Subdivision Surfaces for Fast Approximate Implicit Polygonization

Brian Wyvill and Pauline Jepp
Department of Computer Science
University of Calgary*

Kees van Overveld
Philips Research &
Dept. of Mathematics and Computing Science
Eindhoven University of Technology†

Geoff Wyvill
Dept. of Computer Science
University of Otago‡

October 30, 2000

Abstract

We propose a fast method of generating an approximate polygonal mesh from an implicit surface. Current workstations are still not capable of producing polygon meshes fast enough for the interactive modelling of complex implicit models. We propose a hybrid method that combines current implicit polygonization techniques with the faster sub-division surface technique applied as a post-process to smooth the implicit mesh. In an interactive enviroment the smooth mesh points can be migrated to the implicit surface, in idle moments when the user is not interacting with the model. The technique can be further enhanced by providing tools for the user to indicate areas of interest that can be preferentially smoothed.

Keywords Interactive modelling, implicit surfaces, polygonization, sub-division surfaces.

1 Introduction

Since the early 1980's implicit modeling has gained in importance as a useful modelling technique. The cartoon like models of Blinn [Blinn 82], Wyvill [Wyvill 86] and Nishimura [Nishimura 85] have been replaced with the more sophisticated models which combine CSG and implicit blending techniques, see [Pasko 95] and [Wyvill 99]. The big advantage of implicit techniques over parametric, is that very complex shapes can be described extremely quickly using a skeletal model, [Bloomentha 97]. One of the big drawbacks to implicit modelling is the time taken to visualize such complex models. Although it is possible to convert these models to polygons using a variety of techniques [Ning 93] and [Wyvill 96], these methods are not fast enough for the visualization of complex models. Particle methods, such as [Witkin 94] produce an impression of the surface in a shorter time than a full polygonizer, but still require a further step to produce a mesh, so that for complex models the method is still relatively slow for an interactive modelling system.

Apart from speed there are other important quality issues, voxel based methods can produce ambiguous tilings ([Ning 93]) and too many or poorly shaped triangles. Adaptive methods are either too slow ([Bloomentha 88]) or are not able to tile certain topological cases [Overveld 93].

Sub-division surfaces [Doo 78]) can also be visualized as a smooth polygonal mesh. Whereas an implicit surface tiler has to make many function evaluations to find a point on the implicit surface, the sub-division surface technique contains no such search mechanism. Instead each iteration produces a new mesh which becomes smoother at each iteration. Sub-division surface techniques can be used to approximate C^1 [Hoppe 94] or even C^2 curved surfaces [Biermann 00], but this behaviour emerges from the sub-division rules, the procedure is faster than implicit surface polygonization as it avoids the implicit search mechanism.

In this paper we first of all give overviews of our implicit polygonization and sub-division surface techniques. This is followed by the details of our method along with some preliminary results and a discussion on the future directions of the project.

* Address: 2500 University Drive N.W., Calgary, Alberta,Canada, T2N 1N4 Email: [mtigges|blob]@cpsc.ucalgary.ca
† Address: P.O.Box 513, 5600 MB, Eindhoven, The Netherlands. Email: wsinkvo@info.win.tue.nl
‡ Address: P.O.Box 56, Dunedin New Zealand Email: geoff@cs.otago.ac.nz

2 Polygonization

In this section we examine our implicit surface polygonization technique and review the reasons why it does not operate at interactive rates. We have tried several polygonization algorithms, the first [Wyvill 86] has a number of advantages and we chose this algorithm as the starting point for the current method. The scheme divides the world into a mesh of cubic voxels, and tests selected voxel vertices by finding the implicit value of the vertex P, in question and testing against the implicit funciton, $f(P) = 0$. Various voxels are selected as seed voxels and if any edge is found that has a vertex inside and a vertex outside the surface, it is assumed that a piece of the surface intersects that edge. The algorithm continues by visiting the surrounding voxels and checking for intersections. Voxels are marked when they are visited and the algorithm terminates when there are no more unvisited voxels waiting to be checked.

