface and (iii) calculating the polygons. In the first stage (i), the scalar field values at the grid points (corners of grid cells) have to be calculated. During stage (ii) all cells which are intersected by the surface (*boundary cells*) (Figure 2) are found and in the stage (iii) polygons in each boundary cell are produced according to the scalar field values in the corners of the cell. Previous attempts have been made to speed up steps (ii) and (iii) of polygonisation [3, 4, 21]. Our experiments are concerned with the first phase of the polygonisation process, the grid-data generation (i).

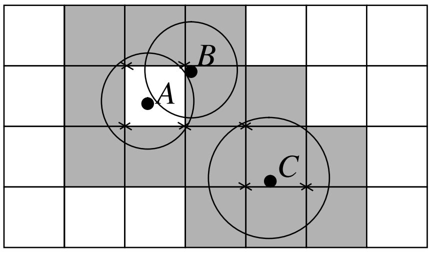

Figure 2: (2D case) Three primitives A, B and C defining an implicit surface in a 2D grid. A circle around each primitive represents the radius of influence. Grid points within each radius of influence are shown as crosses. The boundary cells are shaded in grey.

2 Grid-data generation algorithms

An implicit surface is defined by primitives positioned in space. If we divide the space into grid cells, some grid points will be influenced by a number of primitives (Figure 2), i.e. the primitives give non-zero contributions to the scalar field value at those grid points. We can identify two techniques for generating grid-data: *the naive method* and *the update method*.

2.1 Naive method

The naive method of generating the grid-data is to iterate over all primitives (including non-influencing primitives) and sum the scalar field contribution from them for each grid point. In this simple approach the process of calculating scalar field values at all grid points has to be repeated each time primitives are added, deleted or moved in the scene. When only a few primitives are involved, large parts of the scene remain unchanged, yet all the grid-data is recalculated. We can address such computation inefficiency using the update method.

2.2 Update method

In the update method the initial stage is to store a list of influencing primitives for each grid point. Then, at a given grid point, only primitives influencing it are considered in the calculation of the scalar field value. The update method requires additional data structures to be created and maintained for a scene. Once the data structure is created, it has to be updated for each change occurring in the scene. There is no need to recalculate the parts of the scene which have not been altered. The primitives added, deleted or moved have to be inserted, deleted or moved to the influence lists of relevant grid points and the field value at the affected grid points has to be recalculated.

We will compare the two methods of grid-data generation in two experiments: (i) generating a static image of a scene and (ii) generating the frames of an animation sequence.

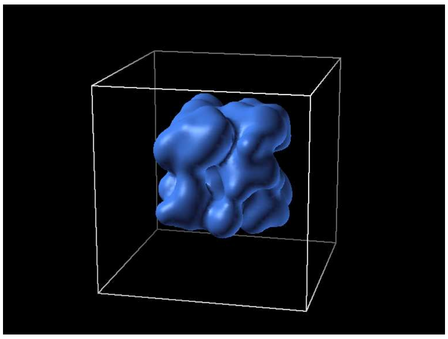

Figure 3: Static image test: 128 primitives, radius 25.

3 Static image experiment

The first experiment was designed to compare the performance of the two grid-data generation techniques in the case of polygonising a single image of a static scene.

3.1 Test sets

The test data for the static image experiment consisted of a randomly generated cluster of primitives (each of the same radius) in the middle of a cubic world (Figure 3). The world dimensions were 100x100x100 units and primitives were randomly placed in the 50x50x50 units cube in the centre of the world. For each test case the number of primitives in the scene and their radius were independently varied. In one set of tests the number of primitives was fixed at 32 and their radius was varied from 10 to 100 (Table 1 in the Appendix). In the second set, the number of primitives was fixed at 128 (Table 2 in the Appendix). The time required for generating grid-data for both grid-data generation methods was measured. For the update method, the time required for creating the additional data structure was also added. Ten tests in each configuration were run and the results averaged.

