Curvature Dependent Polygonization of Implicit Surfaces

Bruno Rodrigues de Araújo Joaquim Armando Pires Jorge
Department of Information Systems and Computer Engineering
INESC-ID/IST/Technical University of Lisbon
R. Alves Redol, 1000-29 Lisbon, PORTUGAL
brar@immi.inesc-id.pt,jorgej@acm.pt

Abstract

We present an algorithm for polygonizing closed implicit surfaces, which produces meshes adapted to the local curvature of the surface. Our method is similar to, but NOT based on, Marching Triangles, in that we start from a point on the surface and develop a mesh from that point using a surface-tracking approach. In a marked departure from previous approaches, our meshes approximate the surface through heuristics relying on curvature. Furthermore, our method works completely on-the-fly, resolving cracks as it proceeds, without the need for any post-remeshing step to correct failures. We have tested the algorithm with three different representations of implicit surfaces, Variational, analythical and MPU, using non-trivial data sets, yielding results that illustrate the flexibility and scalability of our technique. Performance comparisons with variants of Marching Cubes show that our approach is capable of good accuracy and meshing quality without sacrificing computing resources.

1. Introduction

Polygonization of implicitly defined surfaces has been the subject of considerable work in the past years. This is because such surfaces are instrumental in representing smooth objects in a number of fields ranging from animation and computer aided design to medical imaging.Constructing a visual representation of the surface involves obtaining the set of points that satisfy the equation $f(x, y, z) = 0$ either by ray-tracing, or by polygonizing the surface. While the most common polygonization approaches are based on volumetric space decomposition, recent work has focused on surface-based methods that can often both provide better meshing approximations to the intended surface at a lower computational cost and do so efficiently, i.e. with a minimum of triangles.

In this document, we describe a fast, progressive algorithm that generates adaptive meshes suitable for use in interactive applications using Hessian and gradient information. After surveying related work, we discuss curvature measures in detail. Then we describe our algorithm. Finally we present results from our work, comparing them to space-subdivision approaches and discuss performance vs. quality tradeoffs.

2. Previous Work

Since their introduction, implicit surfaces provide a solution of choice for imaging medical information or point sets arising from digitizing complex models thanks to their flexibility in manipulating data. However precise and fast visualization of implicit surfaces is difficult. Since the more obvious approaches based on ray-tracing [9, 7, 25, 14] are computationally too expensive for interactive applications even using solutions for speeding intersection calculations such as Kalra [15].fTherefore, subsequent research has focused on approximating surfaces using polygon meshes, more suitable for real-time visualization using commodity hardware. Moreover, polygonal meshes enable us to explore trade-offs between fidelity of representation and interactive performance. In what follows, we will survey the main approaches to surface visualization.

Cell partitioning techniques are the most popular methods for rendering implicit surfaces, by creating a polygonal mesh that fits the surface. The main implementation of this method is the Marching Cubes (MC) algorithm [18] and its variant Marching Tetrahedra (MT) [6] that made it viable to use implicit surfaces for representing model data. MC and its many variants, based on space subdivision provide a simple and effective approach made popular by the availability of source code. However, the resulting meshes are not homogeneous and they exhibit poor quality.

Surface tracking denotes a family of polygoniza-

Proceedings of the XVII Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI'04)
1530-1834/04 $ 20.00 IEEE

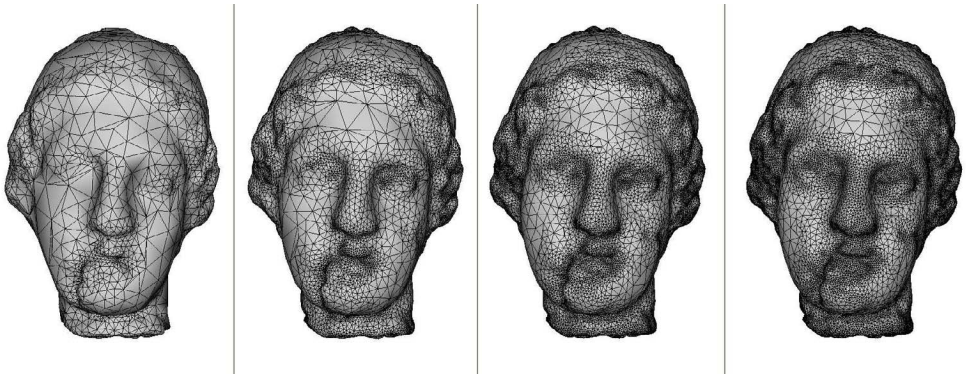

Figure 1. Different triangulations of Igea Head from a variational implicit surface defined by a 2002 point dataset: (from left to right): 4492 triangles, 17806 triangles, 33924 triangles, and 47992 triangles

