EXACT VISUAL HULL FROM MARCHING CUBES

Chen Liang and Kwan-Yee K. Wong
Department of Computer Science, The University of Hong Kong, Pokfulam, Hong Kong

Keywords: Visual hull, marching cubes, reconstruction.

Abstract: The marching cubes algorithm has been widely adopted for extracting a surface mesh from a volumetric description of the visual hull reconstructed from silhouettes. However, typical volumetric descriptions, such as an octree, provide only a binary description about the visual hull. The lack of interpolation information along each voxel edge, which is required by the marching cubes algorithm, usually results in inaccurate and bumpy surface mesh. In this paper, we propose a novel method to efficiently estimate the exact intersections between voxel edges and the visual hull boundary, which replace the missing interpolation information. The method improves both the visual quality and accuracy of the estimated visual hull mesh, while retaining the simplicity and robustness of the volumetric approach. To verify this claim, we present both synthetic and real-world experiments, as well as comparisons with existing volumetric approaches and other approaches targeting at an exact visual hull reconstruction.

1 INTRODUCTION

The *visual hull*(Laurentini, 1994) mesh reconstructed from silhouettes has many applications in the field of 3D vision. It offers a rather complete description of the scene object, especially for smooth curved objects. In many cases, the reconstructed visual hull mesh can be directly fed to some 3D applications as a showcase, or used as a good initialization for various surface reconstruction algorithms.

Volumetric approach has been widely adopted for reconstructing the visual hull from silhouettes for its simplicity and robustness. An early form of this approach appears in Martin and Aggarwal (Martin and Aggarwal, 1983) where the space is rasterized into parallelogram structure. In subsequent works such as (Chien and Aggarwal, 1986; Potmesil, 1987; Szeliski, 1993), the volume representing the visual hull evolves into a single hierarchical representation termed as the *octree*. The major advantage of using such a representation is its ability to handle objects with complicated topology without compromising the simplicity of its internal data structure. Coupled with the *marching cubes* algorithm (Lorensen and Cline, 1987), a surface can be extracted from the octree for rendering, or serving as an initial mesh for recovering fine details on the surface (Cross and Zisserman, 2000; Hernández and Schmitt, 2004). A major problem of the approaches mentioned so far is that the octree offers only a binary description of the visual hull. Therefore, it is insufficient for the marching cubes algorithm to interpolate vertex positions during the extraction of surface mesh triangles. To increase the accuracy of the reconstructed mesh, more subdivisions of the octree is required, and this leads to a tremendous increase in mesh complexity. Among recent studies that attempt to address the accuracy issue, Mercier et. al. (Mercier and Meneveaux, 2005) casted pixel-rays onto each face of the voxels in order to refine the vertices of the mesh triangles. Unfortunately, this method is computationally expensive and the continuity between adjacent voxels has to be handled explicitly; Erol et. al. (Erol et al., 2005) proposed an adaptively sampled octree to reduce the number of octree subdivisions. They also replaced the binary value at each vertex of the voxels with an estimated weighting. However to compute this weighting, they need to approximate the distance field of the visual hull volume with a 3D grid of values (a field), which in turn, depends on the distance function computed on each silhouette in the first place.

The polyhedral approach is proposed as an alternative to volumetric approach to address the accuracy issue. By freeing itself from discretizing the space,

Liang C. and K. Wong K. (2008).
EXACT VISUAL HULL FROM MARCHING CUBES.
In *Proceedings of the Third International Conference on Computer Vision Theory and Applications*, pages 597-604
DOI: 10.5220/0001080405970604

this approach attempts to compute the exact intersection of viewing cones. The idea was first realized in (Baumgart, 1975), where the visual hull was computed directly as the intersection of polyhedralized viewing cones. Matusik et al. (Matusik et al., 2001) proposed an efficient algorithm capable of computing the polyhedral visual hull in real time, in the case of a few cameras. However, it suffers from numerical instability when more cameras are introduced. Lazebnik et al. (Lazebnik, 2002) derived the visual hull as a topological polyhedron computed from the epipolar constraints. In (Boyer and Franco, 2003; Franco and Boyer, 2003), Franco and Boyer computed exact polyhedral visual hull by cutting and joining the visual rays casted from silhouettes and joining them together. Despite the complexity involved in joining the visual ray segments, the visual hull constructed is highly accurate in terms of silhouette consistency. However, both methods suffer from producing ill-formed mesh triangles.