3.2 Results and Discussion

We expected that the update method would be considerably faster than the naive method. However, the results presented in Figure 4 for 32 and 128 primitives show that the update method only works better for small (radius 10-50) and fewer (32) primitives. When the size of primitives gets larger (close to the size of the world), the update method proves to be very inefficient.

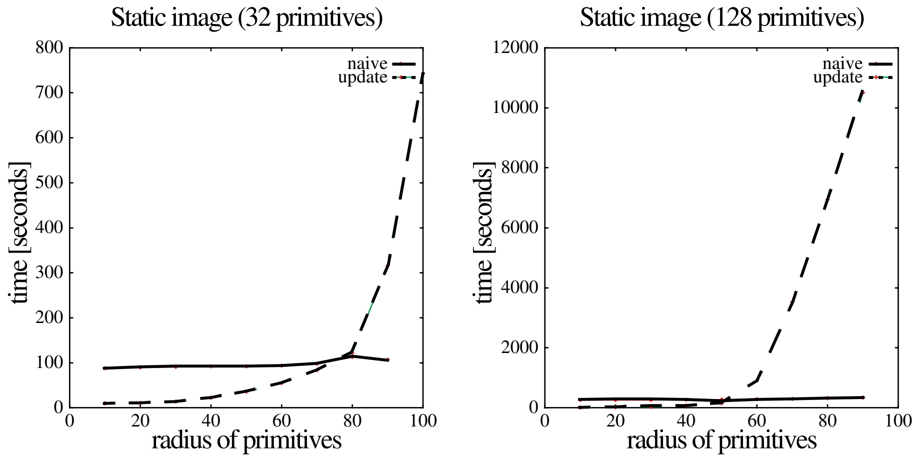

Figure 4: The results of the static image experiment: The plots show the time in seconds of grid-data generation for the scene from Figure 3 for varying radius of primitives in two cases: 32 and 128 primitives

The naive method has a constant cost which depends only on the number of primitives in the scene. The update method depends on the radius of primitives in the scene: the larger the primitives, the more grid points they influence and therefore the longer the time required to create the additional data structure. In a practical situation most of the primitives in a scene would probably be small in relation to the world size. In these cases using the update method for grid generation would reduce the computation time.

4 Animation experiment

The animation experiment was designed to compare the two techniques during the creation of several frames in an animation sequence. In this case the update method takes advantage of the fact that only some of the primitives move during each animation step (Note: all primitive can move in the worst case).

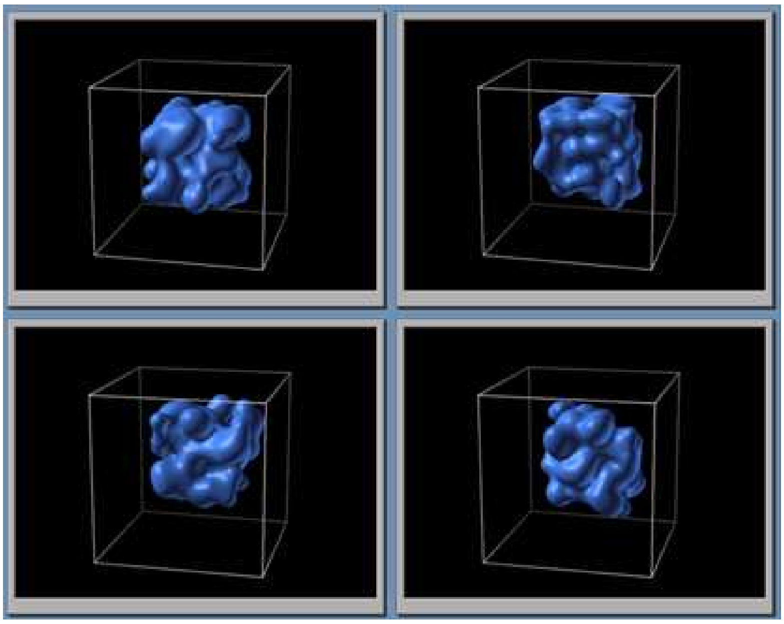

Figure 5: Animation test: 128 primitives, radius 25, randomly moving 100 primitives