This algorithm has the advantage that it assumes nothing about the topology of the scene, it produces a consistent mesh without the mesh ambiguity problems of [Lorensen 87]. The user can control the size of the voxels, it is easy to implement and reasonably fast. What slows the algorithm down is that a large number of implicit function evaluations have to be done. Once a voxel edge is found to intersect the surface, then some kind of root finding algorithm must be applied to iterate to the intersection point. This involves several implicit function evaluations. Also, the normals must be calculated at polygon verttices for rendering purposes and with black box implicit functions, for which the gradient cannot be found analytically, a numeric technique must be adopted, again involving further implicit function evaluations, (see [Bloomentha 88]). The dominating cost of the uniform space subdivision algorithm is the number of implicit function evaluations that must be done to produce a mesh.

3 Sub-division Surface Scheme

A lot of interest has recently been shown in subdivision surface schemes. The idea is to take an initial polygon mesh and smooth the mesh by repeatedly cutting corners. The rules according to Catmull/Clark [Catmull 78] and Sabin/Doo, [Doo 78] produce surfaces that can be proven to converge to a piecewise B-spline where the original vertices are the control points. We use the Sabin/Doo scheme in our algorithm, it is considerably faster than implicit surface polygonization, requires no implicit function evaluations, but the resulting surface is not necessarily close to $f(p) = 0$.

4 The Hybrid Scheme

We identify two types of actions that take place in the formation of a polygonal approximation of an implicit surface:

- mesh subdivision
- vertex adjustment

From an initial mesh derived from the implicit polygonizer, the recursive subdivision provides a smoothed mesh at interactive rates. The vertices will not necessarily be on the implicit surface so the basic idea is to adjust the vertices during idle moments when the user is not interacting with the model.

4.1 Toplogical Issues

At places of high curvature, small triangles are needed in order to give sufficient visual accuracy. However, mesh subdivision does not alter the topological genus of the surface, so before mesh subdivision can start, we have to make sure that an initial mesh exists that is topologically correct (i.e. topologically equivalent with the final mesh).

As noted in section 2, the uniform space sub-division scheme, samples space at the vertices of a uniform, cubic voxel grid. This technique does not assume any information about the topology or curvature of the surface. Thus features that are so small they fit within a voxel will be missed. If the size of the voxel grid is too big, topological features can also be mistaken. For example the two spheres (circles) in Figure 1 will appear connected, since all samples taken are within the two spheres. At a higher sampling density the spheres will appear disconnected.

In the method presented here we assume that the user will choose the sampling density interactively, altering the sampling grid to suit the topology the user wishes to model. The user can then describe a skeletal implicit surface model (as described in [Bloomentha 97]) and choose to show the implicit surface as a smooth polgygonal mesh. The user can control the quality of the mesh in a number of ways. Firstly an appropriate sampling density must be chosen. As indicated above the sampling density of the initial grid will determine the topology of the displayed model.

The lower the density of the initial mesh the faster the system will respond, but the more likely it is that desired features may be missed, so the user will choose an appropriate option.