In this paper, we propose a novel approach for reconstructing an exact visual hull from silhouettes. Our approach is based on the existing octree and marching cubes algorithm. The key to our approach is a simple and efficient strategy to directly estimate the exact positions where the voxel edges intersect with the visual hull. This exact intersection computation will replace the interpolation procedure for locating the vertex position in the traditional marching cubes algorithm. While producing more accurate visual hull mesh, this proposed approach retains the simplicity of the volumetric approach. Compared with the polyhedral approach, the method generates significantly more regular mesh triangles and is much easier to implement. The proposed approach has been verified by both quantitative and qualitative comparisons with other existing volumetric and polyhedral approaches.

2 BACKGROUND

An octree is a tree structure commonly used for representing volume in 3D space. It can be seen as a 3D equivalence of a quadtree (in 2D space) and a binary tree (in 1D space). Each leaf-node corresponds to an actual volume element, also termed as *voxel*, in 3D space. The leaf nodes are attached to non-leaf nodes higher in the tree hierarchy. A non-leaf node does not correspond to a real volume but the bounding box of all its descendants. The root node is thus the bounding volume of the object to be reconstructed.

An octree of an object can be reconstructed from the silhouettes through recursive subdivision and projection tests. The process usually begins with a single

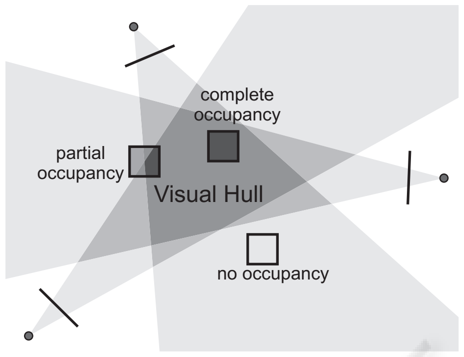

Figure 1: Three types of voxels in octree based visual hull reconstruction: complete occupancy (black), partial occupancy (gray) and no occupancy (white).

voxel. Each voxel is projected onto each image and tested against the silhouette. The test result classifies the voxel, by how much of its volume is occupied by the visual hull, as one of these three types: $\{black, gray, white\}$ (see Fig. 1), which indicate complete occupancy, partial occupancy and no occupancy, respectively. Among the three type of voxels, only voxels with partial occupancy contains the potential visual hull boundary and is subject to further subdivision until certain termination criterion is reached, such as when the maximum allowed number of subdivision is reached.

Once the octree is reconstructed, marching cubes algorithm can be applied to extract the object surface. To do this, the voxel occupancy of a leaf node is encoded into an 8-bit value using the occupancy of its eight vertices. This value is then used to index into a pre-defined lookup table which defines surface triangles within the voxel that will form part of the final visual hull mesh. However, since the octree gives only binary occupancy information, the procedure of computing the triangle vertex by interpolating the voxel vertex values in the standard marching cubes algorithm becomes meaningless. A simple strategy was proposed in (Montani et al., 1994) to use mid-points for the triangle vertices, which unfortunately creates a jagged surface (see Fig. 2(middle)). If we apply smoothing to the mesh to reduce the jaggedness, real features on the surface will also be smoothed out.

We will present in the next section a simple and efficient strategy that can directly estimate the exact positions where the voxel edges intersect with the visual hull. The estimated position can be used in place of the linear interpolation result in the marching cubes algorithm for extracting an accurate visual hull mesh (see Fig. 2(right)).

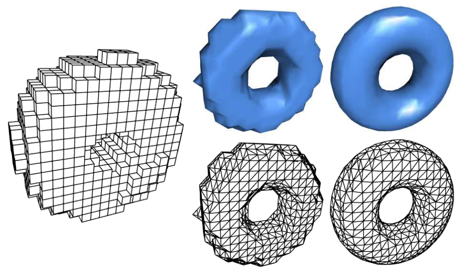
Figure 2: An octree and the mesh extracted from it: (left) The octree; (middle) mesh/surface produced by marching cubes using mid-points of the voxel edges; (right) mesh/surface produced using the exact intersections between the visual hull and voxel edges.

