SAVANT: A new efficient approach to generating the visual hull

Alex Lyons, Adam Baumberg and Aaron Kotcheff

Canon Research Centre Europe Ltd, The Braccans, London Road, Bracknell, Berkshire, UK

Abstract

We present a new approach to generating a boundary representation (a polygonal mesh) of the visual hull from a set of silhouettes of an object taken from known camera positions. The approach uses spatial subdivision to compute the positions of vertices and then traverses the vertices around planar faces to produce a facetted representation of the visual hull. We show that, unlike standard approaches, the method is practical even for a large number of complex silhouettes. It is applicable to any 3D modeling system that uses the "shape-from-silhouette" approach to generating 3D models of objects. The approach can also be extended to efficiently compute polygonal mesh representations of the intersection of a set of arbitrary polyhedral models. This extends its applicability to CAD systems.

Keywords: Solid Modeling, Computational Geometry, Polygonal Modeling, Mesh Generation, Geometric Modeling, Computer Vision, CAD.

1. Introduction

Many 3D modeling systems use the "shape from silhouette" approach to computing the shape of an object from a set of images taken from known positions (e.g. Niem[1], Matusik et al[2]). The approach uses the "visual hull" approximation to the shape, which is the maximum volume that reproduces all the silhouettes of an object[3,4]. A good approximation to the visual hull can be obtained by intersecting the back-projection of a finite set of silhouette images. Alternative approaches, such as those using stereo[5] or voxel coloring[6] rely on matching feature correspondences or on photo-consistency across images. However, the silhouettes can be easily obtained in controlled environments (e.g. with a chroma keying technique[7]). The shape from silhouette approach is therefore capable of producing robust results in a wide-baseline system, where obtaining feature correspondences is difficult, and incorporates information from multiple images in a natural way. In addition, 3D modeling systems are often required to produce a boundary representation of the shape, that is a representation of the boundary as a polygonal mesh, rather than a volumetric representation, in order to efficiently render the model using standard graphics hardware. This can be easily obtained using the shape from silhouette approach. There are two commonly used approaches to generating a mesh representation of the visual hull: volumetric sampling and direct intersection.

The volumetric sampling approach typically uses a voxel grid surrounding the object to produce a "voxel carve"[8,9]. The voxels are often stored in an octree structure to speed

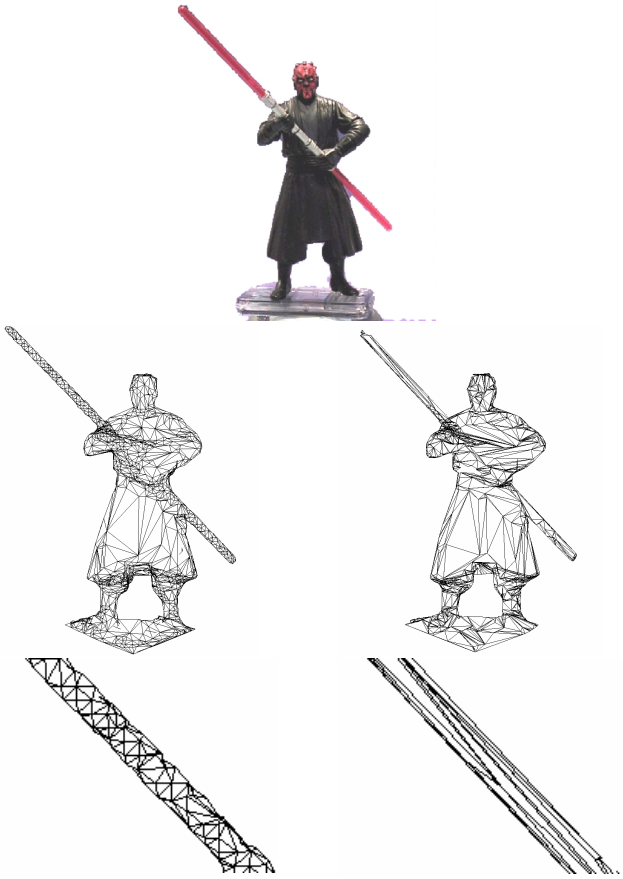

Figure 1: *Drawbacks of the volumetric sampling approach. Top: Example image. Left: Mesh produced by sampling approach. Right: Mesh produced by direct intersection. Bottom: Close-up of long thin structure for the two approaches highlighting the aliasing problem for the volumetric sampling approach.*

up calculations. Nodes in the octree are projected into the silhouette images to determine if they are fully inside or outside the visual hull. In this way a volumetric representation of the visual hull is generated. The volumetric representation is then converted to a boundary representation using the Marching Cubes algorithm (or some variant of it)[10] or simply smoothing the mesh obtained from the visible node faces. Approaches based on voxel carving have the problem that, due to the use of a regular grid, there are often severe alias artifacts at sharp features on the extracted surface (see Figure 1). Recent advances have attempted to overcome these limitations by incorporating a feature detection step that attempts to treat sharp edges and corners separately (see Kobbelt et al[11]). This solution works as long as the object has obvious sharp edge features. However for models obtained from real images it is not always easy to determine the correct threshold for detection of sharp edges and corners. An additional problem with volumetric sampling methods is that they tend to generate models containing an excessive number of faces and vertices. These methods can be very slow when run on a high-resolution grid. They can lead to a large memory requirement and do not efficiently represent the surface in low curvature regions. To avoid this, the model can be post-processed using a mesh optimization algorithm (eg Hoppe et al[12]). However this can produce additional artifacts and is inefficient in low curvature parts of the surface.