tion techniques including methods such as Marching Triangles [3] and Hartmann's algorithm [10]. These approaches can achieve good results by generating meshes of evenly-sized and quasi-equilateral triangles. However these are not adapted to local surface properties, requiring large numbers of small triangles to approximate surfaces with large variations in curvature. Hybrid approaches such as shrink-wrap [29] combine surface tracking with cell partitioning to partially overcome this, applying a deformations from a sphere to obtain a suitable approximation to the intended surface.

Adaptive meshing techniques have been developed to achieve better quality approximations. One such technique is mesh simplification [11] which, starting from a high-quality mesh, applies edge split and collapse operations to simplify and optimize it. Mesh quality is assessed in terms of reduced number of triangles, homogeneous triangle sizes, minimizing distance to the surface, curvature and normal direction errors among others. Other algorithms [21, 30] rely on mesh subdivision. Starting from a topologically correct meshing they apply subdivision steps, which obey the same criteria and mesh simplification metrics. These approaches can produce good quality results at the cost of a post-processing step applied on top of a first approximation either based on MC such as [26, 23] or MT.

Other techniques try to achieve similar results using an adaptive triangulation obtained by growing an initial mesh and then filling cracks as a post–processing step. One such method is the surface tracker from [4] which uses a variant of Hilton's Marching Triangle to produce an adaptive mesh. Their main result relies on constraining triangle edge lengths to local surface characteristics using both a mid-point projection heuristic and Delaunay triangulation properties. However, this method does not use curvature estimation. Karkanis [16] developed a similar approach that relies on heuristic curvature estimation using the minimum radius of several geodesics that cover a point. However, this requires expensive computation due to repeated evaluations of the implicit function for any given point and it is not clear that this method yields higher accuracy than ours.

In the following sections, we introduce a new method for polygonizing implicit surfaces based on Hartmann's [10] which focuses on points and front angles to generate an adaptive triangulation in a single step. Our method uses "true" curvature measures based on differential geometry, to avoid expensive estimation methods while providing good quality meshes.

3. Curvature Measures for Implicit Surfaces

Curvature measures are widely used in Computer Graphics both to encode shape characteristics and to improve meshing of surfaces. Since implicit surfaces are represented by mathematical models, they make it possible to derive mathematic notions of curvature using differential geometry. As discussed in [8], methods to obtain these measures are well defined for parametric functions, however their application on implicit mathematical function is not trivial and not so well documented. Hugues [12] discusses differential geometry applied to implicit surfaces, providing interesting clues to retrieve shape characteristics from implicit function mathematics. Our approach uses these notions to extract relevant information from implicit functions as

needed by our polygonization algorithm.

Given a point $X \in R^3$ such that $F(X) = 0$ where F is the implicit function that define our model. The gradient vector $\vec{G}$ and the normal unit vector $\vec{N}$ are defined at the point X using the first order partial derivative of F:

$$\vec{G} = \bigtriangledown F = \left[\begin{array}{ccc} \frac{\partial F}{\partial x} & \frac{\partial F}{\partial y} & \frac{\partial F}{\partial z} \end{array} \right]^T \text{ and } \vec{N} = \frac{\vec{G}}{\|\vec{G}\|}$$

The curvature information of the function F in R^3 at point X is held by the Hessian matrix H, which is defined by second order partial derivatives of F:

$$H = \left[\begin{array}{ccc} \frac{\partial^2 F}{\partial x^2} & \frac{\partial^2 F}{\partial xy} & \frac{\partial^2 F}{\partial xz} \\ \frac{\partial^2 F}{\partial yx} & \frac{\partial^2 F}{\partial y^2} & \frac{\partial^2 F}{\partial yz} \\ \frac{\partial^2 F}{\partial zx} & \frac{\partial^2 F}{\partial zy} & \frac{\partial^2 F}{\partial z^2} \end{array} \right]$$