3 MARCHING CUBES FOR EXACT VISUAL HULL

Our approach assumes similar settings as most surface from silhouette algorithms - a set of calibrated cameras denoted as $\mathbf{C}_i$ and the extracted silhouettes. For silhouettes extracted in the form of parametric curves such as B-Snakes (Cipolla and Blake, 1990), a closed-form solution for the visual hull vertices is possible, as will be discussed in Section 3.2. Our approach also accommodates, in a very efficient way, a more common case where the extracted silhouettes are in the form of binary masks $I(x,y) \in \{0,1\}$. This will be discussed in Section 3.3.

3.1 Theoretical Framework

Our proposed approach is based on the following observation, for a voxel edge that intersects with the visual hull, the 3D intersection should project onto the boundary of at least one silhouette, and this projection is the 2D intersection between that silhouette and the projected voxel edge. Given a calibrated camera, we can form a line-to-line projectivity between the projection of the voxel edge and the edge itself, which can be used to obtain the 3D intersection from the 2D intersection with a closed-form solution.

Following the idea of marching cubes, for a voxel edge with two end vertices having different occupancy, the visual hull should have at least one intersection with that edge. In fact, with sufficient number of spatial subdivisions, the case of multiple intersections along one voxel edge will eventually reduce to several voxel edges with exactly one or zero intersection. Our key problem is to determine the exact position of the intersection, under the case of exactly one intersection.

Let us consider an arbitrary edge $\mathbf{e}_0$ of an arbitrary

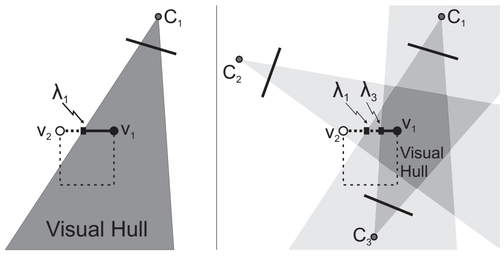

Figure 3: A voxel edge is carved by viewing cones. (left) Only one viewing cone; (right) The combined effort of several viewing cones.

voxel, where the visual hull intersects with the edge for one time. The two ends of the edge, $\mathbf{v}_1$ and $\mathbf{v}_2$, will have different occupancy. Without loss of generality, we assume that $\mathbf{v}_1$ is inside the visual hull volume. When there is only one view, the visual hull is equivalent to the viewing cone constructed from the silhouette in this view, and the intersection between the visual hull and the voxel edge is simply the intersection of this edge with the viewing cone (see Fig. 3(left)). Let us denote the intersection between the edge and the viewing cone of $\mathbf{C}_i$ by λ_i. In the general case with multiple views, the visual hull volume, by its definition, is the intersection of all viewing cones. $\mathbf{e}_0$, as a spatial line segment, is also carved by all these viewing cones. The part of $\mathbf{e}_0$ inside the visual hull should also be the intersection of all $\mathbf{v}_1\lambda_i$. Since $\mathbf{v}_1$ is inside the visual hull and shared by every segment $\mathbf{v}_1\lambda_i$, the real intersection point with the visual hull should be λ_m, where $m = argmin_i |\mathbf{v}_1\lambda_i|$, which corresponds to the intersection with the viewing cone of $\mathbf{C}_m$.

If the cameras have been calibrated, the viewing cones can be constructed from the silhouettes, and the intersection λ_i between the viewing cone of $\mathbf{C}_i$ and $\mathbf{e}_0$ can be computed. In practice, constructing the viewing cones may be complicated and computationally expensive. We can alternatively estimate such intersections in the image space, because the voxel edge $\mathbf{e}_0$ and its projection on the image are related by a line-to-line projectivity which can be readily computed given a calibrated camera.

3.2 Lifting From 2D to 3D

In our local line-to-line projectivity for the edge e, the vertex $\mathbf{v}_1$ is set as the origin, and the positive direction is defined by the vector $\mathbf{v}_2 - \mathbf{v}_1$ (see Fig. 4). Viewing from the image, the intersection λ_i projects to $\mathbf{s}_i$ which is the intersection of the edge's projection with the silhouette. Given the projection matrix $\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}]$ for the camera, the line-to-line projectivity is readily

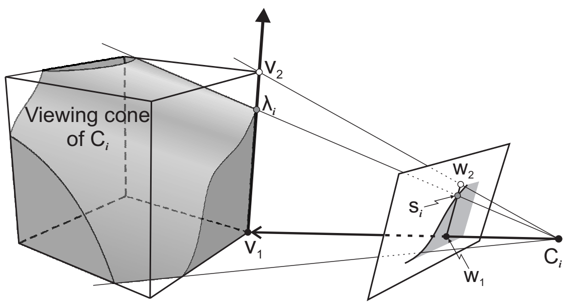

Figure 4: Lifting from 2D to 3D.