Direct intersection avoids these problems by directly generating a polygonal mesh representation of the visual hull without using a regular grid (see Loehlein[13], Matusik et al[14]). This is generated by first approximating each silhouette by a polygon (e.g. using Eu and Toussaint[15]) and then back-projecting each polygonal silhouette to form a set of "polygon cones". The polygon cones are then intersected in a pairwise fashion. This generates the visual hull incrementally. At each step the current polyhedral approximation of the visual hull is intersected with the next polygon cone to obtain a new approximation.

However, we will show that existing direct intersection methods are often too time consuming for a large number of complex silhouettes. This is because the complexity of adding a new silhouette increases with each silhouette added. We have therefore recognised that it is desirable to generate the visual hull using an efficient batch method rather than generating it incrementally. The new approach presented in this paper achieves this by using a spatial subdivision method to directly generate the visual hull from a set of silhouettes of an object taken from known camera positions, taking into account all of the polygon cones simultaneously. We call the approach SAVANT (Silhouette Approximation, Vertex ANalysis and Triangulation). This paper compares the SAVANT algorithm with a standard volumetric sampling approach and with the pairwise intersection approach. SAVANT is shown to be practical even when there are a large number of complex silhouettes. It is applicable to any 3D modeling system that uses shape from silhouettes to generate 3D models of objects. We will also show how the SAVANT algorithm can be extended to efficiently compute boundary representations of the intersection of a set of general polyhedral models. This extends the applicability of the algorithm to CAD systems.

2. Previous Work

2.1 Background

The discussion that follows will consider previous work that provides the context for the SAVANT algorithm. Algorithms for directly generating the visual hull require a set of silhouette images of an object from known camera positions (with known camera parameters). These are approximated by polygons using a standard polygon approximation algorithm (e.g. Eu and Toussaint[15]). The polygon cones are then intersected to generate a mesh representation of the visual hull. The situation is shown in Figure 2 for the simplest case of two silhouette images.

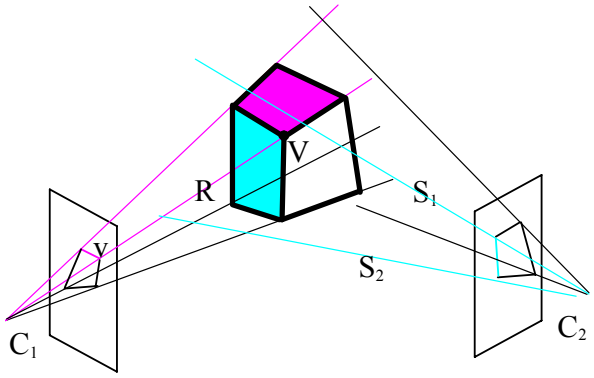

Figure 2: *Intersection of two polygon cones. The polygonal boundary of the visual hull is shown in bold face. The ray R, formed by back-projecting a polygon vertex v in camera C_1 intersects with the infinite triangle formed by the rays S_1 and S_2 in camera C_2 to produce the vertex V.*

In the case illustrated in Figure 2 the calculation of the visual hull is extremely simple. All of the vertices of the visual hull are obtained by intersecting one of the rays formed by back projecting a polygon vertex in one image with one of the infinite triangles formed by back-projecting a polygon edge in the other image. Therefore by testing for intersections of rays from one of the cameras with infinite triangles from the other camera, all of the vertices can be found. The polygonal faces can then be traversed using a boundary traversal algorithm (e.g. Szilvasi-Nagy[16]).

2.2 Intersection of two polyhedra

Szilvasi-Nagy[16] describes an algorithm for determining the

boundary representation of the intersection of two polyhedra using a "plane sweep" approach. The vertices of the intersection (where a face from one of the polyhedra is intersected by an edge from the other one, or where one of the polyhedra has vertices inside the other) are extracted from the projections of the polyhedra into 2 planes. Labels for the 3 faces are stored for each vertex. This information is then used to polygonize the surface of the region of intersection. Further details are explained in Section 2.3.

The approach of Szilvasi-Nagy[16] does not generalize easily to a batch method using more than 2 polyhedra. However, Srinivasan et al[17] developed a plane sweeping algorithm for computing the intersection of multiple polygon cones, using a contour representation, rather than a boundary representation.