However, the Hessian matrix H represents the curvature evolution of the scalar field F which supports the implicit function, rather than the surface defined by $F = 0$. Indeed, we need to study curvature values and directions on the plane tangent to $F = 0$ at X. In order to compute these, we need to use the matrix C defined by the partial derivatives of the normal $\vec{N}$ instead of the Hessian:

$$C = \left[\begin{array}{ccc} \frac{\partial N_x}{\partial x} & \frac{\partial N_x}{\partial y} & \frac{\partial N_x}{\partial z} \\ \frac{\partial N_y}{\partial y} & \frac{\partial N_y}{\partial y} & \frac{\partial N_y}{\partial z} \\ \frac{\partial N_z}{\partial x} & \frac{\partial N_z}{\partial y} & \frac{\partial N_z}{\partial z} \end{array} \right]$$

The matrix C can be defined using the Hessian matrix S and the gradient vector $\vec{G}$ as we can see from the following equation:

$$C_{ij} = \frac{H_{ij} * \|\vec{G}\| - \frac{G_i * dot_j}{\|\vec{G}\|}}{\|\vec{G}\|^2}$$

where $dot_j = \vec{G}.\left[\begin{array}{ccc} H_{j0} & H_{j1} & H_{j2} \end{array} \right]^T$

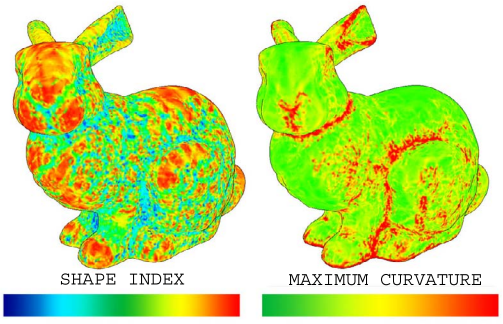

Figure 2. Curvature measures for the Stanford Bunny data set using the MPU implicit model.

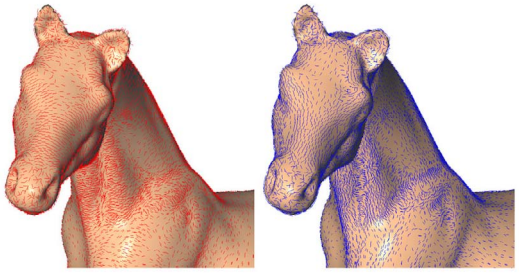

Figure 3. Horse's head showing Principal Directions of Curvature, k_{max} (left) and k_{min} (right)

Since C is defined in terms of the normalized gradient, one of its eigenvalues has zero value and the corresponding eigenvector is normal to the surface (and thus collinear to $\vec{G}$). The other two eigenvalues k_1 and k_2 are called the principal curvatures. Their respective eigenvectors are the principal directions defined to lie in the plane tangent to the surface at X. According to [8] and [17], the following curvature measures can be computed from k_1 and k_2:

- Gaussian Curvature $= k_1 * k_2$
- Mean Curvature $= \frac{k_1 + k_2}{2}$
- Maximum Absolute Curvature $= max(|k_1|, |k_2|)$
- Shape Index $= -\frac{2}{\pi} \arctan \frac{k_{max} + k_{min}}{k_{max} - k_{min}}$

Figure 2 depicts both the shape index (on the left side) and maximum absolute curvature (on the right) for a rendering of the Stanford Bunny represented using the MPU implicit model proposed by [22]. The shape index is a representation of curvature proposed by [17] which depicts both the convexity and concavity features in a scale-independent manner. Visual comparison with Masuda [19] curvature estimation shows that curvature results are very similar, thus illustrating the validity of our measures. Figure 3 shows the principal directions obtained from the matrix C for the Horse data set, also computed from an MPU representation.

Both the gaussian and mean curvature values are widely used by re-meshing algorithms and several works [5, 20] present estimations of these as applied to meshes. However, our approach uses the maximum absolute curvature (MAC) to estimate polygon size for two main reasons. For one, we do not need to distinguish between negative and positive curvature values to compute edge length when creating polygons. Also, this value is guaranteed to be no smaller than any curvature measures on the plane tangent to $F(X) = 0$, regardless of direction. These

