Technical Report : Feature Sensitive Surface Extraction from Volumetric Data

Arjun Verma
arjun.verma@iiitb.org
IMT2017008

***Abstract*—The representation of geometric objects as volumetric data has its advantages in techniques such as fast surface interrogation and application of boolean operations (union, intersection, etc.) on the objects. However, there is also a need for a representation where the neighborhood information within the surface is explicitly available. Such information finds its use in shape optimization and freeform modelling techniques. Thus, it is of utmost benefit to find effective conversion algorithms that generate surface representations of an object taking in the volumetric data as input. The existing techniques before this paper failed to take into account the reduction of alias effects generated during the conversion process. This paper addresses this issue by proposing a new representation for the discrete distance fields : *directed distance field* along with a new technique : *the extended Marching Cubes algorithm*.**

***Index Terms*—surface extraction, aliasing effects, directed distance field, extended Marching Cubes algorithm**

I. Introduction

There are two different kinds of surface representations in computer graphics : *parametric surfaces* and *implicit surfaces*. Parametric surfaces are generally given by a function that map two dimensional parameter domain to a three space. On the other hand, implicit surfaces (volumetric representation) are defined as the zero level iso-surface of a 3-dimensional scalar field $f(x, y, z)$. Abstractly, parametric surfaces can be seen as the *range* of a function and implicit surfaces can be seen as the *kernel* of a function. The conversion from implicit surfaces to parametric is done by finding surface samples and connecting them to a polygonal mesh. It is this conversion that the paper focuses on.

This paper proposes an extension to the well known *Marching Cubes (MC)* algorithm [1] to generate an enhanced quality of output mesh during the surface extraction process. The motivation to come up with such an algorithm came from the need to address the aliasing artefacts generally observed during such type of conversions. These aliases occur due the fact that the existing techniques use discrete volume data and sample the implicit surface $f(x, y, z) = 0$ on a uniform spatial grid. These issues have been rectified by using the *Directed* volume data and the *Extended Marching Cubes* algorithm respectively which have been discussed in detail in subsequent sections. The results obtained from the application of proposed techniques have been shown in 1.

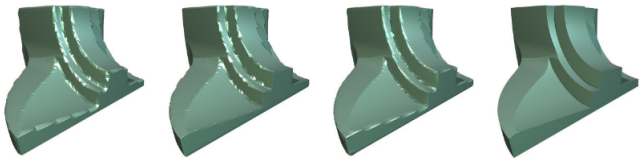

Fig. 1. The first image depicts the use of Standard MC + scalar distance field. The second image depicts the use of Standard MC + directed distance field. The third image depicts the use of Extended MC + scalar distance field. The fourth image depicts the use of Extended MC + directed distance field.

II. Related Work

The process of surface extraction from volumetric data can be divided into two sub-problems. First, one needs to find a dense set of surface samples and then connect them to in a topologically consistent manner to approximate the original mesh *S*. There are two categories of techniques that one can proceed with : *grid based techniques* and *grid-less techniques*.

Grid-less techniques involve beginning with some initial polygonal approximation of the surface *S* and is iteratively improved by attracting the mesh vertices to the surface *S*. The scalar field f is used as a potential field to facilitate the movement of the vertices. By combining this attractive force with a regularizing force, grid-less techniques produces meshes of high quality if *S* is smooth [2], [3], [4]. However, in the presence of sharp features, alias effects come into the picture.

Grid-based techniques on the other hand extract a piece of the surface S for each cubic cell in the grid $g_{i,j,k}$. The cell edges intersect with the surface S to produce surface samples which are then interconnected to produce a triangle mesh. The *Marching Cubes* algorithm [1] is the predecessor for most of the grid-based techniques which involve storing a pre-processed triangulation in a table for all possible configurations of edge intersections.

The proposed technique in this paper is thus an extension on the existing grid-based techniques as they are inherently simple and efficient. The algorithm applies an adaptive refinement strategy which split only those cells that contain a piece of S. This gives us a crust of the finest level cells around the surface. Since each cell is processed separately it can be assumed we have a uniform grid and the adaptive octree traversal enumerates a part of its cells.

