Direct Extraction of Normal Mapped Meshes from Volume Data

Mark Barry and Zoë Wood

Abstract. We describe a method of *directly* extracting a simplified contour surface along with detailed normal maps from volume data in one fast and integrated process. A robust dual contouring algorithm is used for efficiently extracting a high-quality "crack-free" simplified surface from volume data. As each polygon is generated, the normal map is simultaneously generated. An underlying octree data structure reduces the search space required for high to low resolution surface normal mapping. The process quickly yields simplified meshes fitted with normal maps that accurately resemble their complex equivalents.

1 Introduction

Volume data is used in scientific and medical imaging as well as in the process of scanning physical objects [1]. This data structure has the benefit that it can describe surfaces, spatial extents and interior structures. The most popular method for viewing volume data requires extracting a *contour* surface or isosurface. Typically an extracted contour surface of high detail will contain a very large number of polygons. Approximating the original surface by reducing the extracted mesh's resolution is desirable.

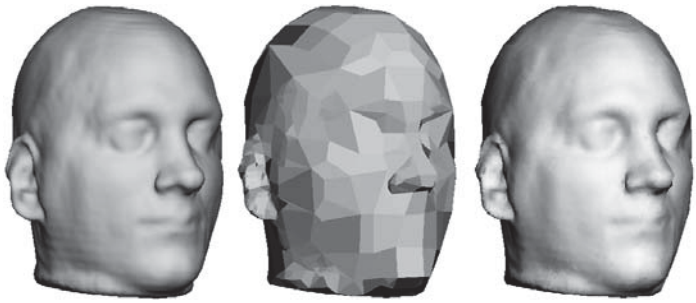

Fig. 1. A human head extracted from a 130 x 128 x 128 volume. All meshes are dual contour surfaces. Left: 56,637 quads. Center: 1,406 quads. Right: same as center but with normal maps, generated with our algorithm in about 1 second.

A technique known as *normal mapping* is one way to achieve the appearance of fine detail on a low resolution version of the surface. Current normal mapping algorithms are applicable to volume data only after a surface has been extracted, requiring a lengthy, multi-step process. We demonstrate a novel method of greatly shortcutting the current process by directly extracting a simplified surface along with normal maps in one integrated process, using the volume data structure to our advantage. The process is very fast - capable of extracting normal mapped meshes of various resolutions in the time of one second on average, giving the user the ability to quickly choose a mesh at the desired level of detail. A low resolution mesh extracted using our algorithm can render almost fourteen times faster then its high resolution equivalents. Yet even normal mapped low resolution meshes, with 92% fewer polygons, appear nearly identical to their high resolution equivalents (see Figure 5).

2 Previous Work

2.1 Isosurface Extraction

The *Marching Cubes* (MC) algorithm [2] can generate a high-quality mesh that captures a contour's fine details, but commonly generates very large meshes, ranging in the millions of polygons. The *Extended Marching Cubes* algorithm [3] presents an improvement over the MC algorithm by generating additional contour vertices *within* cubes, which results in higher-quality meshes that capture features such as sharp edges. The downside of these methods is the generation of a high resolution mesh in order to display the contour surface's finest details.

2.2 Adaptive Contouring

Adaptive polygonization or adaptive contouring is one method of dealing with high resolution meshes produced by MC. An *octree* structure [4] [5] [6] can be generated to adaptively represent the volume data. The challenge of adaptive contouring on simplified octrees is that the resulting polygonal mesh is not "water-tight". The problem of generating a closed polygonal mesh from a simplified octree has been extensively studied [7] [8] [5]. In general the solutions proposed require restrictions on how the octree is simplified.

Ju et. al. [9] present an alternative to previous adaptive contouring methods using dual contouring. Dual contouring methods generate a contour vertex *within* a cube's interior. The algorithm presented by Ju et. al. is an *adaptive* dual contouring method that produces a "crack-free" contour surface from a simplified octree. Unlike other adaptive contouring methods, this dual contouring method does not impose restrictions on how an octree is simplified nor requires any sort of crack patching. In addition, it performs as well as [3] in terms of preserving distinct features such as sharp edges. In order to adaptively simplify the octree data structure that represents the volume data, the authors use the quadric error functions (QEFs) introduced in [10]. Fixes and improvements to [9] are discussed

in [11], [12], and [13]. Due to its ability to directly extract adaptive high quality meshes from volume data, we use this algorithm to create our low resolution meshes.