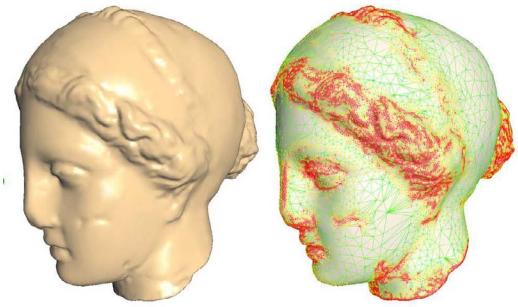

Figure 4. MPU Igea head polygonized using our approach (left) and MAC measure (right)

arguments show that the MAC is a reliable measure of tessellation size at any point of F. We use MAC to define the minimum radius r_{min} of curvature as follows:

$$r_{min} = \frac{1}{max(|k_1|, |k_2|)}$$

We then use r_{min} to compute heuristic edge length for the triangles that constitute the mesh at point X. However, we limit edge length to avoid creating overly small or too large triangles. Figure 4 depicts the resulting meshing of Igea Model [1] using our algorithm and presents MAC values using a color scale. As we can see in high-curvature areas, depicted in red, triangle size tends to be small and triangles become large in low-curvature areas, depicted in light blue.

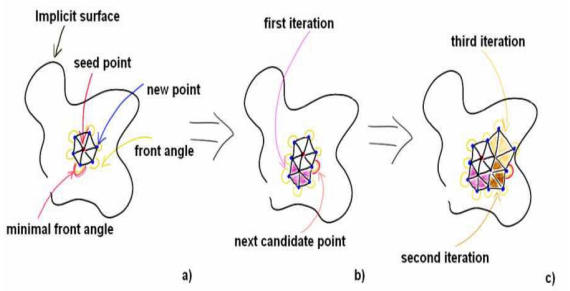

Figure 5. a) Starting triangulation from a seed point, b) 1^{st} iteration, c) 2^{nd} and 3^{rd} iteration

```
POLYGONIZATION( implicit-surface )
 1  pt0 ← GET-SEED-POINT( implicit-surface )
 2  f0 ← CREATE-EMPTY-FRONT()
 3  S ← CREATE-EMPTY-STACK()
 4  mesh ← CREATE-EMPTY-MESH()
 5  INITIAL-TRIANGULATION(pt0, f0, mesh)
 6  PUSH-FRONT(f0, S)
 7  while IS-NOT-EMPTY(S)
 8  do
 9      f0 ← POP-FRONT(S)
10      while FRONT-SIZE(f0) > 3
11      do
12          ACTUALIZE-ANGLES(f0)
13          pt ← GET-POINT-MINIMAL-ANGLE(f0)
14          ptI ← SELF-INTERSECTION(pt, f0)
15          if ptI = NIL /* no intersection */
16            then f1 ← FRONT-COLLISION(pt, f0, S)
17                 if f1 = NIL /* no collision */
18                   then EXPAND(pt, f0, mesh)
19                   else UNIFY-FRONT(f0, f1, S)
20            else SPLIT-FRONT(f0, ptI, S)
21      TRIANGULATE(f0, mesh)
22      DELETE(f0)
23      return mesh
```

Figure 6. Algorithm pseudo-code

4. Our Adaptive Surface Tracking Algorithm

Our algorithm works by point expansion along the surface. We start from a seed point and find its projection on the surface using an iterative procedure known as Newton Step. Then we expand that point generating an hexagon lying on the plane tangent to the surface. Each new vertex is also projected on the surface. The triangles joining those vertices are added to our empty tessellation as shown in Figure 5 a). The vertices of the initial hexagon define the starting front. This is a closed polygon joining unexpanded points from the polygonization boundary. Then the algorithm applies a polygonization cycle as presents the pseudo-code of the Figure 6. At each cycle, a point of the front is selected to expand the triangulated mesh. Before proceeding with the expansion step, meshing overlap is avoided checking two scenarios. The intra-collision in the front which requires division creating new fronts as depicts Figure 8. The inter-collision with other fronts obtain by a possible previous subdivision of a front. Figure 7 shows a situation where this test fails. In this event both fronts must merge at the candidate point location. Along the execution of the polygonization cycle, fronts remain stored in a stack data structure and the current front shrinks down to three points to form a unique triangle. When the triangulation of the actual front is finished, we proceed to expand another front from the stack. The polygonization process ends when there are no more fronts, yielding the resulting polygonal mesh.

5. Expanding Triangles

The expansion process is applied at each candidate point. This step generates new triangles to be added to the mesh approximating the surface and creates new points to expand the actual front. Before applying this process, we verify that new triangles will not overlap others in the mesh. The candidate point is selected as to have minimal front angle θ_{min}. We compute this front angle using the adjacent points in the front and we generate both maximum curvature measure and a local coordinate system definition (normal $\vec{n}$ and two perpendicular vectors $\vec{t_1}$ and $\vec{t_2}$, lying on the plane tangent to the surface at that point).