III. Discussion

A. Directed Distance Field

The volume representation for a given surface S consists of a scalar valued function f such that,

$$[x, y, z] \in S \iff f(x, y, z) = 0$$

This function f is not uniquely defined for a given surface. One choice for f however is the *signed distance field* function which assigns to every point (x, y, z) $\in R^3$ its distance,

$$f(x, y, z) := dist([x, y, z], S)$$

If a point is enclosed within the surface, the distance value is assigned a negative sign otherwise its assigned positive. Such a function helps to easily compute the frequently used boolean operations in modeling,

$$[x, y, z] \in S_1 \cap S_2 \iff max\{f_1(x, y, z), f_2(x, y, z)\} = 0$$
$$[x, y, z] \in S_1 \cup S_2 \iff min\{f_1(x, y, z), f_2(x, y, z)\} = 0$$
$$[x, y, z] \in S_1 \backslash S_2 \iff max\{f_1(x, y, z), -f_2(x, y, z)\} = 0$$

With such a representation, the standard way to store the distance field f is for a surface S in an efficient data structure is to sample f on a uniform spatial grid $g_{i,j,k} = [ih, jh, kh]$, where h is the size of the cell edge. The sampled distances are thus computed as,

$$d_{i,j,k} = f(ih, jh, kh)$$

and can be interpolated on each grid cell as,

$$C_{i,j,k} = [ih, (i + 1)h] \times [jh, (j + 1)h] \times [kh, (k + 1)h]$$

by a tri-linear function such that we obtain a piecewise approximation $f*$ of f and the surface generated by $f*$, $S*$, approximates S.

A huge limitation of this technique is that the samples on $S*$ are not approximately close to S in the presence of sharp features. This can be seen as an example in 2. Consider the two neighboring green grid points in the presence of a sharp feature on the red contour. Sampling the scalar valued distance field at both grid points (given by blue) and estimating the sample point by linear interpolation does not lead to a good approximation of the intersection of the red contour and green cell edge.

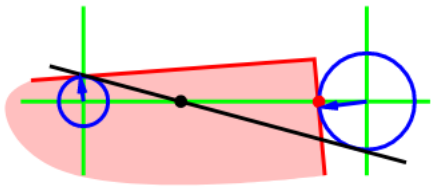

Fig. 2. Sample point (black dot) obtained as an approximation for the intersection (red dot) using the scalar distance field

While one can certainly try to better this approximation by refining the cell edge more by reducing h or by using higher order polynomial interpolants, this however doesn't take care of the root of the problem which is the generation of aliasing artefacts. Such techniques also lead to unnecessary increase in computations and complexity.

To address this issue at the core, the paper [5] proposes the use of *directed distance field*. This means that for each grid point $g_{i,j,k}$, three directed distances in x, y and z directions are computed instead of the scalar valued distances $d_{i,j,k}$, i.e,

$$d_{i,j,k} = \begin{bmatrix} dist_x \\ dist_y \\ dist_z \end{bmatrix}$$

The boolean operations still remain the same and the only fact that changes is that they are now computed component wise. The Marching Cubes algorithm can still be applied to this new representation with the same sign configuration as before. The intersection point between grid $g_{i,j,k}$ and $g_{i+1,j,k}$ is now computed as,

$$s = (1 - |d_{i,j,k[x]}|/h)g_{i,j,k} + (|d_{i,j,k[x]}|/h)g_{i+1,j,k}$$

The gained advantage that we have by using this technique has been demonstrated in 3. By storing the blue directed distances in x and y direction at every green

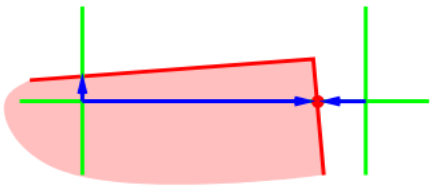

Fig. 3. Sample point obtained exactly as the intersection point (red dot) using the directed distance field

grid point, we can compute the exact intersection points of the contour with the cell's edges.

The directed distance fields can either be computed locally for each cell edge separately or one can combine collinear cell edges to concatenate grid points, $g_{0,j,k}, g_{1,j,k} \ldots, g_{n,j,k}$ into one axis aligned ray $g(\lambda) = g_{0,j,k} + \lambda[1,0,0]$ and then compute all intersections along this ray. The intersection points are then used to store the directed distances corresponding to grid points $g_{i,j,k}$.

B. Extended Marching Cubes

Although we have been able to compute the exact surface samples with the help of directed distance field, the hindrance of removing the aliasing artefacts still remains. We could reduce the approximation error by refining the grid cells however the normals of the extracted surface S^* would still not converge to those of S. We thus wish to locally adapt the sampling grid to the features of the object without using techniques such as higher order polynomial interpolants to preserve the simplicity and the efficiency of the standard Marching Cubes. The paper [5] does this by using additional information from the distance field f to extrapolate the behaviour of the surface near the feature. Instead of directly connecting the intersection points of the contour with the cell edges, the paper additionally uses the contour normals to compute a linear local approximation (tangent elements) for each intersection point. The intersecting tangents thus yield an additional sample point which better approximates the sharp feature. This has been shown clearly in 4