4.1 Test sets

The animation experiment used the same scene as in the static image experiment. For each animation step a number of primitives in the scene were moved in random directions. The translation vector had a random length within the range from one to two grid cells. Figure 5 shows frames from the tested animation sequence. For each test case the number of primitives in the scene, their radius and the number of primitives moving during the animation were independently varied. The number of primitives changed from 32 to 128, their radius from 10 to 60 and the number of moving primitives from 10 to the total number of primitives in the scene. In each test ten frames of animation were created and the average time for grid-data generation for one frame was calculated. In the update method the time for creating and updating the secondary data structure was also added to the total time. Each test was run ten times and the results averaged.

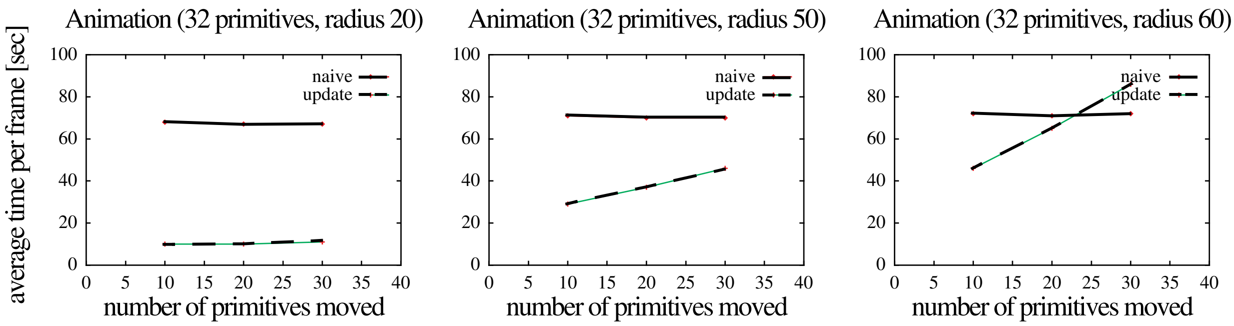

Figure 6: The results of the animation experiment: The plots show the average time in seconds of polygonising one frame of animation from Figure 5 for the radius of primitives 32 and a varied number of moved primitives

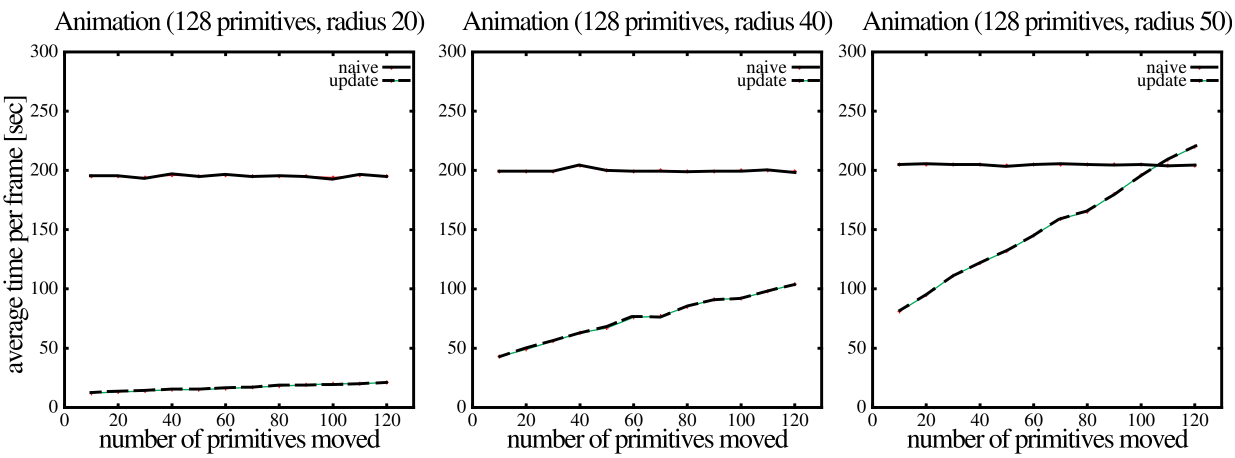

Figure 7: The results of the animation experiment: The plots show the average time in seconds of polygonising one frame of animation from Figure 5 for the radius of primitives 128 and a varied number of moved primitives