2.3 Boundary traversal

Once the vertices of the intersection of two polyhedra have been found, the boundaries of the polygon faces of the intersection polyhedron can then be traversed by labeling the vertices according to triples of faces from the original polyhedra[16]. Szilvasi-Nagy carried this out for only two polyhedra, but we can generalize the approach for the vertices of the visual hull formed from the intersection of an arbitrary number of polygon cones. In this approach, every polygon silhouette edge is given a unique ID. These IDs are used to label faces in the visual hull. Triples of plane IDs are used to label visual hull vertices. Pairs of plane IDs are used to label visual hull edges. The polygon faces are then obtained by using the labels to traverse the edges around each face in order. For example, if the visual hull planes are labeled A,B,C,…, and we are traversing the polygon face associated with plane A, then after connecting vertex {A,B,C} to {A,B,D}, along the {A,B} edge, the next vertex must lie on the {A,D} edge, i.e. it must be {A,X,D} for some new plane X. The new plane is found, suppose it is F, then the next vertex must lie on the {A,F} edge, and so on. This continues, until we return to the starting vertex {A,B,C}. This traversal connects the vertices into sets of polygon faces i.e. it generates a boundary representation of the visual hull. There are some special cases as there is sometimes an ambiguity that needs to be resolved. Further details of this polygon traversal algorithm are contained in Szilvasi-Nagy[16]. Note that for this traversal algorithm to succeed there must be no coincidental meeting of more than 3 planes at a single vertex. However, this is not restrictive in practice, as we can add a small amount of random noise to the input data to ensure that this is always the case, without noticeably affecting the visualization of the final result.

2.4 Incremental visual hull

The direct calculation of the boundary representation of the visual hull can be generalized to an incremental approach for more than two silhouette images, (see Loehlein[13], Matusik et al[14]). We summarize the idea, and improve on their approaches, below.

A typical incremental visual hull algorithm starts with a polyhedron containing the object, for example, the polyhedron defined by intersecting the first two polygon cones. This polyhedron is intersected with the next cone to produce a new polyhedron. This polyhedron is then intersected with the polygon cone from the next silhouette image and so on. In this way the intersection of a polyhedron with a cone can be reduced to a 2D polygon intersection problem. The reduction is achieved by projecting the polyhedron into the view from which the next silhouette image was taken.

Loehlein's original treatment[13] is complicated by the fact that his method involves wiring up the polygon faces in 2D and then filling in the gaps in the surface. Matusik et al[14] use an edge-bin data structure, and the subsequent calculation of the visual hull faces produces a mesh that contains multiple vertices and may contain T-junctions. We have found that a simpler approach is to find all the vertices in the intersection of the polyhedron and the polygon cone, then use triples of faces to label them, and finally traverse the polygon faces using the triples of IDs, in the manner of Szilvasi-Nagy[16] (described in Section 2.3). This is guaranteed to produce a mesh without T-junctions and is considerably simpler than other approaches[13,14].

All the vertices in the intersection of the polyhedron and the polygon cone arise from one of the following 3 cases, which may all be reduced to 2D calculations:

1. Vertices from the original polyhedron that are inside the new polygon cone. To find these in 2D, test whether the projection of the vertex lies inside the new polygon.

2. Intersections of edges of the original polyhedron with planes from the new polygon cone. These are found in 2D by intersecting the projection of the polyhedron edge with an edge of the new polygon.

3. Intersections of edges from the new polygon cone with faces from the original polyhedron. These are found in 2D by carrying out point-in-polygon tests of vertices of the new polygon against the polygons formed by projecting the faces of the original polyhedron.

Figure 3 shows each of the 3 types of vertex, labeled by their type, for the case of the intersection of a cube and a four-sided cone.

For each vertex that has been found, the labels of the 3 planes meeting at the vertex are recorded and the 3D position of the vertex is calculated. Once all the vertices have been found, the boundaries of the polygon faces can then be traversed by using the IDs of the planes, as

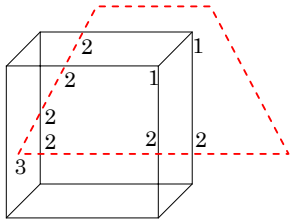

Figure 3: *Intersection of a cube and a four-sided cone. The projected polyhedron is shown (solid line) and the new polygon is shown (dashed line) in the new image. Each of the 3 types of vertex is labeled by their type: (1) vertices from the original polyhedron that are inside the new polygon cone; (2) intersections of edges of the original polyhedron with planes from the new polygon cone; (3) intersections of edges from the new polygon cone with faces from the original polyhedron.*

described in section 2.3. This generates a new polyhedron that may be intersected with further polygon cones in exactly the same way, until all of the polygon cones have been incorporated.

Finally, once all the polygon cones have been incorporated into the model, the faces of the final polyhedron can be triangulated, if required (e.g. using Seidel[18]), and the faces can be texture mapped using image data from the original images, (e.g. in a similar way to Niem[1]).

3. SAVANT: Batch Visual Hull

We have found that the incremental calculation of the visual hull is often too time consuming for a large number of complex silhouettes. This is because the complexity of adding a new silhouette increases as the number of faces and vertices of the current polyhedral approximation of the visual hull increases. In addition the method involves computing a large number of intermediate vertices that may be later discarded by intersecting the polyhedron with the new polygon cone. We recognize that it is therefore desirable to generate the visual hull using an efficient batch method, which takes into account all the polygon cones simultaneously.