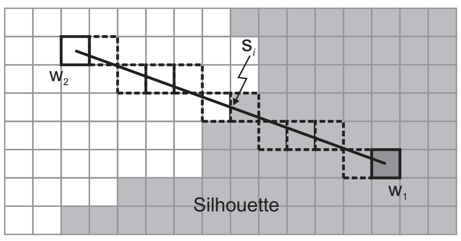

Figure 5: Intersecting a silhouette with a projected voxel edge in the case of binary images. The squares with bold dashed border are the rasterized coordinates of the line $\mathbf{w}_1\mathbf{w}_2$.

known by aligning e with the x-axis of the world coordinate system. To do this, we right-multiply the aligning transformation with $\mathbf{P}$ and we get:

$$\mathbf{P}_{1D} = \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \\ p_{31} & p_{32} \end{bmatrix} = \mathbf{K}\left[\mathbf{R}\left(\frac{\mathbf{v}_2-\mathbf{v}_1}{|\mathbf{v}_2-\mathbf{v}_1|}\right)|\mathbf{t}+\mathbf{R}\mathbf{v}_1\right]$$

Once $\mathbf{P}_{1D}$ is computed, we can use it to lift the image position $\mathbf{s}_i$ to its true 3D position λ_i along the voxel edge. Let us denote $d_i = |\lambda_i - \mathbf{v}_1|$:

$$\begin{bmatrix} kx \\ ky \\ k \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \\ p_{31} & p_{32} \end{bmatrix} \begin{bmatrix} d_i \\ 1 \end{bmatrix}$$

$$d_i = \frac{p_{12} - xp_{32}}{xp_{31} - p_{11}} = \frac{p_{22} - yp_{32}}{yp_{31} - p_{21}}$$

where (x,y) is the 2D coordinate of $\mathbf{s}_i$. Once d_i for every camera is computed, the true intersection between the edge $\mathbf{e}$ and the visual hull is then simply $\mathbf{v}_1 + min\{d_i\}\frac{\mathbf{v}_2-\mathbf{v}_1}{|\mathbf{v}_2-\mathbf{v}_1|}$.

3.3 Speed-up Techniques

An important step in our proposed approach involves determining the intersection between silhouettes and the projected voxel edges. While exact intersections can be computed as a closed-form solution for the silhouettes extracted by B-Snakes or any parametric curves, there are more common situations where the silhouettes are available as binary images $I(x,y) \in \{0,1\}$, providing a maximum accuracy up to the pixel level. In such a case, we can use a modified version of the *Bresenham's line algorithm* (Bresenham, 1965) to perform the intersection computation in a pure integer form so as to achieve a very high computational speed.

The idea is to perform a fast traversal along the rasterized coordinates of a projected voxel edge. As we have already had the projections of $\mathbf{v}_1$ and $\mathbf{v}_2$, which are $\mathbf{w}_1$ and $\mathbf{w}_2$, the rasterized voxel edge on the image can be approximated by the rasterized line connecting $\mathbf{w}_1$ and $\mathbf{w}_2$ (see Fig. 5). A slightly modified version of the Bresenham's line algorithm can then be used here for a quick traversal of the rasterized coordinates along this line. Unlike for the rendering purpose, we do not even need to traverse the whole line, but terminate once the silhouette boundary is reached. The intersection computation is summarized in Algorithm 1, which assumes occupancy of $\mathbf{v}_1$ is always black (i.e., inside the visual hull volume).