2.3 Simplification and Normal Maps

A popular approach to dealing with high resolution output meshes from MC involves first extracting a high resolution surface and later simplifying it [14] [10] [15]. The low resolution simplified meshes are often a gross approximation of the original mesh's high resolution appearance. One common approach to achieve the appearance of a high resolution mesh using simplified geometry is through the use of a normal map [16] [17]. One of the most popular methods to generate normal maps is presented in [18]. The process first assumes that a high resolution mesh has been simplified into a low resolution approximation. For each polygon in the low resolution mesh, a texture map is created and sampled. For each sample, a ray is cast to determine the nearest corresponding point on the high resolution mesh. This is a costly operation that can involve searching every polygon in the high resolution mesh for a possible intersection with the ray. In [18], the search is optimized by partitioning the search space into cells. Note that creating this spatial partitioning data structure essentially requires the re-creation of a volume data structure if the mesh came from a volume.

In contrast, our method avoids the initial step of extracting of a high resolution mesh. By using adaptive contouring, the algorithm is designed to directly extract the final simplified mesh. Another advantage to using our algorithm is a limited search space for mapping fine to coarse features, which is naturally implemented using the existing octree data structure. Collecting normals to create a normal map is limited to "searching" only four octree cubes that a polygon spans (see Section 3.2).

The authors of [19] present a method of constructing a progressive mesh such that all meshes in the progressive mesh sequence share a common texture parameterization. Thus a normal mapped progressive mesh can easily be created. Our work generates a new set of normal maps for each mesh extracted, while the progressive mesh approach uses a single normal map for all mesh resolutions. The single parameterization approach may be more efficient in the end but takes a substantially longer time to construct for all mesh resolutions.

3 Algorithm Overview and Contributions

We present a method to extract geometrically simplified, low polygon meshes with their accompanying normal maps from volume data. The algorithm follows these simple stages:

1. Create an adaptive octree from the original volume using QEF.
2. Extract the dual contour using [9].
3. For each low resolution polygon, create the associated normal map using the octree data structure.

Steps one and two follow the work of [9] and thus are not presented here in any detail. Step three is our main contribution as we know of no other method to directly extract normal mapped meshes from volume data. Not only is our method novel in its application to the volume domain, but our method of generating normal maps by gathering the fine data in a fine-to-coarse approach (versus a coarse-to-fine approach) is likewise new (see Section 3.3). We discuss the details of the algorithm in the following sections. For complete details on the algorithm, please see [20].

3.1 Dual Contouring of Volume Data With Normal Map Extraction

Our goal is to extract a geometrically simplified surface with a normal map directly from volume data. In addition to being a robust adaptive contouring algorithm, dual contouring is ideally structured for easy extraction of normal maps as well. Thus we use the dual contouring method presented in [9] for both the creation of the octree using QEF and the surface extraction. Dual contouring builds the final mesh by connecting four minimizing vertices found within four neighboring cubes in the octree. If the adaptive structure of the octree includes neighboring cubes of different levels in the hierarchy, triangles are generated. The implementation treats triangles as quads having two vertices fused together. We therefore refer to all surface polygons as *quads*. To make up for the coarse appearance of the mesh alone, normal maps are created and applied to quads as they are generated.

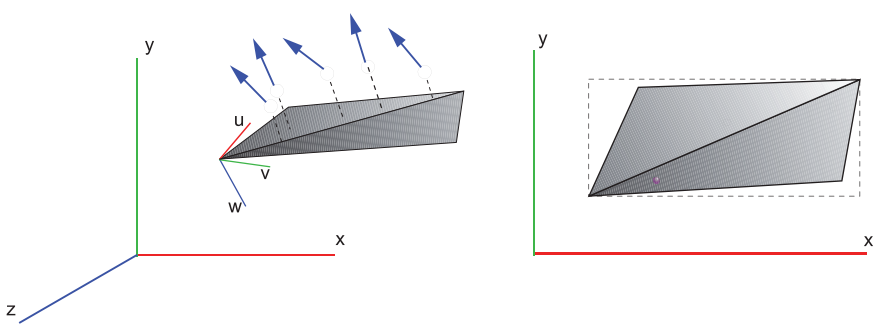

Fig. 2. Projection of fine-level contour vertices and their associated normals onto a contour quad. The dotted bounding box on the right represents the normal map. Note that there are typically *many* more fine-level samples then shown here. See Figure 3.