We compute the edge length for each triangle generated by expansion using the heuristic multiple of the radius of curvature. This multiplying factor controls the precision of our approximation. To achieve better results and more reliable edge length measure, we compute the heuristic value at both neighbor points on the front considering the smaller value as the effective edge length for the expansion.

After computing edge length, we define the number n of triangles needed for the complete expansion of the candidate point and its incident angle θ_i as follows trying to create quasi-equilateral triangles:

$n = floor(\frac{3\theta_{min}}{\pi})$ and $\theta_i = \frac{\theta_{min}}{n}$

To correct extreme case due to truncated function, we apply the adjustments proposed by Hartmann's [10] using our heuristic edge length for distance checking with adjacent points to the candidate point. Finally, we extend the mesh with triangles joining the new and candidate points. The candidate point is then replaced on the front by the new points.

6. Merging and Splitting Fronts

When we expand fronts, a common problem to all surface tracking approaches is to avoid self intersections. We solve it by using heuristic edge length. First, for each point in the front, we identify the set of visible points threatened by expansion. For each visible point relying on the same side of the surface, we check if the distance between points is smaller than the heuristic edge length. To cover the case where the expansion will yield no new points, we perform an additional for overlap when closing a minimal front angle. This test is not discussed in Hartmann's algorithm but it is required due the non constant edge length of our approach. If the collision test fails, the expansion cannot proceed and the actual front needs to be split. Then specific triangle expansions are applied in order to avoid duplicate points between the two new fronts which are stored in a global stack. While our approach is similar to Hartmann's [10], we need to consider in the collision test the neighbors of adjacent points. This is another modification required by our adaptive approach. To speed-up triangulation, when a new front contains only tree points after the splitting step, we triangulate them directly without creating an additional front.

On the other hand, we need to verify whether the expansion of a point will collide with all other fronts in the stack, using the same rules as the intra-collision test. When a collision is detected, we merge the actual front with the colliding one, instead of proceeding with the expansion. This unifies both fronts at the candidate point by connecting it to the nearest point on the other front. After this step, the old front is removed from the stack, replaced by the current one.

7. Discussion

Since we follow a surface tracking approach, our algorithm, as depicted by Figure 9, avoids two problems that often arise in spatial subdivision algorithms such as MC [18] and MT [6]: topological inconsistencies created by too large cells and useless triangles in a pattern that respect the spatial subdivision, rather than local features of the surface. Our approach solves both prob-

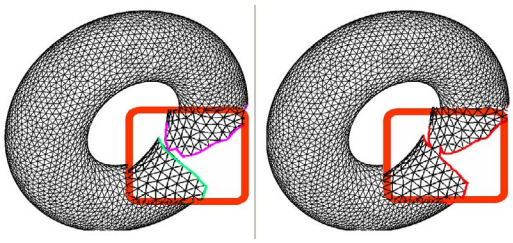

Figure 7. Merging different fronts(left) after collision(right)

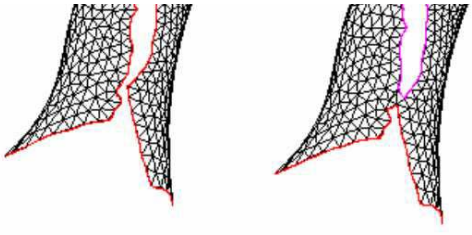

Figure 8. Splitting a single front(left) after collision(right)

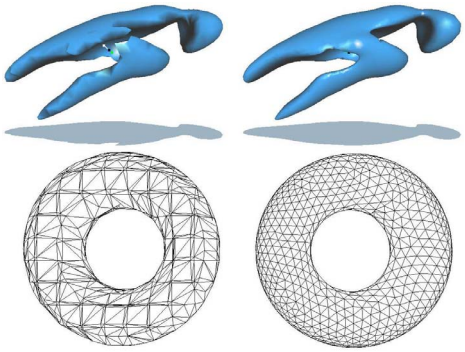

Figure 9. MT Topology preservation (top row) and triangulation pattern (bottom row) problem: MT meshes(left) and our meshes (right).

lems without requiring an expensive post-remeshing step as is done by Treece [26] or Igarashi [13]. Other iterative techniques such as Akkouche's [4] based on Marching Triangles can produce similar results. However our method uses curvature measures extracted from implicit functions, we would like to make a formal comparison of both measures, but it was not possible since the data set used by them are not available.