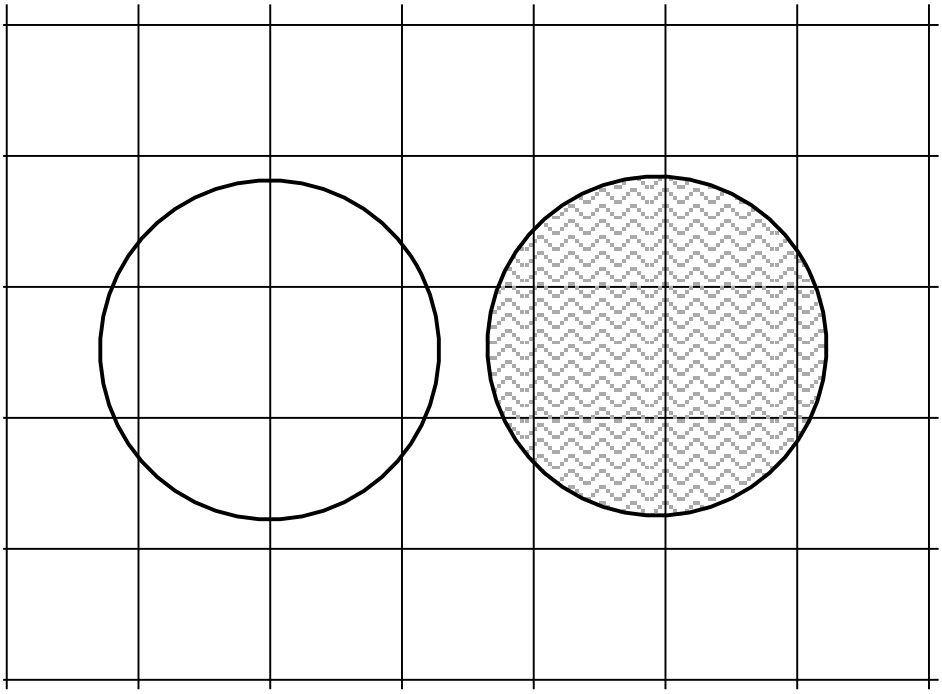

Figure 1: Slice through a sampling grid, with two implicit spheres.

4.2 Subdivision Issues

There are four situations when vertices recieve location-data (i.e. x, y, z coordinates):

1. During the formation of the initial mesh
2. Newly created vertices that result from mesh subdivision have to have a first location
3. Vertices that have received a first location in 2 above, may have to be moved either because the resulting mesh is not smooth, and/or because they are not in the solution of $f(x.y.z) = 0$
4. As a result of user interaction, the function $f(x, y, z)$ is modified, and the mesh should adjust accordingly. In this case we assume that the modifications do not imply topological changes.

For the above types of actions, we see a number of different algorithms:

4.2.1 Algorithm A - Simple Subdivision

Mesh subdivision according to some suitable subdivision scheme. The newly introduced vertices get an arbitrary (provisional) location, for instance in the centroid of the triangle that was subdivided.

4.2.2 Algorithm B - Doo-Sabin

Mesh subdivision according to the corner cutting scheme of Catmull/Clark and Sabin/Doo. Here the provisional location of the newly created vertices is such that the resulting surface can be proven to converge to a piecewise B-spline where the original vertices are the control points.

4.2.3 Algorithm C - Root Finding

Vertex motion as a result of an iterative solution of $f(x, y, z) = 0$. Let p be an abbreviation for (x, y, z), then we can use a Newton Raphson approach assuming an initial guess for p exists, say p', such that $f(p') = a$ and a is not too large; then we write $p = p' + \delta_1$, and an approximation for δ_1 follows from setting $\delta_1 = lambda * gradf(p')$; 1st order Taylor expansion and solving for lambda gives $lambda = -a/|\nabla f(p').\nabla f(p')|$, so $\delta_1 = -a\nabla f(p')/|\nabla f(p').\nabla f(p')|$.

4.2.4 Algorithm D - Relaxation

Vertex motion as a result of local relaxation. Let p have neighbour vertices $p_i, i = 1, 2, ..., n$. Then we declare the environment of p smooth if p is in the centroid of the p_i. In order to increase the smoothness we compute a correction vector δ_2 that is to be applied to p (and for each of the p_i, there are also correction vectors that are computed according to the same scheme). We set $pc = \sum_{i=0}^{n} p_i$, and $\delta = pc - p; next, \delta_2 = (n * \delta - \sum_{i=0}^{n} \delta_i)/(n+1); \ here, \ the \ \delta_i = pc_i - p_i, \ and \ pc_i$ is the centroid of the neighbours of p_i. It can be seen that the assignment $p < -- p + \delta_2$ increases the smoothness round p, whereas it leaves the centroid of the cluster $(p, p_1, p_2, p_3, ...p_n)$ in tact. The computed δ_2 vectors are only applied after they have all been computed. For a subset of points p, we may decide not to apply the adjustment: these points are left unaffected.

4.2.5 Observations on the algorithms

We observe the following:

- only algorithm C drops points p onto the implicit surface. However, C does not control the motions of points *in* the implicit surface. A possible effect of C therefore could be that the mesh gets unevenly sampled (very skinny triangles may result, or even triangles may be turned 'inside out'). Also this is the only algorithm that requires the implicit function and its gradient and is therefore slow.

- algorithm D assures the surface not only to be smooth, but also to be evenly sampled. If all mesh points are allowed to move freely, algorithm D strives for configurations where every vertex point is in the middle of a regular n-gon. However, algorithm D has no knowledge about the function $f(x, y, z)$, and there is no guarantee that regions that result from repeated application of algorithm D are on the implicit surface.