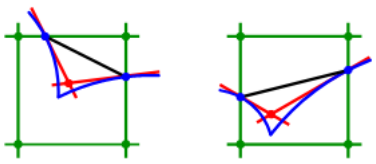

Fig. 4. The intersection of the tangential elements serves as an approximation for the sharp feature in the cell

In the extended Marching Cubes algorithm, different types of sharp features have to be handled differently. These types primarily include *feature edges* and *corners* and the cells comprising of them are classified accordingly. 5 depicts this feature detection. This classification is similar to that of the cells in the extended octree structure where the leaves of an octree for a CSG model are tagged as face cells, edge cells and vertex cells. If the cell does not contain a sharp feature, we generate a local triangular mesh by using the standard Marching Cubes algorithm. However, in the presence of a sharp feature, the tangents at sample points are utilized as per the extended Marching Cubes to give one additional sample point. Instead of using the standard triangulation, a triangle fan is generated with the new additional point at its centre. Thus, the advantages of regular data structures has now been combined with the flexibility of adaptive sampling.

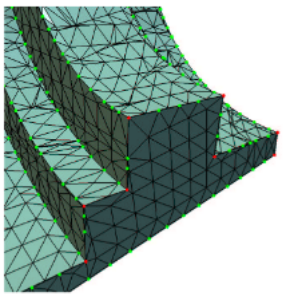

Fig. 5. Red dots are classified as corners whereas the green dots are classified as sharp edges

Since the triangle fans for the sharp features have been added without any consideration for the neighboring cells, a post processing step is required for the computed triangle mesh to accurately represent the presence of features. In the post processing step, as few of the mesh edges are *flipped* with the condition being that an edge is flipped only if connects two feature samples after the flip. This process has been shown in 6.

With the algorithm defined, the only thing now left is to show how the features are classified and how the feature sample point is computed.

1) Surface Normals: Both, feature detection and sampling requires the computation of additional information in the form of surface normals. For the surface normals, one needs to exactly evaluate the gradient of the distance field. These gradients can be computed in advance during the discretization of the distance field.

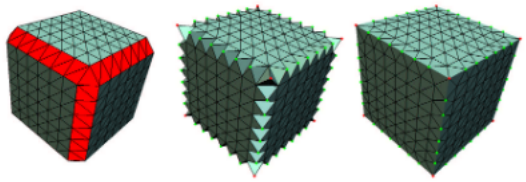

Fig. 6. The first image depicts the classification of cells based on the sharp features detected in them. The second image shows the triangle fan obtained after the extended MC is applied for such cells. The last image shows the result after edge flipping as per the condition discussed in the section

For *implicit surfaces*, the gradients are computed with respect to all three coordinates symbolically if the analytic function f is known. For *polygonal meshes*, the normalized vector pointing from the query point to the closest point on the surface is the normalized gradient of the distance field. For the extended Marching Cubes, since we only need points lying exactly on the surface *S*, we can simply use the normal vector of that triangle on which the sample point lies. For *point clouds*, we utilize the same technique as above and use the normal vector of the nearest scattered point. In case of the information available as *scalar distance fields*, we either compute the gradient of the tri-linear interpolant or we estimate the gradient in each grid point by divided differences using the neighboring grid points.

2) Feature Detection: We represent the surface samples obtained by the intersection of the octree cell $C_{i,j,k}(h)$ with the surface *S* as $s_0, s_1, ..., s_n$. If there are multiple components in the cell, we assume all s_i to lie in the same component. The feature detection and sampling can be applied individually to each component. Let n_i be the unit surface normal of S at s_i. Our aim is now to find out if the surface patch *S* represented by s_i contain a sharp feature. This is done by computing the opening angle of the normal cone spanned by n_i. If,

$$\theta := min_{i,j}(n_i^T n_j)$$

is smaller than some threshold θ_{sharp} then we expect the surface to have a sharp feature. Let n_0 and n_1 be the normals that enclose the largest angle and the normal vector to the plane spanned by them is computed as $n^* = n_0 \times n_1$. To bifurcate if the sharp feature is an edge or a corner, we estimate the maximum deviation of the normals n_i from the plane spanned by n_0 and n_1. We thus calculate,

$$\phi := max_i |n_i^T n^*|$$

and check if it is greater than some threshold ϕ_{corner}. Both the thresholds are generally taken to be large enough with θ_{sharp} being around 0.9 and ϕ_{corner} around 0.7. These are necessary to distinguish between sharp corners and curved feature lines.

The parameters are highly intuitive to understand and one can picture this by imagining a handkerchief on the ground. If it is lifted from the ground at an arbitrary point, the point will have a large θ_{sharp} and ϕ_{corner} and would thus represent a corner. Similarly one can imagine the visualizations for an edge.