Unlike Marching Triangles and its variants, which are directed by edges, our algorithm works by expanding points. Both Akkouche's [4] and Karkanis's [16] method are good examples of the former strategy. These approaches are only able to generate one extra point for each edge considered which may require a high number of steps in areas of rapid change. Our point expansion guided by maximum absolute curvature is better suited to such cases, since edge length is recomputed at each point and generate up to three triangles. The main advantage of our approach, compared to both methods, is that we do not require any post step to solving cracks since our front are able to split and merge.

8. Comparison and Results

To present the modularity of our algorithm, we use different models of varying complexity, using Turk's variational implicit surfaces (VIS)[28] and also MPU [22]. We chose VIS for their general and flexible formulation. MPU allows creating implicit functions for large datasets, using a very different approach. We used such model to create Figures 4 and 3. For each implicit model, only the following mathematical functions must be implemented: $Get\_Value(X)$, $Get\_Gradient\_Vector(X)$, $Get\_Normal\_Vector(X)$, $Get\_Hessian\_Matrix(X)$. These functions are straightforward to code and only depend on the surface representation. To speed some calculations up we also implemented composite functions such as $Get\_Value\_and\_Gradient()$ for the Newton Step and $Get\_Gradient\_and\_Hessian()$ thus avoiding extra work.

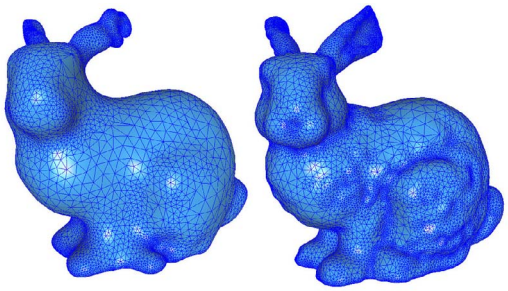

Figure 10. Stanford Bunny VIS reconstruction: left (13422 tri.), right (34440 tri.)

All times shown were measured on a P4 processor at 1800MhZ, with 512Mb of RAM running windows XP and examples are taken from well-known data sets : Stanford Bunny [2] and the Igea head [1]. The corresponding VIS functions were computed using both Turk's and Yngve's methods [27, 31]. Since VIS are a global mathematical representation, the second derivative is algorithmically simple to compute, but the associated computational costs grow quadratically with the number of constraints of the model. For this reason we use two versions of the Stanford Bunny, one based on 800 constraints and the other created from 2000 constraints (Figure 10) and larger data-sets use MPU implicit functions. For comparison, the Igea Head images shown in Figure 1 are based on 4004 constraints.

Computational analysis: Since our algorithm uses a surface tracking approach, it is difficult to measure its complexity. The best criterion for performance comparisons is the time of polygonization. Table I compares our method with MT [6] for the same models using different precisions. The precision factor corresponds to heuristic limits in our approach where a value of 1 means that we use the maximum radius of curvature directly, a value of 2 means to divide it by half and so on. For MT, a value of 0 in this column indicates the minimum distance between two VIS constraint points was used, a value of 1 indicates we multiplied this by 0.7, 2 by 0.6 and so on. We can verify that for the same surface, our approach is faster in terms of triangles per second. We can also see that the values obtained for

Bunny 800	*Our Approach*					*Marching Tetrahedral*				
Heuristic Factor	1	2	3	4	5	0	1	3	4	5
Time (ms)	1963	8492	17225	27860	39196	5888	13519	27309	42861	76310
Triangles	1518	6562	13422	21802	29696	3324	7000	14124	22204	39552
Triangles/s	773	773	779	783	758	565	518	517	518	518
Average Ratio L/s	1.477	1.332	1.291	1.278	1.267	4.655	5.403	5.273	5.792	5.475
Deviation	0.480	0.281	0.211	0.206	0.164	19.075	48.617	39.162	41.380	45.395
Average Depth error	0.533	0.205	0.091	0.059	0.037	0.320	0.168	0.082	0.057	0.033
Phong error	6.286	2.352	1.077	0.688	0.453	4.672	2.579	1.286	0.900	0.499
Phong StDev.	8.607	2.682	1.379	1.106	0.811	10.559	5.463	2.552	2.050	1.263

Table 1. Comparison between our approach and MT for the Stanford Bunny with 800 constraints

different precisions show a similar evolution. We conclude that using real curvature information does not incur a penalty for variational implicit surfaces. Moreover, MT is a good reference since is considered the fastest method for polygonizing implicit surfaces. Our results show that the cost of computing eigenvalues of the Hessian matrix does not introduce significant performance costs.