A basic "brute force" batch method proceeds by "generate-and-test" as follows. We note that every vertex in the final model can only arise from the intersection of 3 of the polygon cone faces, and that the intersection must lie inside all the polygon cones. Therefore we can generate the complete set of candidate vertices by back-projecting triples of polygon edges, and only keep the candidate vertices whose projection is inside the silhouette in all of the images.

However this method is extremely computationally expensive: if the total number of polygon edges is n, and the total number of images is m, this approach would require $O(mn^3)$ point-in-polygon tests. Clearly this "generate-and-test" approach quickly becomes impractical as n and m become large.

The SAVANT algorithm efficiently finds all of the vertices of the visual hull simultaneously, by combining a bottom-up generate-and-test search with a top-down spatial subdivision to prune the search. The bottom-up search consists of generating and testing candidate vertices, which are formed from triples of polygon cone faces. The top-down pruning consists of projecting 3D regions into the images and eliminating any regions that project to a 2D region that is completely outside any of the silhouettes.

The search for the vertices of the visual hull therefore proceeds by starting with a large initial volume enclosing the visual hull. For simplicity, this may be specified by a set of cubes. Each cube is then processed by either subdividing it, discarding it, or enumerating the vertices within it, on the basis of the projections of the cube into the silhouette images. The subdivided cubes are then processed in the same way and the calculation continues until the entire initial region has been processed. The advantage of proceeding in this way is that large regions can be discarded without further calculation, therefore avoiding the combinatory explosion of brute force calculation. Further implementation details are discussed in Sections 4 and 5.

Once the vertices of the visual hull have been obtained, the boundary of the polygonal faces of the visual hull can be traversed, by using the IDs of the planes, as has already been described in section 2.3.

3.1 Summary

To summarize, SAVANT is a new algorithm for efficiently computing the boundary representation of the visual hull. The steps in the SAVANT algorithm are as follows:

1. Approximate the silhouettes with polygons and give each polygon edge a unique ID.

2. Calculate the vertices of the visual hull and label them with triples of IDs.

3. Generate the polygon faces of the visual hull.

The SAVANT algorithm makes step (2) efficient by searching for the vertices of the visual hull using a combination of a bottom-up search with a top-down spatial subdivision to prune the search. This makes the algorithm for batch visual hull computation practical, even for a large number of complex silhouette images.

The next section describes the steps involved in calculating the vertices of the visual hull in more detail.

4. Calculating the vertices

4.1 Finding the initial volume

An initial volume needs to be defined which encloses the visual hull. Since SAVANT uses projections of 3D regions into the images to infer information about what is contained within the region, the initial volume needs to lie completely in front of all of the camera optical centers.

One way to define the initial volume is to define it as the union of a set of cubes. First a very large cube is found which encloses the object, but which may not lie completely in front of all the cameras. This initial cube is placed on a stack.

Cubes are taken from the stack and processed as follows:

- If the cube lies behind any camera or the cube is smaller than some predetermined size, it is discarded;
- else, if the cube is in front of all the cameras, it is added to the initial volume;
- else the cube is subdivided and its children are placed on the stack.

This continues until the stack is empty. In this way the initial volume is defined, consisting of the union of a set of cubes, and is guaranteed to lie entirely in front of all the cameras and to be the largest such region up to the tolerance given by the minimum cube size.

4.2 Processing cubes

The region being processed consists of a set of cubes that can be processed in any order. It is therefore convenient, as in the calculation of the initial volume, to place the cubes in a stack-like structure. A cube is taken from the stack and is then either subdivided, discarded, or the vertices within it are calculated, on the basis of the projections of the cube into the silhouette images. If a cube is subdivided its children are placed onto the stack. The calculation terminates when there are no more cubes on the stack.

In the simplest version of the algorithm we carry on subdividing cubes until there is at most one candidate vertex in the cube. To determine whether there is a candidate vertex in the cube we note that each vertex arises from the intersection of 3 of the polygon cone faces and it must lie within all of the other polygon cones. We therefore count the total number of polygon cone faces that intersect the cube. This can be done by projecting the cube into all of the images and counting the total number of polygon edges that intersect with the projections of the cube.

There is one candidate vertex if both the following conditions hold:

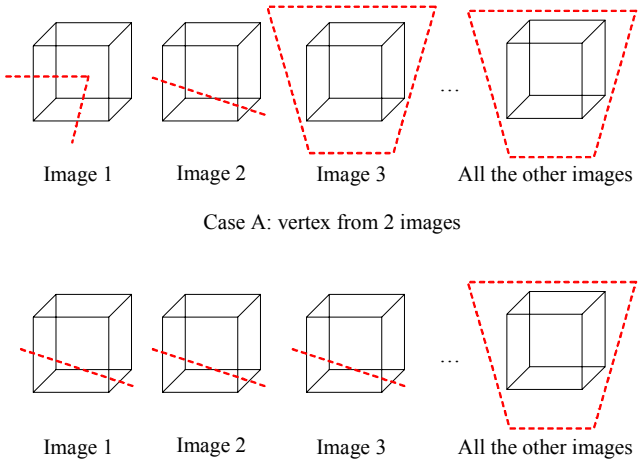

Figure 4: *Candidate vertex from 2 or 3 images. The projected cube in each image is shown as a solid line and the polygon edges are shown as a dashed line.*

1. There are exactly 3 polygon edges that intersect with the projection of the cube, involving edges in at least 2 images.
2. There are no images in which the projection of the cube is completely outside the silhouette.