- in areas where the iso surface $f(x, y, z) = 0$ is smooth (for instance, far from C^1 discontinuities as may arise from non-differential operators in f such as MAX or MIN[1]), the result of applying algorithm C and the result from applying algorithm D will be to a large extent indistinguishable. Nevertheless, application of one iteration from D is expected to be much cheaper than applying one Newton Raphson iteration of C (even if we suffice with using non-updated estimates for $\nabla f(p')$ for some iterations). Therefore, algorithm D can be seen as a cheap predictor, whereas C is a corrector. But in many places (namely, far from C^1 discontinuities), the predictor is assumed to be sufficiently accurate to completely leave out the corrector.

- also algorithm D, being an iterative algorithm, needs an initial estimate for the vertices p. Algorithm B is suitable to play this role.

4.3 The Hybrid Algorithm

The following algorithm makes the use of all of the techniques, algorithms A-D described above. For the moment we describe the hybrid algorithm without making use of user interaction.

The first step is to construct an initial mesh, the mesh should be sufficiently dense so that it captures all topological peculiarities and shape features (such as small protrusions) that otherwise might be missed as a result of of the algorithm below, but it does not necessarily have to capture all C^1 discontinuities. This could be done as an option chosen by the user. The mesh vertices that result are all on the implicit surface. So, in every triangle, the three corner vertices p_j satisfy $f(p_j) = 0$. label all vertices with ('ON','UNKNOWN'). The attribute 'ON' (as opposed to 'MAYBE') means that this vertex satisfies $f(p) = 0$. The attribute 'UNKNOWN' (as opposed to 'SMOOTH') means that we are not sure if this vertex is in a smooth area. For instance, it could be that there is a C^1 discontinuity nearby. In these areas, we will have to apply algorithm C for every new vertex that is created. However, vertices that result from splitting a triangle that has all three vertices 'SMOOTH' are assumed to be sufficiently approximated by algorithm D.

```
READY=false;
WHILE (not READY) {
     READY=true;
     FOR (all triangles T) {
         IF(T is sufficiently small or the angles with its
            neighbour triangles are sufficiently close to 180 degrees)
                  label T as 'OK';
```

[1] We use MAX and MIN in the *BlobTree*for defining CSG operations homogeneously with belnding see [Wyvill 99]

```
        ELSE     label T as 'NOT_OK';
    }
    FOR (all triangles T that are NOT_OK) {
        READY=false;
        IF(all three T's edge-sharing neighbours are also NOT_OK)
           subdivide T with algorithm B;
        ELSE subdivide T with a subdivision scheme according to
             algorithm A that does
             not affect those of T's edge-sharing neighbours
             that are OK;
        IF(T's vertices are all labeled ('ON','SMOOTH'))
           label new vertices ('MAYBE','SMOOTH');
        ELSE label new vertices ('MAYBE',"UNKNOWN');
    }
```

The first part of the algorithm above has created new vertices, that are not on the surface, amidst a collection 'old' vertices that are on the surface. Those new vertices that resulted from algorithm B are probably not too far from the surface, unless the surface has a local C^1 discontinuity or another non-smooth feature. Next we have to get all the new vertices on the surface. For the ('MAYBE','SMOOTH') ones, this can be done with algorithm d. For the ('MAYBE','UNKNOWN') ones we can also apply algorithm D first, but then we have to apply algorithm C in order to check (and possibly correct) if they satisfy $f(p) = 0$. Apply a limited number of iterations of algorithm D to all vertices; only apply the displacement vector δ_2 to those vertices labeled ('MAYBE',...);