Mesh quality analysis: Good meshes tend to have twice as many points as triangles. Our approach tries to respect other factors that characterize a good meshing. Meshes resulting from MT show patterns that arise from the space subdivision approach. We can easily see in Figure 9 that MT generates a poor mesh with triangles of very different sizes. Looking at Table I, we can see this by looking at the two rows that list average edge length and ratio of largest to smallest edge lengths of each triangle. Average edge length for MT meshes is almost twice the size of our meshes, while ratios of larger to smaller edges of a triangle show very high values with enormous standard deviations, which are characteristic of the poor meshes produced by MT which contain many ill-formed triangles. In contrast, our approach produces triangles with edge ratios not above 1.5 and with small standard deviations even for coarse approximations. We verified also that approximately 50 percent of generated triangles by our algorithm are equilateral. Table I shows that increasing the heuristic accuracy factor yields more equilateral triangles, which shows that our approach generates better meshes. For each model, we present the standard deviation to ascertain that this holds along the entire surface.

Meshing accuracy analysis: To assess visual fidelity, we compare our mesh with ray-tracing which is the more accurate method to visualize implicit surfaces. We analyze accuracy using two metrics. The first is a depth error.This averages the distance error from a computed ray-surface intersection to the ray-mesh intersection obtained from the same ray and our resulting mesh. The other measure is what we call the Phong error, which is the average angular difference between the normal to the surface and our mesh, taken at each ray intersection. We use the normal of the implicit function at each surface-ray intersection and compare that to the vector obtained by interpolating the normals of the vertices at the intersected triangle as done by Phong shading. Table I shows this measure in degrees. As we can see, we achieve an average Phong error smaller than half degree with a small standard deviation. In comparison to Marching Tetrahedra, our approach achieves better results with smaller standard deviation.

9. Conclusions and Future Work

In this paper we have presented a fast, progressive algorithm that generates adaptive meshes suitable for use in interactive applications. Our algorithm accomplishes this by using Hessian curvature information unlike previous progressive approaches, which resort to less-precise approximation schemes. Our experimental shows that using "real" curvature features provides better results than traditional approaches based on Marching Cubes without a performance penalty or the need for a re-meshing step. The performance of our algorithm depends solely on the implicit surface topology and complexity. However, we are not able to predict the time required for polygonizing a surface. Also, our method currently does not support topologically disjoint features. Using topological studies based on Morse Theory as presented in [24] might solve this problem.

Acknowledgements

This work was supported in part by the Portuguese Science Foundation through grant

POSI/34672/SRI/2000 and by European Commission through grant # IST-2000-28169 (SmartSketches).

References

[1] Sample model cyberware page. http://www.cyberware.com/samples/index.html.

[2] Standford 3d scan repository. http://www-graphics.stanford.edu/data/3Dscanrep/.

[3] J. I. A. Hilton, A. Stoddart and T. Windeatt. Marching triangles: range image fusion for complex object modeling. *International Conference on Image Processing*, 1996.

[4] S. Akkouche and E. Galin. Adaptive implicit surface polygonization using marching triangles. *Computer Graphics Forum*, 20(2):67–80, 2001.

[5] P. Alliez, D. Cohen-Steiner, O. Devillers, B. Levy, and M. Desbrun. Anisotropic polygonal remeshing. *ACM Transactions on Graphics. Special issue for SIGGRAPH conference*, pages 485–493, 2003.

[6] J. Bloomenthal. An implicit surface polygonizer. In P. Heckbert, editor, *Graphics Gems IV*, pages 324–349. Academic Press, Boston, 1994.

[7] J.-D. Gascuel. Implicit patches: An optimised and powerful ray intersection algorithm. In *Implicit Surfaces'95*, pages 143–160, Grenoble, France, Apr. 1995. Proc. of the first international workshop on Implicit Surfaces.

[8] A. Gray. *Modern Differential Geometry of Curves and Surfaces.* CRC Press, Boca Raton, 1997.

[9] J. C. Hart. Ray tracing implicit surfaces. In *SIGGRAPH 93 Modeling, Visualizing, and Animating Implicit Surfaces course notes*, pages 13–1 to 13–15. 1993.