The two cases for condition (1) that need to be considered are shown schematically in Figure 4.

Before a candidate vertex can be accepted, the algorithm needs to calculate whether the back-projections of the 3 polygon edges intersect, and if they do intersect, whether the intersection point is within the cube. This is the only 3D test that needs to be done. If the intersection point lies within the cube then we have found a true vertex of the visual hull. The vertex is stored (i.e. its position and the labels of the 3 polygon edges which generated it) and the cube is then discarded.

Cubes that do not contain a single candidate vertex are either subdivided or discarded according to the following rule:

Any cube that has less than 3 planes intersecting it is discarded, as it cannot contain a vertex of the visual hull. In addition, any cube whose projection in any of the images is outside the silhouette is also discarded, as it must then lie entirely outside the visual hull. Also, if the projected cube intersects with polygon edges only in a single image then the cube is discarded, as the intersection of the polygon cone faces would be at the optical center, which is assumed to be outside the initial region. If none of these conditions holds, the cube is subdivided. In its simplest version, the cube subdivision produces 8 child cubes, which are placed

on the stack. The calculation continues until the stack is empty.

The SAVANT algorithm for calculating the vertices of the visual hull cubes may be summarized by the following pseudo-code:

```
Define initial region enclosing visual hull
Push cubes in initial region onto stack
While stack not empty
{
  Pop cube from stack
  Project cube into silhouette images
  If one of the 2 cases in Figure 4 holds
  {
    There is a candidate vertex
    Generate position P of candidate vertex
    If P is in the cube
    {
        Store verified vertex position and labels
    }
    Discard cube
  }
  Else if (there are less than 3 planes intersecting the cube
          OR the cube projects outside one of the silhouettes
          OR cube intersects silhouette in exactly one image)
  {
    There are no candidate vertices in the cube
    Discard cube
  }
  Else
  {
    Subdivide cube
    Push children onto stack
  }
}
```

Figure 5: *Pseudo-code for SAVANT vertex calculation.*

5. Implementation

We have implemented a number of extensions to the basic algorithm that speed it up considerably. These are described below.

1. In the simple version described above the cube is projected into all the silhouette images. However this is unnecessary, as the projection of a parent cube provides information about its descendants. In particular, we can reduce the number of times cubes need to be projected into images by realizing that, if the projection of a cube is entirely within a silhouette then the projection of all its children will also lie within that silhouette. Therefore that silhouette image no longer needs to be tested for that cube or any of its descendants. We can therefore exploit this by storing a list of "active images" with each cube. The children of the cube inherit this list. Initially every image is on the active image list, but if a cube projects to a region entirely within a silhouette then that image is removed from that cube's active image list.
2. There are potentially a large number of intersection tests between polygon edges and the projections of a cube into the images. These intersection tests can be sped up considerably by storing the bounding boxes of the polygon edges in a quad tree. Many of the intersection tests can be avoided by first carrying out a "conservative test" for collisions of the bounding box of the projection of the cube against this quad tree. If the conservative test shows that there is no collision, then the cube cannot intersect the polygon edge and so there is no need to carry out the full intersection test[19].
3. We have found that it is more efficient for the algorithm to stop subdividing cubes before the number of candidate vertices drops to a single candidate. We stop subdividing and start exhaustively generating candidate vertices when the total number of intersections between the projected cube and the polygon edges drops below some suitable threshold, n. When this occurs we enumerate all the triples of polygon edges that intersect the projections of the cube in two or three images. For each triple we generate the candidate vertex and test whether its position is inside the cube. We then output a list of vertices within the cube rather than a single vertex. Our experiments have shown that, for our implementation, a suitable value for n is 15.
4. Many unnecessary calculations can be avoided by early termination of the tests. For instance as soon as the algorithm has found a silhouette image in which the projection of the cube lies entirely outside the silhouette, then the cube can be discarded without further calculation. Also, during the enumeration of the intersections between the projections of the cube and the silhouette boundaries, as soon as the number of intersections becomes greater than the threshold, n, the cube can be subdivided without further processing of that cube.