Algorithm 1 Silhouette Intersection with Voxel Edge Projection.

```
 1: Initialize $x_1, x_2, y_1, y_2$ and $\varepsilon$ using $\mathbf{w}_1$ and $\mathbf{w}_2$
 2: $s \leftarrow 0, \Delta x \leftarrow x_2 - x_1, \Delta y \leftarrow y_2 - y_1, y \leftarrow y_1$
 3: for $x \leftarrow x_1$ to $x_2$ do
 4:   if $I_i(x,y)$ indicates outside the silhouette then
 5:     exit for
 6:   else
 7:     if $2 * (\varepsilon + \Delta y) < x_2 - x_1$ then
 8:       $\varepsilon \leftarrow \varepsilon + \Delta y$
 9:     else
10:       $y \leftarrow y + 1, \varepsilon \leftarrow \varepsilon + \Delta y - \Delta x$
11:     end if
12:   end if
13: end for
14: $\mathbf{s} \leftarrow \frac{x - x_1}{\Delta x}$
15: $\lambda \leftarrow$ lift-to-3D($\mathbf{s}$)
```

We will show later in Section 4.3 that with above speed-up strategy, our algorithm adds merely marginal computational cost on top of the original octree/marching cubes based approach.

4 EXPERIMENTS

In this section, we compare our proposed approach (ExMC) with the conventional discretized octree/marching cubes approach (MC), as well as the Exact Polyhedral Visual Hull method (EPVH) (Franco and Boyer, 2003) which is capable of producing an exact visual hull mesh from silhouettes. In order to achieve a fair comparison, all three algorithms take the same set of silhouettes in the form of binary images.

4.1 Quantitative Comparisons

We performed quantitative comparisons of the visual hull meshes produced by the three approaches using a series of synthetic experiments. In each experiment, a synthetic model is rendered at some known viewpoints to produce the ground-truth silhouettes in the form of binary images, which are fed to the visual hull reconstruction algorithms to produce a visual hull mesh. The same rendering pipeline is used again to project the visual hull mesh onto the images, which are compared with the ground-truth silhouettes. The error measurement adopted for comparison is modified from the silhouette cost function presented in (Lensch et al., 2001), taking both the miss and false alarm cases into account:

$$Err(S,V) = \frac{\sum (S_i \cap \bar{V}_i) \cup (\bar{S}_i \cap V_i)}{\sum S_i \cup V_i} \in [0,1] \qquad (1)$$

where S_i is the ground-truth silhouette and V_i is the projection of the visual hull on the ith image, $\bar{S}_i$ and $\bar{V}_i$ is the complement of S_i and V_i, respectively.

The synthetic models used range from simple torus to rather complicated models including a knot[95], a bunny and a deer (see Fig. 6(a)-(i)). Table 1 shows the accuracy of the three algorithms, when they are set to produce similar number of triangles in the reconstructed visual hull mesh. Table 2 shows the number of triangles needed by each approach to achieve similar level of accuracy. In addition, we also include the result of MC algorithm with mesh smoothing applied, which is commonly used as a post-processing operation by many octree/marching cubes based algorithms.

In terms of silhouette consistency, it can be seen that our approach performs slightly better than EPVH, and both our approach and EPVH perform significantly better than MC. In addition, our approach generates a much more regular mesh than that of EPVH (see Fig. 6(j)-(m)). Another observation is that although applying smoothing can make the jagged mesh produced by MC visually better, it will smooth out features and tend to shrink the mesh (see Fig. 7(c)), resulting in even higher silhouette inconsistency.

We also compare the performance of MC and our approach under different number of triangles produced. This is achieved by varying the size of the leaf-node voxels in the octree (see Fig. 8). The result shows that although the performance of MC, with or without smoothing, will eventually converge to that of our approach because of smaller voxel size, the resulting mesh will also become over complicated. The advantage of our approach is to produce a reasonably accurate visual hull reconstruction with much

Table 1: Comparison between the silhouette inconsistency $Err(S,V)$ of MC, MC with smoothing, EPVH and our proposed approach when they generate similar number of triangles in the visual hull mesh. The approximate numbers of triangles produced are also shown above.

	Torus	Knot	Bunny	Deer
Triangles	6,000	27,800	23,500	26,400
MC	3.68%	6.28%	1.85%	3.65%
MC (sm)	2.29%	7.04%	2.32%	4.32%
EPVH	0.8%	2.31%	0.76%	1.69%
ExMC	0.51%	1.43%	0.84%	1.36%

Table 2: Comparison between the number of mesh triangles required by MC, MC with smoothing, EPVH and our proposed approach to achieve similar silhouette consistency.