3.2 Generating Normal Maps

A normal map can be generated and stored in many different polygonal formats; triangles, rectangles, packed charts, etc. We choose to generate a normal map per quad. Normal map generation starts with the creation of a rectangular map.

The dimensions of the rectangle are calculated based on the size of the quad. Because the quad is defined in three-dimensional space and the normal map is only two-dimensional, the quad must be projected onto a two dimensional space.

The quad's orthonormal basis is computed and used to calculate the projection (see Figure 2). The orthonormal basis matrix is composed of three mutually perpendicular row vectors: $\boldsymbol{u}$, $\boldsymbol{v}$, and $\boldsymbol{w}$. We use the cross product of the quad's diagonals to compute $\boldsymbol{w}$, leaving the vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ to be computed. There are no constraints on the orientation of $\boldsymbol{u}$ nor $\boldsymbol{v}$, as long as all three vectors are mutually perpendicular. Choosing values for $\boldsymbol{u}$ and $\boldsymbol{v}$ translates into how the quad will be oriented on the normal map and determines the normal map's dimensions required to bound the quad. For the most efficient use of space, the $\boldsymbol{u}$ vector can be oriented parallel with the quad's longest edge. The $\boldsymbol{v}$ vector can then be computed from $\boldsymbol{w} \times \boldsymbol{u}$. Note that not all quads are planar, however, we have found that using the cross product of the quad's diagonals is a reasonable mapping for our algorithm.

The x-y-z coordinate of each of the quad's vertices is multiplied with the basis matrix to perform the projection. After the projection, the z-component of the quad's vertices can be dropped and the coordinates treated as two-dimensional. The two-dimensional quad is bounded with a rectangle representing the normal map. Next the normal map is filled with the appropriate normal vector values. Sampling is chosen to match the sampling rate of the volume data, thus approximately one texel is created on the normal map per finest level voxel spanning the quad.

The normal maps act to fill in the fine detail that the simplified contour surface lacks. The finest detail that can be obtained comes from the contour vertices generated at the leaves of the octree. Note that each finest level voxel has a gradient computed using a finite difference, and finest level vertices and normals are computed as in [9]. The process of generating a normal map involves capturing the normals from these fine-level contour vertices. Recall that contour vertices are contained within an octree cube and surface polygons are formed by connecting four minimizing vertices, thus each polygon spans four octree cubes. Normals for a low resolution quad are obtained by collecting all the fine-level contour vertices contained within four spanning cubes in the octree. Each cube calls a recursive function for each of its child cubes, traversing down the octree of sub-cubes until reaching a leaf cube. At a leaf cube, contour vertices, if any, are extracted and returned in an array. As all calls to the recursive function return, the arrays of fine-level contour vertices are combined. These fine-level contour vertices are then projected onto the quad's normal map.

3.3 Normal Map Sampling

The projection step samples normals across the normal map but may leave remaining texels' normal values undefined. One method to completely fill the normal map would be to interpolate the normal values at the projected points across undefined texels. Many methods of interpolating scattered data are available [21] but we found that using a very simple sampling method was sufficient for all

Fig. 3. Close-up view of one normal map of the dragon, shown at a coarse level (16K) and the finest level (225K). The finest level view shows the wireframe projected onto the normal map extracted for the coarse mesh. The normal map shown is approximately 30 by 30 texels, closely matching the resolution of the finest level mesh.

example meshes shown. This approximation works by first initializing the entire normal map with the contour quad's normal. After initialization, the fine-level vertices are projected onto the normal map as before, overwriting any initialized texels. Such a simple mapping works well due to the QEF used to generate the low resolution surface, where a low resolution polygon closely matches the high resolution surface. Any areas where the surface may fold back on itself should remain at a high resolution during simplification and thus not cause problems with normal map creation. As shown in figure 3, the normal map produced by this sampling method smoothly matches the original data. Other interpolation methods or even the pull-push method of [22] could be explored in future work, however, our simplified approximation works well due to choosing a sampling rate which matches the resolution of the volume and the qualities of the QEF surface simplification.