The second part of the algortithm proceeds as follows:

```
    FOR(all vertices p labeled ('MAYBE','UNKNOWN')){
        evaluate f(p);
        IF(f(p) is close to 0) {
            label p as ('MAYBE','SMOOTH');
        /* We know that p is near the implicit surface (but probably not
           on it, and it does no harm to allow it to take part in
           future runs of algorithm D). It got close to the implicit surface
           without us forcing it by means of algorithm C, so apparently,
           the environment is smooth so that algorithm D gives
          the correct result. */


        relabel p's neighbours that are labeled ('ON','UNKNOWN') as ('ON','SMOOTH');

        /* we do this relabelling for efficiency reasons. Indeed, due to lack of
           knowledge about the implicit surface, the vertices from the initial mesh were
           all conservatively labeled ('ON','UNKNOWN'), but many of them deserve a label
           ('ON','SMOOTH'). Maintaing the conservative 'UNKNOWN' assignment will cause
           many unnecessary function-evaluations in the future, so as soon as we know
           that p is in a SMOOTH area, we assume that the triangle from which p originat
           as a result of subdivision, is also entirely 'SMOOTH'. A similar argument
           applies to other vertices than the ones from the initial mesh that received
           'UNKNOWN': some of these also will be converted later on to SMOOTH. */
        }
        ELSE { apply algorithm c;
              /* enforce p down to the surface */
               label p as ('ON','UNKNOWN');
              /* p is on the surface, but we may have to do function evaluations in its
                 neighbourhood for new vertices to be created later */
        }
    }
}
```

At the end of this algorithm all triangles are sufficiently small and the mesh looks smooth, but not all vertices are really on the surface. Only the vertices labeled ('ON','...') are guaranteed on the surface. For all other vertices, we may want to spend additional computational effort for instance during idle time in an interactive application, to execute algorithm C.

4.4 User Interaction

The above algorithm assumes no user interaction, but three forms of user interaction can be easily interwoven. These interactive versions require visiting all the vertices in the same order as they were introduced, so the data structure that represents the mesh should also record a rank number for all vertices.

4.4.1 Modifications To the Implicit Function

The user might initiate small modifications of the implicit function. All vertices should be labeled ('MAYBE','UNKNOWN'), and next in the oldest vertices (=the vertices that made up the initial mesh), alorithm C should be executed. These oldest vertices receive ('ON','UNKNOWN'). Next, the entire algorithm as outlined above is executed, except that most of the subdivisions are not actually performed, because the resulting vertices already occur in the data structure. Nevertheless, it might be that there are some triangles NOTOK that were OK in the first run; these triangles need subdivision. As a result of multiple modifications, it might be that the mesh becomes too dense: many unnecessary triangles will result. Standard mesh decimation algorithms (see [Hoppe 93], [Lindstrom 98]) may be used to periodically 'clean' the mesh.

4.4.2 Additional Smoothing

The user might indicate some regions where additional smoothing should take place: possibly to improve the distribution of vertices over the mesh. In these regions, temporarily the 'ON' label should be replaced by 'MAYBE'. Indeed, as we said earlier, algorithm C causes vertices to land on the surfaces, but it does not guarantee an even distribution of the vertices over the surface. After some iterations of algorithm D, the vertices that originally were 'ON' should have algorithm C executed and reset their label to 'ON'. Finally, some iterations of algorithm D to the 'MAYBE' vertices smooths minor visual artifacts in the neighbourhood of 'ON' vertices that are surrounded by vertices that are not on the surface.

4.4.3 Early Smoothing

The user might indicate regions were the label 'SMOOTH' was set too early. For instance, a small protrusion in the middle of an otherwise smooth area may get overlooked in the course of the subdivision algorithm. In these areas, the 'SMOOTH' value should be replaced by 'UNKNOWN', and algorithm C should be invoked deliberately. Again, some iterations of algorithm D may be useful to redistribute the vertices over the surface to improve the uniformity of the mesh.

5 Results and Implementation

Our implementation does not currently include all of the algorithm detailed in 4.3. So far we have implemented algorithms A and C. Thus we produce triangles that are not even in size and the vertices will not in the first instant lie on the surface. Idle moments are used by algorithm C to move the points to the surface, not an ideal situation as artifacts as mentioned above could result, ie very skinny triangles, or even triangles that may be turned 'inside out'. Our implementation does give us some preliminary results that indicate that the full algorithm is worth pursuing.

The initial grid resolution for the implicit polygonizer provides the polygon mesh which is then smoothed at various levels of subdivision by the Doo-Sabin algorithm. The number, labelled as grid resolution (4,8,16,32) indicates the number of voxels on a side of the grid. Table 1 and table 2 shows various statistics and the images are shown in the corresponding figures, Figure 2 and Figure 3.

It can be seen from the tables that for all the examples the time for subdivision plus the time taken to find the initial polygonization is considerably smaller than the time to generate an equivalent number of polygons by polygonization alone. For each point, P, an error value is found by calculating $(f(P) - f(0))/||\nabla(p)||$ The error is then averaged over the surface. It can be seen that error in the subdivision surface decreases as the number of initial voxels increases (as expected).