	Torus	Knot	Bunny	Deer
$Err(S,V)$	0.8%	2.31%	0.76%	1.69%
MC	119.8k	164.4k	141.7k	106.8k
MC (sm)	21.9k	98.8k	137.7k	94.7k
EPVH	6.2k	39.6k	23.5k	26.4k
ExMC	2.4k	12.5k	28.2k	18.6k

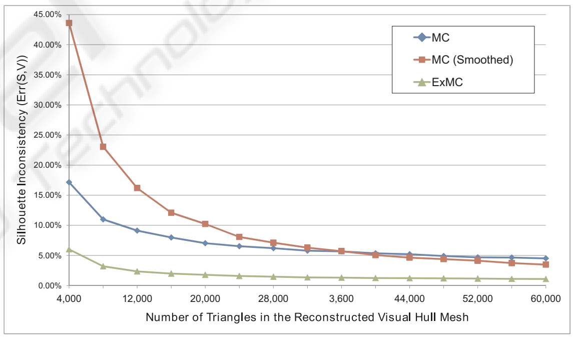

Figure 8: Comparison of silhouette inconsistency ($Err(S,V)$) between MC and our proposed approach with different number of triangles produced. The Knot sequence is used in this comparison.

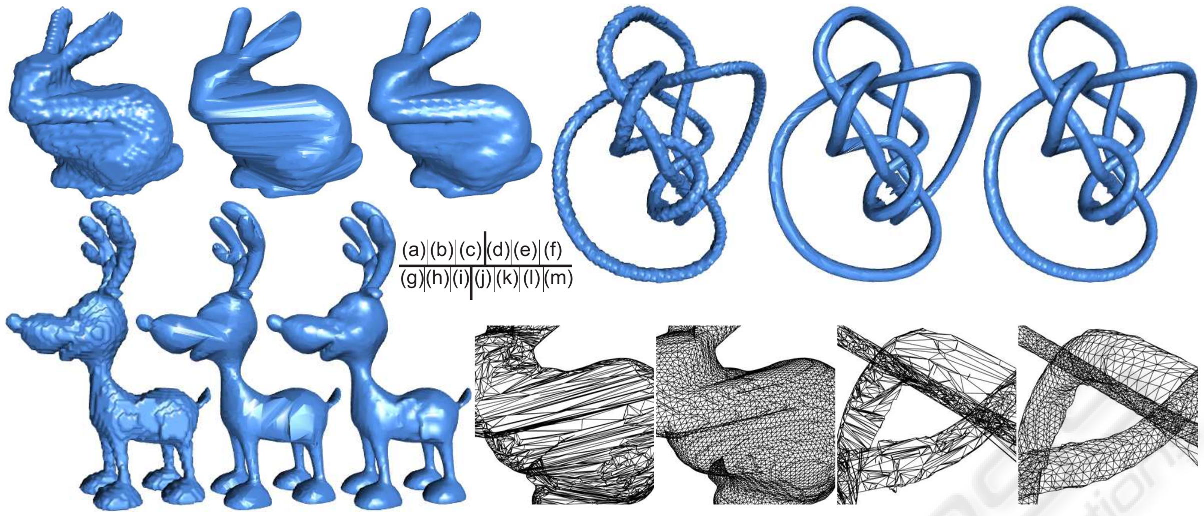

Figure 6: Reconstructed visual hull meshes. (a)-(c) Bunny model reconstructed by MC, EPVH and ExMC using 36 images, respectively. (d)-(f) Knot[95] model reconstructed by MC, EPVH and ExMC using 34 images, respectively. (g)-(i) Deer model reconstructed by MC, EPVH and ExMC using 20 images, respectively. (j)-(k) Close-ups of the reconstructed Bunny by EPVH and ExMC, respectively. (l)-(m) Close-ups of the reconstructed Knot[95] by EPVH and ExMC, respectively.