Note that most previous work [18] [19] for generating normal maps use a ray shooting technique, where texel samples on the coarse polygon are sampled by searching for an associated point on the high resolution mesh. In this way, these approaches generate a densely sampled normal map for the coarse mesh, where many of the sampled normals will be an interpolated normal from an interior point on the face of the fine mesh. We take the opposite approach by projecting the high resolution sample points to the coarse mesh. This approach is arguably faster, not only because of the octree data structure but because of the number of samples being mapped from fine to coarse (versus the many texel samples generated across the coarse face needing an associated mapping to the fine mesh). Our approach could be considered as less accurate, however, the only sample points that are lost are the interpolated normals from the interiors of the high resolution faces. We found that our fine-to-coarse approach performed not only very quickly but also with visually pleasing results.

The following outlines the main steps in generating normal maps:

1. Use the dual contouring algorithm to extract quads.
2. For each quad created, generate its normal map.
   (a) Calculate the quad's orthonormal basis.
   (b) Transform/project the quad's vertices into two-dimensional space by using the quad's orthonormal basis.
   (c) Collect all fine-level contour vertices (position and normal) for the quad.
      i. For each of the four cubes which the quad spans, recursively traverse the octree to the leaves and append vertices to a common array.
   (d) Project all fine-level contour vertices onto the quad using the quad's orthonormal basis, (associated normals are unchanged).
   (e) Bound the projected quad with a rectangular normal map.
      i. Find the max and min x and y values of vertices in the array.
      ii. Allocate normal map texels of size x extent by y extent. Initialize all texels to quad's normal.
      iii. For each projected contour vertex: Copy its normal into the normal map using integer-rounded x and y of its position.

4 Results

One important measure of success is how well the low resolution normal mapped surface resembles the original high resolution surface without normal maps. Figures 1, 4 - 8 compare a low resolution and normal mapped mesh to its high resolution equivalent, demonstrating the excellent results from using our algorithm. Another factor to consider is the speed at which a mesh, along with its normal maps, can be extracted from the volume. The longest extraction time for our algorithm was for the dragon mesh from a 356 x 161 x 251 volume, along with normal maps, at the highest resolution. This operation takes a little over 3 seconds. Extracting simpler meshes (which would be the more frequent request) takes less time - an average of 1 second for the meshes shown in the following figures. This speed makes it easy for a user to quickly switch between varying levels of mesh resolution. A user may wish to begin with a low resolution mesh for fast display and cursory examination. Then as the user desires more geometric accuracy, a higher resolution mesh may be quickly extracted. It is very important to emphasize that a low resolution mesh is *directly extracted*.

A fast rendering time is one of the main motivations for using normal mapped surfaces. Table 1 compares the rendering performance of high resolution meshes to low resolution, normal mapped, meshes. It is clear that rendering low resolution normal mapped meshes is a lot faster, yet they retain a significant amount of detail as shown in Figures 1, 4 - 8. For the dragon example, even on the lowest resolution meshes, the scales are still clearly defined. The flat-shaded meshes alone do not come near to displaying that kind of detail. At the lowest resolutions, maintaining correct topology of the original high resolution mesh begins to fail. In the extreme example of Figure 6 (Right), the flat-shaded mesh alone grossly resembles the dragon model; applying normal maps restores most of the

Fig. 4. All dragon models extracted from a 356 x 161 x 251 volume. All meshes are dual contour surfaces. Left: high resolution – 225,467 quads. Center: low resolution – 43,850 quads. Right: same as center but with normal maps applied.

Fig. 5. Left: 225,467 quads. Center: 16,388 quads. Right: same as center but with normal maps applied.

Fig. 6. Left two: flat-shaded and normal mapped dual contour surface (558 quads). Right two: flat-shaded and normal mapped dual contour surface (65 quads).

finer details. Figures of the mouse embryo and human head demonstrate that the process works equally well for medical imaging applications.