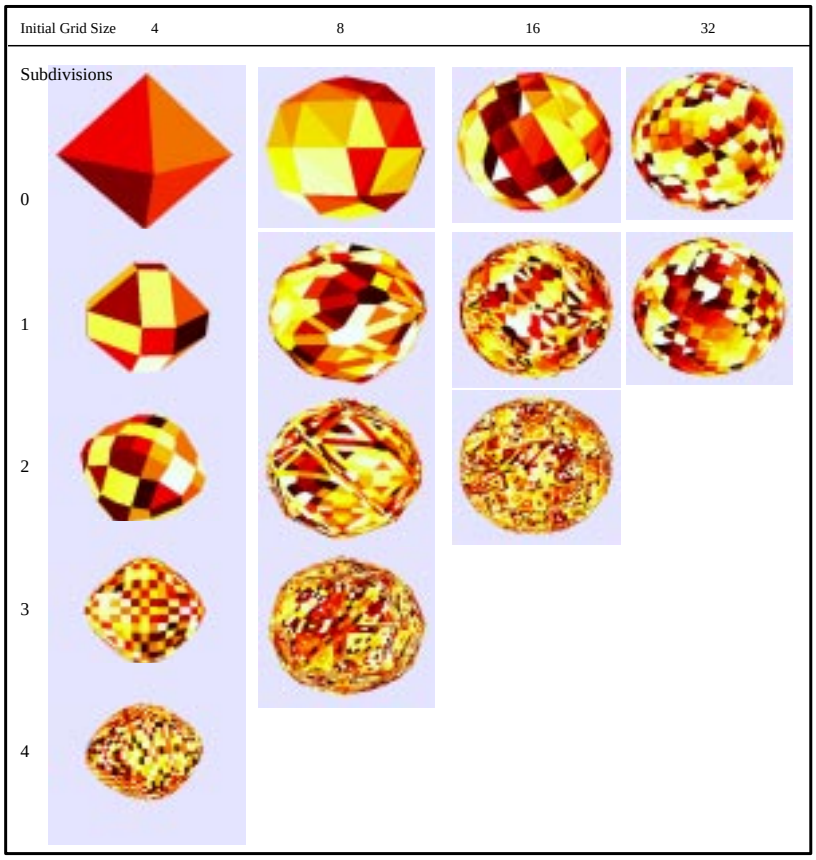

Figure 2: Spheres at various initial grid sizes and levels of subdivision.

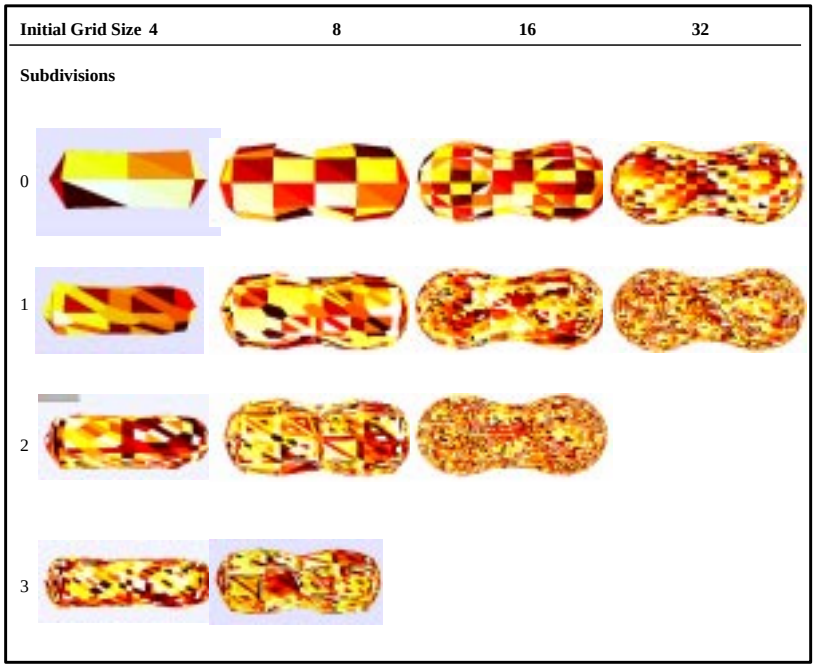

Figure 3: Blended Spheres at various initial grid sizes and levels of subdivision.