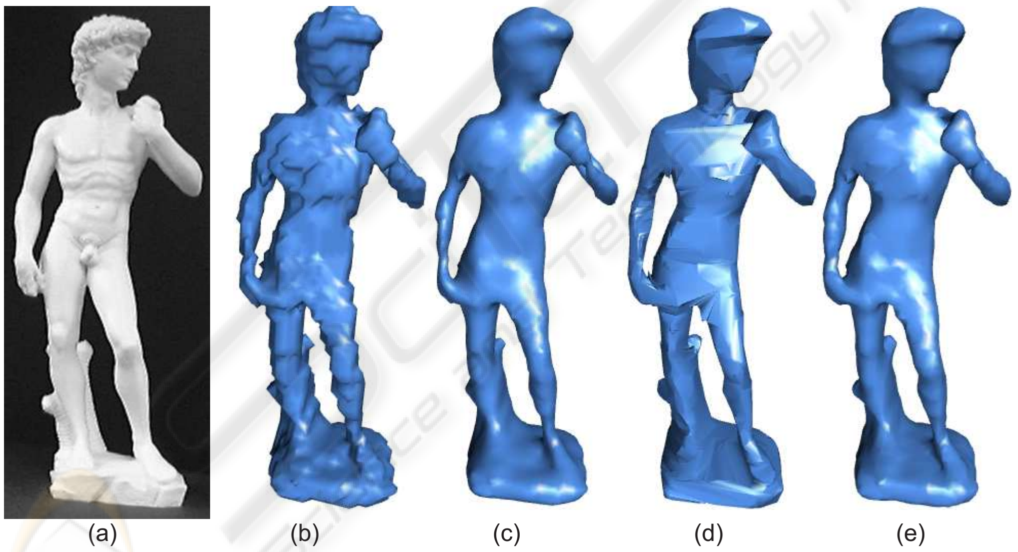

Figure 7: The David sequence (19 images). (a) One of the original images; (b)-(e) Visual hull mesh produced by MC, MC with smoothing, EPVH and our proposed approach, respectively.

larger leaf-node voxel size, and hence less complicated mesh.

4.2 Real World Experiments

In this section, we demonstrate the reconstructions of several real world objects. The image sequences of these objects are acquired with an electronic turntable and calibrated using the method described in (Wong and Cipolla, 2001). The david sequence consists of 19 images, and every visual hull mesh shown in Fig. 7 consists of approximately 8,000 triangles. The dinosaur sequence consists of 36 images and the reconstructed visual hulls have approximately 30,000 triangles (see Fig. 9). The reconstructed left hand of the dinosaur is also magnified for comparison. It can be seen that the result of MC suffers from severe jaggedness. If smoothing is applied, many details are lost and some shrinking effect becomes apparent as well. EPVH, on the other hand, coincides much more

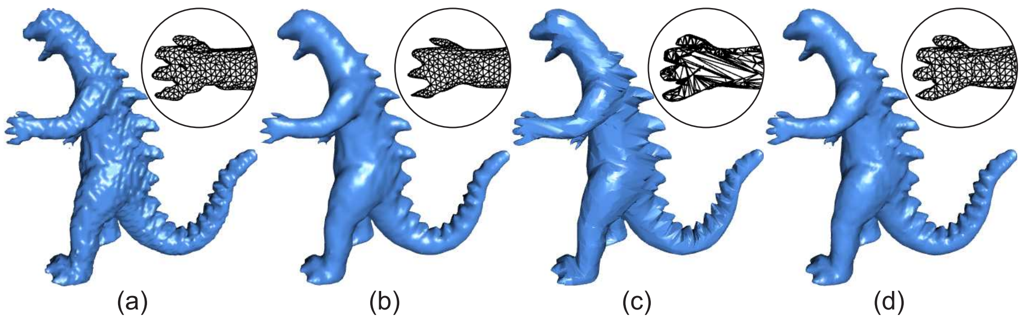

Figure 9: The Dinosaur sequence (36 images). (a)-(d) Visual hull mesh produced by MC, MC with smoothing, EPVH and our proposed approach, respectively.

faithfully with the silhouettes, but it generates quite a few artifacts on the surface as well as ill-formed triangles. Comparatively, our proposed approach caters both the silhouette consistency and the mesh regularity reasonably well.

4.3 Time Comparisons

In Table 3, we show the computation time of above algorithms when they set to generate similar number of triangles. The computation time is measured on a PC powered by a 3.0GHz CPU. We can see that our approach requires only marginal extra computational time than MC. EPVH tends to require more time as the object topology gets more complicated - this is because more time is required to cut each visual ray and connecting them together. Comparatively, MC and ExMC suffer less from the increase in topology complexity.