4.1 Limitations

Though we describe a method that is fast and yields excellent visual results, there is always room for improvement. One inefficiency in the system as it stands is that each polygon allocates its own normal map. Ideally the collection of normal maps could be packed into one or a few *atlases* [18] [19]. Though the result would be a more efficient use of memory space, the packing process is computationally intensive. The memory occupancy for the normal data stored in the normal maps created by the current algorithm for the dragon models shown in figure 5 and 6 range from 7 megabytes for the 225K mesh to 2 megabytes for the 65 face mesh.

Fig. 7. A mythical creature extracted from a 316 x 148 x 332 volume. All meshes are dual contour surfaces. Left: 150,823 quads. Center: 10,950 quads. Right: same as center but with normal maps.

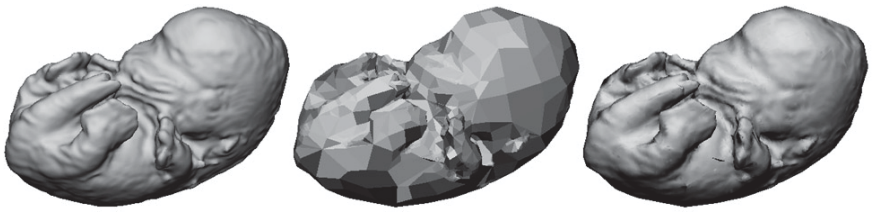

Fig. 8. A mouse embryo extracted from a 256 x 128 x 128 volume. All meshes are dual contour surfaces. Left: 64,896 quads. Center: 3,035 quads. Right: same as center but with normal maps.

Table 1. Performance data for the examples shown in Figures 1, 4 - 8

	Quad Count			Render Time (ms)		
	Hi-Res	Lo-Res	Reduction	Hi-Res	Lo-Res	Speedup
Figure 1	56,637	1,406	97.5%	91	3	30.3
Figure 4	225,467	43,850	80.6%	360	90	4.0
Figure 5	225,467	16,388	92.7%	360	26	13.8
Figure 6 (L)	225,467	558	99.8%	360	1	360
Figure 6 (R)	225,467	65	99.97%	360	0.3	1200
Figure 7	150,823	10,950	92.7%	245	22	11.1
Figure 8	64,896	3,035	95.3%	103	6	17.2

As mentioned in Section 3.2 we use the cross product of the quad's diagonal to create our planar mapping. Our results on numerous input meshes show that such a mapping is sufficient for our application, however, other parameterizations could be explored. In addition, our normal map sampling method is very simple, more complex methods for interpolating normal data could be explored.

5 Conclusion and Future Work

We describe a method of directly extracting normal mapped contour surfaces from volume data. This method greatly shortcuts the current multi-step process of extracting a high resolution mesh, simplifying it, and generating normal maps. The process of generating normal maps is greatly streamlined due to the use of an octree data structure and the dual contouring algorithm. The visual results of this process are of excellent quality - the low resolution normal mapped meshes closely resemble their high resolution equivalents. This process is also very fast - generating and displaying a normal mapped surface in less than a few seconds.

Future work includes extending our algorithm for use in computer games. Destructible game objects could be implemented similar to the work presented in [9]. Additionally, though we use dual contouring's QEF metric for mesh simplification, the use of other error metrics could be explored. The QEF metric aims to preserve the most prominent features such as sharp edges. Perhaps preserving certain sharp edges would not be necessary if they were instead captured in a normal map. Finally, blending normals across large quad boundaries could be improved as small inconsistencies are occasionally seen.