6. Extensions

The SAVANT approach is not restricted to computations of the intersection of a set of polygon cones. It can be generalized to calculate the intersection of a set of general polyhedral models. This extends the applicability of the SAVANT to CAD modeling systems, in which boundary representations of unions and intersections of polyhedral models can be computed. The key difference from the polygon cone case is that there is no longer a set of silhouette images to project the cubes into. Instead all the calculations are carried out in 3D. We give every face of every polyhedron a unique ID. Then the algorithm proceeds by building an octree to store the faces of the polyhedra. The tests as to whether to discard, subdivide, or generate candidate vertices at a node in the octree are carried out while the octree is being built. We calculate the number of faces in the octree node and only subdivide if there are more than 3 faces and the cube representing the node is inside all the remaining polyhedra. If there are exactly 3 faces we generate the intersection point and test that it is

within the cube representing the node and within all the remaining polyhedra. If it is we add the vertex to the list and label it with the triple of IDs of faces that intersect there. Once the vertices have been found, the boundary of the intersection polyhedron can be traversed, using the labels of the vertices, as described in Section 2.3.

All the extensions described in Section 5 to speed up the algorithm (except extension (2)) can also be implemented in the generalization of SAVANT to the intersection of a set of polyhedra. To generalize extension (1), the "active image" list is replaced by an "active polyhedron" list, which is stored at each node. If a cube is entirely within a polyhedron, then the polyhedron is removed from the "active polyhedron" list for that node and all of its descendents. Extension (3) also generalizes: we can stop subdividing a node when the number of faces in the node drops below a threshold, *n,* and then generate and test a list of candidate vertices in that node, rather than a single one. Extension (4) also generalizes: as soon as the node is found to be completely outside a polyhedron then the node can be discarded, without further calculation, and during the enumeration of the intersections between the node and the polyhedra, as soon as the number of intersections becomes greater than *n*, the node can be subdivided without further processing.

7. Results

7.1 Comparison with incremental method

We carried out tests comparing the runtime performance of the SAVANT visual hull algorithm with the incremental visual hull algorithm described in Section 2 on both real and synthetic image sequences. We observed how the performance depends on the complexity of the model and on the number of images in the sequence. All timings were obtained using a 650MHz Pentium III PC with 128MB memory.

The tests used automatically segmented silhouettes from images taken using a conventional digital camera. Figure 6 shows typical examples of the data sets used and the meshes obtained (rendered with smooth shading). These are the raw results from direct intersection so the meshes are quite irregular and the shading algorithm shows up a few artifacts in the meshes. The meshes can be made more regular if required using a mesh fairing algorithm, such as described by Taubin[20].

A table comparing the timings for the two algorithms for each of these examples is shown in Figure 7. The table shows the number of images, the total number of edges in all the input polygon silhouettes, the triangle count of the final model and timings for the incremental algorithm and the SAVANT algorithm for each example. For a very simple example, such as the Duck, the two algorithms perform similarly. The other two examples, which are more complex than the Duck example, show that SAVANT performs between 3-5 times faster than the incremental algorithm.

Figure 6: *Example data sets. Top: original images. Bottom: Meshes produced by direct intersection (smooth shaded).*

	Duck	**Helmet**	**Fan**
Number of images	15	31	88
Input polygon edge count	1666	7656	28412
Output model triangle count	6188	23956	15028
Incremental algorithm timing (sec)	16.1	519	921
SAVANT algorithm timing (sec)	15.8	102	282

Figure 7: *Timing comparisons between SAVANT and incremental algorithm for real data sets.*

For the most complex model (in terms of output triangle count) the Helmet, the SAVANT algorithm exhibits the greatest improvement in performance over the incremental algorithm. We have found that generally, as the model complexity increases, so does the advantage of using the SAVANT algorithm, compared with the incremental algorithm.

We then compared the behaviour of the two algorithms as a function of model complexity, keeping the number of images in the sequence fixed while varying the complexity of the final model by varying the threshold used to approximate the input silhouettes with polygons. Figure 8 shows the results for a synthetically generated set of 30 images of a sphere, with randomly located camera positions.

Number of triangles in final model	Incremental algorithm timing (seconds)	SAVANT algorithm timing (seconds)
5996	18	33
13804	86	53
26364	291	76

Figure 8: *Dependence on complexity of model with number of images fixed (synthetically generated sphere data set).*

The comparisons with the incremental algorithm show that, while the incremental algorithm is acceptable for silhouettes of low complexity, as the complexity of the silhouette images increases, the runtime of the incremental algorithm increases substantially more rapidly than the SAVANT algorithm.

Further results were generated using real data. The graph in Figure 9 shows similar behaviour to Figure 8, but for a more complex real model, in this case the Helmet data set, not simply a synthetic sphere model. This shows that, in contrast to the incremental algorithm, whose performance depends on the complexity of all the intermediate polyhedra generated during the processing, the complexity of SAVANT is roughly linear with the complexity of the final model, as measured by the number of triangles. This shows that the greater the complexity of the model, the greater is the performance improvement of SAVANT over the incremental algorithm.