4.2 Results and Discussion

Figures 6 and 7 and Tables 3-8 in the Appendix show the results of the animation experiment for 32 and 128 primitives respectively. The plots show how the computation time changes depending on the number of primitives moved in an animation step. The plots represent different radii of the primitives in the scene. The update method proves more efficient in most cases. However, when the radius of primitives in the scene grows larger (radius 50+, world size 100x100x100) and nearly all primitives in the scene are moved, the naive method works better.

5 Conclusions

We have run a series of tests comparing two methods of grid-data generation for polygonising implicit surfaces: the naive method and the update method. The update method gives a significant reduction in computation time for smaller primitives (half the size of the world and smaller) for both the static image and animation experiments. For larger primitives (larger than half the size of the world) the naive method appears to be more efficient since it has a constant cost for all cases. The main advantage of the update method is shown during animation, when it uses the fact that only some primitives in the scene move. Hence, only grid-data in those areas have to be recalculated. However, if almost all primitives in the scene are moved, the update cost is high. For practical purposes there will usually only be small (in relation to the size of the world) primitives in the scene and only some of them will be moving in each animation step. In these cases the update method considerably speeds up the grid-data generation process.

The use of implicit surfaces for modelling and animation is growing in popularity. Speeding up their display is important for interactive applications. For instance, a designer or an animator have to view their creations in real time in order to be able to use implicit surfaces for interactive design or creating computer animation. When interactively sculpting an object an artist or designer is likely to change only one part of it at a time. Similarly an animation sequence will usually consist of a static background and only a few objects moving. In these cases the update method of grid-data generation can accelerate the visualisation process.

6 References

1. Beier, T. “Practical Uses for Implicit Surfaces in Animation“, Siggraph 90 Course Notes No 23, “Modelling and Animating With Implicit Surfaces”, pp 20.1-20.11
2. Blinn, J. F., “A Generalisation of Algebraic Surface Drawing”, ACM Trans. Graphics, Vol. 1, No 3, July 1982, pp 135-256
3. Bloomenthal, J. “Polygonisation of Implicit Surfaces”, Computer Aided Geometric Design, Vol. 5, No 4, November 1988, pp 341-355
4. Bloomenthal, J. and Wyvill, B., “Interactive Techniques for Implicit Modelling”, Computer Graphics, Vol. 24, No 2, 1990, pp 109-116
5. Bloomenthal, J. and Shoemake, K., “Convolution Surfaces”, Computer Graphics, Vol. 25, No 4, July 1991, pp 251-256
6. Desbrun, M. and Gascuel, M-P., “Highly Deformable Material for Animation and Collision Processing”, Proc. 5th Eurographics Workshop on Animation and Simulation, Oslo, September 94.
7. Fujita, T., Hirota, K. and Murakami, K., “Representation of Splashing Water using Metaball Model”, Fujitsu, 1990, Vol. 41, part 2, pp 159-165, in Japanese
8. Gascuel, M.-P., “An Implicit Formulation for Precise Contact Modelling between Flexible Solids”, Proceedings of SIGGRAPH 93 (Anaheim, California, August 1-6, 1993). In Computer Graphics Proceedings, Annual Conference Series, 1993, ACM SIGGRAPH, New York, pp 313-320
9. Graves, G., “The Magic of metaballs”, Computer Graphics World, May 1993, pp 27-32
10. Max, N. L. and Wyvill, B., “Shapes and Textures for Rendering Coral”, Scientific Visualisation of Physical Phenomena, N. M. Patrikalakis (Ed.), Springer-Verlag 1991, pp 333-343
11. Miller, G. and Pearce, A., “Globular Dynamics: A Connected Particle System for Animating Viscous Fluids”, Vol. 13, No 3, 1989, pp 305-309
12. Muraki, S., “Volumetric Shape Description of Range Data Using “Blobby Model”Model””, Computer Graphics, Vol. 25, No 4, July 1991, pp 227-235
13. Nishimura, H., Hirai, M., Kawai, T., Kawata, T., Shirakawa, I. and Omura, K., “Object Modelling by Distribution Function and a Method of Image Generation”, The Transactions of the