Initial Grid Resolution	4				8			16		32
Time to Polygonise	0.43				0.47			0.65		1.50
Number of Faces	8				104			536		2024
Average func value error	0.014				0.008			0.003		0.001
Number of Subdivisions	1	2	3	4	1	2	3	1	2	1
Time to subdivide	0.01	0.06	0.08	0.38	0.19	0.45	1.99	0.41	2.31	1.57
Number of Faces	26	98	385	1538	434	1931	8128	2325	10272	8688
Average func value error	0.130	0.175	0.186	0.189	0.010	0.012	0.013	0.004	0.005	0.001
Total time	0.44	0.49	0.51	0.81	0.66	0.92	2.46	1.06	2.96	3.07
Time per face (t/f)	0.017	0.005	0.0001	0.00052	0.0015	0.00047	0.0003	0.00046	0.00029	0.0003

Table 1: Sphere Polygonisation and Subdivision Times.

Initial Grid Resolution	4				8			16		32
Time to Polygonise	0.43				0.47			0.65		1.50
Number of Faces	24				168			680		2904
Average func value error	0.18				0.068			0.033		0.013
Number of Subdivisions	1	2	3	4	1	2	3	1	2	1
Time to subdivide	0.01	0.10	0.38	1.67	0.15	0.86	3.42	1.49	4.26	1.66
Number of Faces	117	511	2138	8758	891	4092	17535	3613	16531	15332
Average func value error	0.735	0.906	0.938	0.945	0.165	0.183	0.186	0.065	0.071	0.014
Total time	0.44	0.54	0.81	2.10	0.62	1.34	3.89	2.14	4.91	3.16
Time per face (t/f)	0.004	0.001	0.0004	0.00024	0.0007	0.00033	0.0002	0.0006	0.0003	0.0002

Table 2: Blended Sphere Polygonisation and Subdivision Times.

6 Future Work

The next step is to implement the full hybrid algorithm, however there are a number of ways in which user interaction could be used where it is necessary to give the visualization software some help. An automatic way of calculating a reasonable initial grid size would make user interaction somewhat simpler. This could be done using the techniques described in [Nuij 96], however even if the topology could be determined by calculating the critical points, a grid size that is too small will take too long to process and not keep up with the user interaction. There are many possible variations to this technique. It may be possible to smooth the parts of the inital mesh that are the least smooth. For example voxels with widely differeing implicit values at the vertices could be sub-divided then offered to the corner cutting algorithm. It is important to choose the 'right' places for the initial evaluations. In particular, these should be near 'critical' points, for example, sharp corners etc. A future polygonizer might enhance the standard implicit surface formula with optional user's hints. This technique has been used for font descriptions, a merge of Bezier-spline control points and hints; these hints are used during scanconverting the fonts to low resolutions, where every pixel should be rounded the right way. In the case of implicit surfaces, optional 'hints' could be, for example, (x, y, z) points where the evaluator should generate initial polygons.

It might be problematic to guarantee proper time consistent behavior when an implict object moves during animation. Maybe this could be improved somewhat using the same hinting mechanism, if the hint-regions moves in accordance with the object.

7 Conclusion

We have proposed a hybrid algorithm which combines implicit surface polygonization with subdivision surface techniques as well as a smoothing algorithm using a relaxation technique and an algorithm for moving vertices onto the implicit surface. So far we have tested parts of this algorithm on some simple implicit models and found that using a low grid size for an initial polygonization and smoothing the result with the Doo-Sabin algorithm, can save considerable processing time. The main conclusion is that it is worth investigating this approach further to achieve a more efficient polygonization algorithm for complex objects.

8 Acknowledgements

The authors would like to thank the many students, past and present who have contributed towards the implicit modeling research project at the University of Calgary. In particular Callum Galbraith and Mark Fox. We would also like to thank the department of computer science and the MACI project for the use of a large cluster of workstations for high performance computing. This work is partially sponsored by the Natural Sciences and Engineering Research Council.