[10] E. Hartmann. A marching method for the triangulation of surfaces. *The Visual Computer*, 14(2):95–108, 1998.

[11] H. Hoppe. Progressive meshes. In *Proc. of the 23rd annual conference on Computer graphics and interactive techniques*, pages 99–108. ACM Press, 1996.

[12] J. F. Hughes. Differential geometry of implicit surfaces in 3-space - a primer. In *Technical Report CS-03-05*. Brown University, 2003.

[13] T. Igarashi and J. F. Hughes. Smooth meshes for sketch-based freeform modeling. In *Proc. of the 2003 symposium on Interactive 3D graphics*, pages 139–142. ACM Press, 2003.

[14] D. Jevans and B. Wyvill. Ray tracing implicit surfaces. In *Technical Report 88/292/04*. University of Calgary, 1988.

[15] D. Kalra and A. H. Barr. Guaranteed ray intersections with implicit surfaces. In *Proc. of the 16th annual conference on Computer graphics and interactive techniques*, pages 297–306. ACM Press, 1989.

[16] T. Karkanis and A. J. Stewart. Curvature-dependent triangulation of implicit surfaces. *IEEE Computer Graphics and Applications*, 22(2):60–69, March 2001.

[17] J. J. Koenderink. *Solid Shape.* Artificial Intelligence Series. MIT Press, Cambridge, MA, USA, 1990.

[18] W. Lorensen and H. Cline. Marching cubes: a high resolution 3d surface construction algorithm. *Computer Graphics*, 21(4):163–169, July 1987. Proc. of SIGGRAPH'87 (Anaheim, California, July 1987).

[19] T. Masuda. Surface curvature estimation from the signed distance field. *Proc. of the 4th International Conference on 3-D Digital Imaging and Modeling*, pages 361–368, 2003.

[20] M. Meyer, M. Desbrun, P. Schröder, and A. H. Barr. Discrete differential-geometry operators for triangulated 2-manifolds. In H.-C. Hege and K. Polthier, editors, *Visualization and Mathematics III*, pages 35–57. Springer-Verlag, Heidelberg, 2003.

[21] P. Neugebauer and K. Klein. Adaptive triangulation of objects reconstructed from multiple range images. *IEEE Visualization '97*, 1997.

[22] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H.-P. Seidel. Multi-level partition of unity implicits. *ACM Trans. Graph.*, 22(3):463–470, 2003.

[23] Y. Ohtake, A. Belyaev, and A. Pasko. Dynamic meshes for accurate polygonization of implicit surfaces with sharp features. In *SMI 2001 International Conference on Shape Modeling and Applications*, pages 74–81. IEEE, 2001.

[24] B. T. Stander and J. C. Hart. Guaranteeing the topology of an implicit surface polygonization for interactive modeling. In *Proc. of the 24th annual conference on Computer graphics and interactive techniques*, pages 279–286. ACM Press/Addison-Wesley Publishing Co., 1997.

[25] N. Stolte and R. Caubet. Fast high definition discrete ray tracing implicit surfaces. In *5th DGCI - Discrete Geome-try for Computer Imagery*, pages 61–70, Clermont-Ferrand, France, 1995.

[26] G. M. Treece, R. W. Prager, and A. H. Gee. Regularised marching tetrahedra: improved iso-surface extraction. *Computers and Graphics*, 23(4):583–598, 1999.

[27] G. Turk, H. Q. Dinh, J. F. O'Brien, and G. Yngve. Implicit surfaces that interpolate. *Proc. of Shape Modelling International*, pages 62–73, 2001.

[28] G. Turk and J. F. O'Brien. Shape transformation using variational implicit functions. In *Proc. of the 26th annual conference on Computer graphics and interactive techniques*, pages 335–342. ACM Press/Addison-Wesley Publishing Co., 1999.

[29] van Overveld and B. Wyvill. Shrinkwrap: An adaptive algorithm for polygonizing an implicit surface. In *Technical Report 93/514/19*. University of Calgary, 1993.

[30] B. Wyvill, P. Jepp, K. van Overveld, and G. Wyvill. Subdivision surfaces for fast approximate implicit polygonization. *University of Calgary, Dept. of Computer Science, Research Report 2000-671-23*, 2000.

[31] G. D. Yngve and G. Turk. Robust creation of implicit surfaces from polygonal meshes. *IEEE Transactions on Visualization and Computer Graphics*, 8(4):346–359, 2002.