Institute of Electronics and Communication Engineers of Japan, 1985, Vol. J68-D, Part 4, pp. 718-725, in Japanese, translated into English by Takao Fujuwara
14. Opalach, A. and Maddock, S., “Implicit Surfaces: Appearance, Blending and Consistency”, Proc. 4th Eurographics Workshop on Animation and Simulation (Barcelona, 4-5th September, 1993), 233-245.
15. Opalach, A. and Maddock, S., “Disney Effects Using Implicit Surfaces”, Proc. 5th Eurographics Workshop on Animation and Simulation (Oslo, 17-18th September, 1994)
16. Payne, B. A. and Toga, A. W., “Distance Field Manipulation of Surface Models”, IEEE Computer Graphics and Applications, January 1992, pp 65-71
17. Tatsumi, H., Takaoki, E., Omura, K. and Fujito, H., “A new method for 3D reconstruction from serial sections by computer graphics using “meta-balls”: reconstruction of “Hepatoskeletal System” formed by Ito Cells in the Cod Liver”, Computers and Biomedical Research, 1990, Vol. 23, Part 1, pp 37-45
18. Terzopoulos, D., Platt, J. and Fleisher, K. “Heating and melting deformable models (from goop to glop)”, Graphics Interface’89, June 1989, pp 219-226
19. Tonnesen, D., “Modelling liquids and solids using thermal particles”, Proceedings of the Graphics Interface 1991, pp. 255-262
20. Wyvill, G. and McPheeters, C. and Wyvill, B., “Data Structure for Soft Objects”, The Visual Computer, Vol. 2, No 4, August 1986, pp 227-234
21. Wyvill, G. and McPheeters, C. and Wyvill, B., “Animating Soft Objects”, The Visual Computer, Vol. 2, No 4, August 1986, pp 235-242
22. Wyvill, B. and Wyvill, G., “Field Functions for Implicit Surfaces”, The Visual Computer, Vol. 5, 1989, pp 75-82

Appendix. Experimental Results

Static Image Experiment

method / **radius**	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**
naive	88	90	91	92	92	94	98	113	107
update	10	11	14	22	36	55	84	122	316

TABLE 1. Static image experiment results [time] (32 primitives)

method / **radius**	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**
naive	262	263	265	267	274	289	299	326	345
update	10	16	32	71	145	912	3532	6936	10501

TABLE 2. Static image experiment results [time] (128 primitives)

Animation Experiment

method / **moved**	**10**	**20**	**30**
naive	68	67	67
update	10	10	11

TABLE 3. Animation experiment results [time] (128 primitives, radius 20)

Animation Experiment (cont.)

method \ moved	10	20	30
naive	71	70	70
update	29	37	46

TABLE 4. Animation experiment results [time] (128 primitives, radius 40)

method \ moved	10	20	30
naive	72	71	72
update	46	65	86

TABLE 5. Animation experiment results [time] (128 primitives, radius 50)

method \ moved	10	20	30	40	50	60	70	80	90	100	110	120
naive	195	195	194	196	195	196	195	195	195	194	196	195
update	129	13	14	15	15	16	17	18	19	20	20	21

TABLE 6. Animation experiment results [time] (128 primitives, radius 20)

method \ moved	10	20	30	40	50	60	70	80	90	100	110	120
naive	199	199	199	204	200	199	200	199	199	200	200	199
update	43	49	56	63	67	76	77	85	91	92	98	104

TABLE 7. Animation experiment results [time] (128 primitives, radius 40)

method \ moved	10	20	30	40	50	60	70	80	90	100	110	120
naive	205	205	205	205	204	205	205	205	205	205	204	204
update	81	95	111	122	132	145	159	165	180	196	209	220

TABLE 8. Animation experiment results [time] (128 primitives, radius 50)