References

[Biermann 00] Henning Biermann, Adi Levin, and Denis Zorin. Piecewise smooth subdivision surfaces with normal control. *Proceedings of SIGGRAPH 2000*, pages 113–120, July 2000. ISBN 1-58113-208-5.

[Blinn 82] James Blinn. A Generalization of Algebraic Surface Drawing. *ACM Transactions on Graphics*, 1:235, 1982.

[Bloomentha 88] Jules Bloomenthal. Polygonisation of Implicit Surfaces. *Computer Aided Geometric Design*, 4(5):341–355, 1988.

[Bloomentha 97] Jules Bloomenthal. *Introduction to Implicit Surfaces.* Morgan Kaufmann, ISBN 1-55860-233-X, 1997. Edited by Jules Bloomenthal With Chandrajit Bajaj, Jim Blinn, Marie-Paule Cani-Gascuel, Alyn Rockwood, Brian Wyvill, and Geoff Wyvill.

[Catmull 78] E. Catmull and J. Clark. Recursively Generated B-spline Surfaces on Arbitrary Topological Meshes. *Computer Aided Design*, pages 350–355, November 1978.

[Doo 78] D. Doo. A subdivision algorithm for smoothing down irregular shaped polyhedrons. In *Proc. Conf. Interactive Techniques in CAD, Bologna, Italy*, pages 356–360. IEEE, 1978.

[Hoppe 93] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. *Proceedings of SIGGRAPH 93*, pages 19–26, August 1993. ISBN 0-201-58889-7. Held in Anaheim, California.

[Hoppe 94] Hugues Hoppe, Tony DeRose, Tom Duchamp, Mark Halstead, Hubert Jin, John McDonald, Jean Schweitzer, and Werner Stuetzle. Piecewise smooth surface reconstruction. *Proceedings of SIGGRAPH 94*, pages 295–302, July 1994. ISBN 0-89791-667-0. Held in Orlando, Florida.

[Lindstrom 98] Peter Lindstrom and Greg Turk. Fast and memory efficient polygonal simplification. *IEEE Visualization '98*, pages 279–286, October 1998. ISBN 0-8186-9176-X.

[Lorensen 87] W. Lorensen and H. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics (Proc. SIGGRAPH 87)*, 21(4):163–169, 1987.

[Ning 93] Paul Ning and Jules Bloomenthal. An evaluation of implicit surface tilers. *IEEE Computer Graphics and Applications*, 13(6):33–41, November 1993.

[Nishimura 85] H. Nishimura, A. Hirai, T. Kawai, T. Kawata, I .Shirakawa, and K. Omura. Object Modelling by Distribution Function and a Method of Image Generation. *Journal of papers given at the Electronics Communication Conference '85*, J68-D(4), 1985. In Japanese.

[Nuij 96] Wim Nuij and Kees van Overveld. Behaviour of implicit surfaces near critical points. In *Implicit Surfaces '96*. Eurographics, October 1996.

[Overveld 93] Kees van Overveld and Brian Wyvill. Potentials, Polygons and Penguins. An efficient adaptive algorithm for triangulating an equi-potential surface . In *Proc. 5th Annual Western Computer Graphics Symposium (SKIGRAPH 93)*, pages 31–62, 1993.

[Pasko 95] A. Pasko, V. Adzhiev, A. Sourin, and V. Savchenko. Function representation in geometric modeling: concepts, implementation and applications. *The Visual Computer*, 2(8):429–446, 1995.

[Witkin 94] Andrew Witkin and Paul Heckbert. Using particles to sample and control implicit surfaces. *Computer Graphics (Proc. SIGGRAPH 94)*, 28:269–277, July 1994.

[Wyvill 86] Geoff Wyvill, Craig McPheeters, and Brian Wyvill. Data Structure for Soft Objects. *The Visual Computer*, 2(4):227–234, February 1986.

[Wyvill 96] Brian Wyvill and Kees van Overveld. Polygonization of Implicit Surfaces with Constructive Solid Geometry. *Journal of Shape Modelling*, 2(4):257–274, 1996.

[Wyvill 99] Brian Wyvill, Eric Galin, and Andrew Guy. Extending The CSG Tree. Warping, Blending and Boolean Operations in an Implicit Surface Modeling System. *Computer Graphics Forum*, 18(2):149–158, June 1999.