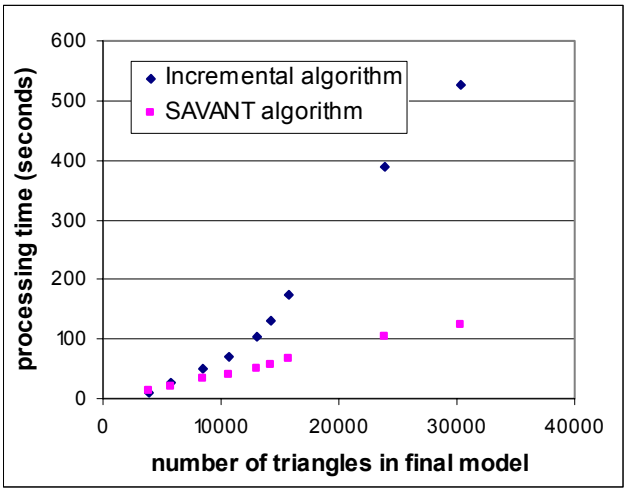

Figure 9: *Dependence on complexity of model with number of images fixed (Helmet data set).*

Finally we compared the performance of the two algorithms as a function of the number of images in the sequence. The results are shown in Figure 10, for the synthetically generated sphere data set.

The behavior of the incremental algorithm can be explained as follows. As the number of images increases, the complexity of the model also increases. Therefore the runtime of the incremental algorithm is actually worse than linear in the number of images, as the later images have a more complex mesh to intersect and this dominates the runtime behavior. In contrast, the runtime of the SAVANT algorithm is roughly linear in the complexity of the final model. This explains why the SAVANT performance improvement is greater as the number of input images increases.

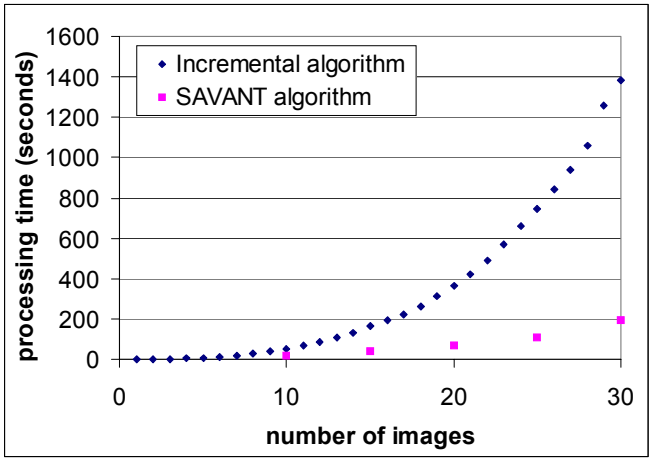

Figure 10: *Dependence of SAVANT and incremental algorithm on number of images in sequence (synthetically generated sphere data set).*

7.2 Comparison with volumetric sampling approaches

There are two commonly used volumetric sampling approaches to generating a boundary representation of the visual hull. The first approach typically uses a voxel grid surrounding the object to produce a "voxel carve"[8, 9]. The voxels are often stored in an octree structure to speed up calculations. Nodes in the octree are projected into the silhouette images to determine whether they are fully inside or outside the visual hull. In this way a volumetric representation of the visual hull is generated. The volumetric representation is then converted to a boundary representation using the Marching Cubes algorithm (or some variant of it)[10] or simply by smoothing the mesh obtained from the visible node faces. The second approach, which is similar, avoids the full octree, and should be more accurate for a given cube size. This uses the fact that the "inside" function, determining whether a point lies inside the visual hull, is defined continuously everywhere in space, not just on a voxel grid. Therefore, the Marching Cubes algorithm can be used to find a boundary representation of the zero crossings of this function, without first calculating the full volumetric representation of the visual hull. When a cube is found which has an edge along which the "inside" function changes, then the intersection of the zero crossing

surface with that edge can be found by binary intersection. These intersection points are then used as the vertices of the triangulation produced by Marching Cubes.

We compared the quality of the results obtained from the SAVANT algorithm with those obtained from the second of these sampling approaches. A typical image used and the meshes produced by the two algorithms (shown as wireframes) are shown in Figure 1.

The results clearly show the problems associated with sampling approaches. Firstly, there is an aliasing problem, which means that in order to approximate the long thin structure to the same degree of accuracy as SAVANT, the sampling approach requires an extremely small grid size. This is not alleviated by intersecting the zero crossing surface with the edges of the cubes, as the positions of the vertices are still quantized in the other two directions and so the method suffers the same aliasing problem as the voxel carving approach. The SAVANT algorithm is able to approximate the long thin structure much more efficiently because it directly uses the polygonal approximation of the silhouette images. Secondly, the use of a small grid size produces an excessive number of triangles in the final model. This can be very slow, leads to a large memory requirement, and requires the use of a mesh optimization algorithm.

In conclusion, the volumetric sampling approach produces more triangles but the result is less accurate than a direct intersection approach, such as SAVANT.

8. Conclusions

We have developed an algorithm for directly computing a boundary representation (a polygonal mesh) of the visual hull from a set of silhouette images. This avoids the aliasing problems associated with methods based on a volumetric sampling, such as those using Marching Cubes. There do exist direct incremental methods for generating the visual hull that avoid the aliasing artifacts associated with sampling-based methods, but these are slow for complex models. The advantage of SAVANT over these is that ALL the cones are intersected simultaneously, giving a significant efficiency gain over incremental approaches, particularly when dealing with a large number of complex silhouettes.