References

1. Curless, B., Levoy, M.: A volumetric method for building complex models from range images. In: SIGGRAPH 1996. Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, pp. 303–312. ACM Press, New York (1996)
2. Lorensen, W.E., Cline, H.E.: Marching cubes: A high resolution 3d surface construction algorithm. In: SIGGRAPH 1987. Proceedings of the 14th annual conference on Computer graphics and interactive techniques, pp. 163–169. ACM Press, New York (1987)
3. Kobbelt, L.P., Botsch, M., Schwanecke, U., Seidel, H.P.: Feature sensitive surface extraction from volume data. In: SIGGRAPH 2001. Proceedings of the 28th annual conference on Computer graphics and interactive techniques, pp. 57–66. ACM Press, New York (2001)
4. Meagher, D.: Geometric modeling using octree encoding. In: Computer Graphics and Image Processing, pp. 129–147 (1982)
5. Wilhelms, J., Gelder, A.V.: Octrees for faster isosurface generation. ACM Trans. Graph. 11, 201–227 (1992)
6. Shekhar, R., Fayyad, E., Yagel, R., Cornhill, J.F.: Octree-based decimation of marching cubes surfaces. In: VIS 1996. Proceedings of the 7th conference on Visualization 1996, p. 335. IEEE Computer Society Press, Los Alamitos, CA, USA (1996)
7. Cignoni, P., Ganovelli, F., Montani, C., Scopigno, R.: Reconstruction of topologically correct and adaptive trilinear isosurfaces. Computers and Graphics 24, 399–418 (2000)

8. Frisken, S.F., Perry, R.N., Rockwood, A.P., Jones, T.R.: Adaptively sampled distance fields: a general representation of shape for computer graphics. In: SIGGRAPH 2000. Proceedings of the 27th annual conference on Computer graphics and interactive techniques, pp. 249–254. ACM Press/Addison-Wesley Publishing Co., New York (2000)
9. Ju, T., Losasso, F., Schaefer, S., Warren, J.: Dual contouring of hermite data. In: SIGGRAPH 2002. Proceedings of the 29th annual conference on Computer graphics and interactive techniques, pp. 339–346. ACM Press, New York (2002)
10. Garland, M., Heckbert, P.S.: Surface simplification using quadric error metrics. In: SIGGRAPH 1997. Proceedings of the 24th annual conference on Computer graphics and interactive techniques, pp. 209–216. ACM Press, New York (1997)
11. Zhang, N., Hong, W., Kaufman, A.: Dual contouring with topology-preserving simplification using enhanced cell representation. In: VIS 2004. Proceedings of the conference on Visualization 2004, pp. 505–512. IEEE Computer Society Press, Los Alamitos (2004)
12. Schaefer, S., Ju, T., Warren, J.: Manifold dual contouring. IEEE Transactions on Visualization and Computer Graphics (2007)
13. Schaefer, S., Ju, T., Warren, J.: Intersection-free contouring on an octree grid. In: Proceedings of Pacific Graphics, IEEE Computer Society Press, Los Alamitos (2006)
14. Heckbert, P.S., Garland, M.: Survey of polygonal surface simplification algorithms. Technical report (1997)
15. Hoppe, H.: Progressive meshes. In: SIGGRAPH 1996. Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, pp. 99–108. ACM Press, New York (1996)
16. Krishnamurthy, V., Levoy, M.: Fitting smooth surfaces to dense polygon meshes. In: SIGGRAPH 1996. Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, pp. 313–324. ACM Press, New York (1996)
17. Cohen, J., Olano, M., Manocha, D.: Appearance-preserving simplification. In: SIGGRAPH 1998. Proceedings of the 25th annual conference on Computer graphics and interactive techniques, pp. 115–122. ACM Press, New York (1998)
18. Cignoni, P., Montani, C., Scopigno, R., Rocchini, C.: A general method for preserving attribute values on simplified meshes. In: VIS 1998. Proceedings of the conference on Visualization 1998, pp. 59–66. IEEE Computer Society Press, Los Alamitos, CA, USA (1998)
19. Sander, P.V., Snyder, J., Gortler, S.J., Hoppe, H.: Texture mapping progressive meshes. In: SIGGRAPH 2001. Proceedings of the 28th annual conference on Computer graphics and interactive techniques, pp. 409–416. ACM Press, New York (2001)
20. Barry, M.: Direct extraction of normal maps from volume data. Technical Report CPSLO-CSC-07-01, California Polytechnic State University (2007)
21. Amidror, I.: Scattered data interpolation methods for electronic imaging systems: a survey. Journal of Electronic Imaging 11, 157–176 (2002)
22. Gortler, S.J., Grzeszczuk, R., Szeliski, R., Cohen, M.: Texture mapping progressive meshes. In: SIGGRAPH 1996. Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, p. 43. ACM Press, New York (1996)