3) Feature Sampling: After classifying a cell as an edge or a corner, we try to find a sample point as close as possible to the feature. The new sample point *p* is obatined by solving the linear system,

$$[..., n_i, ...]^T p = [..., n_i^T s_i, ...]$$

We solve this system with the pseudo-inverse based on the singular value decomposition of $N = [..., n_i, ...]^T$. The pseudo inverse will leads to the point *p* on the feature line which is closest to the origin. In order to guarantee that this point lies within a reasonable configuration of s_i, we apply a coordinate transform to the samples before setting up the system such that their centre of gravity lies on the origin.

IV. Applications

We now briefly discuss the applications where the proposed technique finds it use,

1) CSG Modeling: The feature sensitive sampling is very important for this application as the sharp edges and corners over here indicate intersection of basic objects and thus contain significant design information. It also finds use in the simulation of the milling process.

2) Surface Reconstruction: The feature detection part of the algorithm plays an important role in the reconstruction of a surface from scattered point clouds. Many optimization techniques have been introduced to improve the smoothening of polygons based on local filtering operations [6], [7], [8]. Based on whether a mesh vertex is a sharp edge or a corner, the information can be exploited to improve surface quality in the smoothening step by applying a univariate smoothening scheme to the feature lines and a bivariate smoothening scheme to the non-feature areas.

3) Remeshing: Similar to the feature sensitive smoothening, we can utilize feature information to control the output of the mesh decimation post-process. A mesh decimating scheme based on edge collapsing [9], [10], [11] is tested in the paper. Unless the collapsed

edge is a feature edge, feature vertices are not allowed to change their position during an edge collapse . Doing this, an effectively decimated mesh that preserves most of the relevant feature information is obtained.

V. Conclusion

Thus, the extended Marching Cubes algorithm is able to reliably detect and classify sharp feature regions and accurately samples these features to reduce the alias artefacts. The algorithm still needs to fix some gaps to allow for it to correctly process cells with different refinement levels that meet each other. This would then enable the extended Marching Cubes to be generalized to balanced octrees [12], [13] as well. The algorithm in the paper has not been optimized for computation speed with more attention being paid to the parallelizing of the algorithm.

References

[1] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," in *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, (New York, NY, USA), p. 163–169, Association for Computing Machinery, 1987.

[2] Z. J. Wood, P. Schröder, D. Breen, and M. Desbrun, "Semi-regular mesh extraction from volumes," in *Proceedings of the Conference on Visualization '00*, VIS '00, (Washington, DC, USA), p. 275–282, IEEE Computer Society Press, 2000.

[3] D. Terzopoulos, "Regularization of inverse visual problems involving discontinuities," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 4, pp. 413–424, 1986.

[4] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *INTERNATIONAL JOURNAL OF COMPUTER VISION*, vol. 1, no. 4, pp. 321–331, 1988.

[5] L. P. Kobbelt, M. Botsch, U. Schwanecke, and H.-P. Seidel, "Feature sensitive surface extraction from volume data," in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, (New York, NY, USA), p. 57–66, Association for Computing Machinery, 2001.

[6] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr, "Implicit fairing of irregular meshes using diffusion and curvature flow," in *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, (USA), p. 317–324, ACM Press/Addison-Wesley Publishing Co., 1999.

[7] L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel, "Interactive multi-resolution modeling on arbitrary meshes," in *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, (New York, NY, USA), p. 105–114, Association for Computing Machinery, 1998.

[8] G. Taubin, "A signal processing approach to fair surface design," in *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, (New York, NY, USA), p. 351–358, Association for Computing Machinery, 1995.

[9] M. Garland and P. Heckbert, "Surface simplification using quadric error metrics," in *SIGGRAPH 97 Conference Proceedings* (T. Whitted, ed.), pp. 209–216, Aug. 1997.

[10] H. Hoppe, "Progressive meshes," in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, (New York, NY, USA), p. 99–108, Association for Computing Machinery, 1996.

[11] L. Kobbelt, S. Campagna, and H. Seidel, "A general framework for mesh decimation," in *Graphics Interface*, 1998.

[12] Y. Livnat, H.-W. Shen, and C. R. Johnson, "A near optimal isosurface extraction algorithm using the span space," *IEEE Transactions on Visualization and Computer Graphics*, vol. 2, p. 73–84, Mar. 1996.

[13] R. Shekhar, E. Fayyad, R. Yagel, and J. F. Cornhill, "Octree-based decimation of marching cubes surfaces," in *Proceedings of the 7th Conference on Visualization '96*, VIS '96, (Washington, DC, USA), p. 335–ff., IEEE Computer Society Press, 1996.