The key to the success of the SAVANT algorithm is that it combines a bottom-up generate-and-test search for the vertices of the visual hull with a top-down spatial subdivision to prune the search. This is ultimately what makes batch computation of the visual hull practical.

Several extensions of SAVANT have been described and implemented. In these, information about the projections of regions into silhouette images is cached, quad trees are used to speed up the intersection tests and there is control over the point at which the algorithm stops subdividing regions and starts generating candidate vertices.

We have also described how the SAVANT approach of combining bottom-up generate-and-test with top-down region pruning can be extended to calculations of the intersection of a set of polyhedra, rather than just the visual hull. This extends its applicability to CAD systems.

In conclusion our new SAVANT algorithm has been shown to provide a more efficient and practical method than standard approaches for computing the visual hull from a set of silhouette images and for computing polyhedron intersections. SAVANT gives more accurate results than volumetric sampling methods and is faster than other direct methods with no loss of accuracy.

Acknowledgements

The work was supported by Canon Europa NV and Canon Inc. The authors would like to thank Simon Rowe for some helpful discussions at the beginning of this work and for useful comments during the drafting of this paper. Special thanks go to James Stevenson of the Victoria and Albert museum for providing the Helmet data.

References

1. Niem, W., Automatic Reconstruction of 3D Objects using a mobile camera, Image and Vision Computing (17) 1999, 125-134.
2. Matusik, W., Buehler, C., Raskar, R., Gortler, S. J., McMillan, L., Image based visual hulls, ACM SIGGRAPH Computer Graphics, Annual Conference Series, 2000, 369-374.
3. Laurentini, A. "The Visual Hull Concept for Silhouette Based Image Understanding." IEEE PAMI 16,2 (1994), 150-162.
4. Vaillant, R. and Faugeras, O. D., Using extremal boundaries for 3D object modelling, IEEE PAMI 14,2 (1992), 157-173.
5. Okutomi, M. and Kanade, T., A Multiple-Baseline Stereo. IEEE Trans. On Pattern Analysis and Machine Intelligence volume 15 no.4, April 1993, 353-363.
6. Seitz, S. M. and Dyer, C., Photorealistic Scene Reconstruction by Voxel Coloring, CVPR 97, 1067-1073.
7. Smith, A. R. and Blinn, J. F., Blue Screen Matting, ACM SIGGRAPH Computer Graphics, Annual Conference Series, 1996, 259-268.
8. Potmesil, M., Generating octree models of 3D objects from their silhouettes in a sequence of images, Computer Vision, Graphics and Image Processing 40 (1987), 1-29.

9. Szeliski, R., Rapid octree construction from image sequences, CVGIP: Image Understanding, Vol 58, No 1, 1993, 23-32.
10. Lorensen, W. E., Cline H. E., Marching cubes: A high resolution 3D surface construction algorithm, ACM SIGGRAPH Computer Graphics, Annual Conference Series, 1987, 163-169.
11. Kobbelt, L. P., Botsch, M., Schwanecke, U., Seidel, H-P, Feature sensitive surface extraction from volume data, ACM SIGGRAPH Computer Graphics, Annual Conference Series, 2001, 57-66.
12. Hoppe H., DeRose T., Duchamp, T., McDonald, J., and Stuetzle, W., Mesh Optimization, ACM SIGGRAPH Computer Graphics, Annual Conference Series, 1993, 19-26.
13. Loehlein, M., A Volumetric Intersection Algorithm for 3d-Reconstruction Using a Boundary-Representation, http://i31www.ira.uka.de/diplomarbeiten/da_martin_loehlein/Reconstruction.html
14. Matusik, W., Buehler, C. and McMillan, L., Polyhedral Visual Hulls for Real-Time Rendering, Proceedings of the 12th Eurographics Workshop on Rendering, London, England, June 2001, 115-125.
15. Eu, D., and Toussaint, G. T., On Approximating Polygonal Curves in Two and Three Dimensions, CVGIP: Graphical Models and Image Processing, Vol. 56, No. 3, 1994, 231-246.
16. Szilvasi-Nagy, M., An algorithm for Determining the Intersection of Two Polyhedra, Computer Graphics Forum 3 (1984), 219-225.
17. Srinivasan, P., Liang, P., and Hackwood, S, Computational Geometric Methods in Volumetric Intersection for 3D Reconstruction, Pattern Recognition, Vol. 23, No. 8, 1990, 843-857.
18. Seidel, R., A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. Computational Geometry: Theory and Applications, 1(1), 1991, 51-64.
19. Greene, N., Detecting Intersection of a Rectangular Solid and a Convex Polyhedron, Graphics Gems IV edited by Paul Heckbert, Academic Press, 1994, 74-82.
20. Taubin, G., A signal processing approach to fair surface design, ACM SIGGRAPH Computer Graphics, Annual Conference Series, 1995, 351-358.