Table 3: Execution time comparison between MC, EPVH and our approach to produce visual hull mesh with similar number of triangles. The unit is in milliseconds.

	Knot	Deer	David	Dinosaur
Triangles	27.8k	26.4k	8k	30k
MC	2015	1857	373	2926
EPVH	8804	2429	648	5612
ExMC	2404	2072	442	3497

5 CONCLUSIONS

In this paper, we have proposed a modified volumetric approach based on the existing octree/marching cubes approach. We introduce a simple and efficient way to compute exact visual hull vertices, which replaces the interpolation values used in the original marching cubes algorithms. Like its predecessor, the proposed approach is robust with regards to objects with complicated topology and very easy to implement. On the other hand, the proposed approach improves significantly the quality of the reconstructed visual hull, matching those state-of-art polyhedral approaches in terms of accuracy and requires less computational time. In addition, the visual hull mesh produced by the proposed approach is relatively well-formed. This is not only beneficial for rendering purpose, but also caters the need by many surface evolution algorithms as a good initialization.

REFERENCES

Baumgart, B. (1975). A polyhedron representation for computer vision. In *AFIPS National Computer Conference*.

Boyer, E. and Franco, J.-S. (2003). A hybrid approach for computing visual hulls of complex objects. In *Computer Vision and Pattern Recognition*, volume 1, pages 695–701, Madison, Wisconsin.

Bresenham, J. (1965). Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30.

Chien, C. and Aggarwal, J. (1986). Volume/surface octrees for the represntation of three-dimensional objects. *Computer Vision, Graphics and Image Processing*, 36(1):100–113.

Cipolla, R. and Blake, A. (1990). The dynamic analysis of apparent contours. In *International Conference on Computer Vision*, pages 616–623, Osaka, Japan. IEEE Computer Society Press.

Cross, G. and Zisserman, A. (2000). Surface reconstruction from multiple views using apparent contours and surface texture. In *NATO Advanced Research Workshop on Conference of Computer Vision and Computer Graphics*.

Erol, A., Bebis, G., Boyle, R., and Nicolescu, M. (2005). Visual hull construction using adaptive sampling. In *Proceedings of the Seventh IEEE Workshops on Application of Computer Vision (WACV/MOTION'05)*, volume 1, pages 234–241, Washington, DC, USA.

Franco, J.-S. and Boyer, E. (2003). Exact polyhedral visual hulls. In *British Machine Vision Conference*, volume 1, pages 329–338.

Hernández, C. and Schmitt, F. (2004). Silhouette and stereo fusion for 3d object modeling. *Computer Vision and Image Understanding, special issue on 'Model-based and image-based 3D Scene Representation for Interactive Visualization'*, 96(3):367–392.

Laurentini, A. (1994). The visual hull concept for silhouette-based image understanding. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 16(2):150–162.

Lazebnik, S. (2002). Projective visual hulls. Master's thesis, University of Illinois at Urbana-Champaign.

Lensch, H., Heidrich, W., and Seidel, H. (2001). A silhouette-based algorithm for texture registration and stitching. *Journal of Graphical Models*, pages 245–262.

Lorensen, W. E. and Cline, H. E. (1987). Marching cubes: a high resolution 3d surface construction algorithm. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, volume 21, pages 163–169.

Martin, W. and Aggarwal, J. (1983). Volumetric descriptions of objects from multiple views. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 5(2):150–158.

Matusik, W., Buehler, C., and McMillan, L. (2001). Polyhedral visual hulls for real-time rendering. In *Eurographics Workshop on Rendering*.

Mercier, B. and Meneveaux, D. (2005). Shape from silhouette: Image pixels for marching cubes. *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, 13:112–118.

Montani, C., Scateni, R., and Scopigno, R. (1994). Discretized marching cubes. In *IEEE Conference on Visualization*, pages 281–287, Washington, DC.

Potmesil, M. (1987). Generating octree models of 3d objects from their silhouettes in a sequence of images. *Computer Vision, Graphics and Image Processing*, 40(1):1–29.

Szeliski, R. (1993). Rapid octree construction from image sequences. In *CVGIP: Image Understanding*, volume 58, pages 23–32.

Wong, K.-Y. and Cipolla, R. (2001). Structure and motion from silhouettes. In *International Conference on Computer Vision*, volume 2, pages 217–222, Vancouver, Canada.