Vision, Video, and Graphics (2003)
P. Hall, P. Willis (Editors)

3D S.O.M. – A commercial software solution to 3D scanning

Adam Baumberg, Alex Lyons, Richard Taylor

Canon Resarch Centre Europe, The Braccans, London Road, Bracknell, Berkshire RG12 2XH
http://www.cre.canon.co.uk

Abstract

This paper describes the novel features of a commercial software-only solution to 3D scanning - the 3D Software Object Modeller (3D S.O.M.). Our work is motivated by the desire to produce a low-cost, portable 3D scanning system based on hand-held digital photographs. We describe the novel techniques we have employed to achieve a robust software-based system in the areas of camera calibration, surface generation and texture extraction.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]:

1. Introduction

Conventionally generating photo realistic 3D models has been an expensive and time-consuming operation. To automate this process a wide variety of 3D scanning devices is available, for example using laser-based systems (e.g. Cyberware body scanners), structured light systems as well as passive systems (e.g. Geometrix LightScribe). One particular problem with laser-based and structured light systems is that they do not work well with shiny reflective objects and it is often necessary to coat the surface with a non-reflective layer (e.g. dust with chalk) which can result in unsatisfactory texture data.

Our work is motivated by the desire to produce a low-cost, portable 3D scanning system based on hand-held digital photographs. The only hardware requirements for our system are a camera, a black & white calibration pattern printed on any standard printer and a PC. We designed the system to be as easy to use as possible although a certain amount of skill is required in taking the photos and editing any artefacts in the output model. The target model quality is "reasonable looking" textured 3D models suitable for 3D on the web. In general the more expensive hardware systems are capable of producing very high quality 3D models. However, we have found that our low-cost software solution can often produce comparable results, especially when comparing models suited to bandwidth limited e-commerce applications. In this context the appearance (and file size) of the textured model is much more important than the geometric accuracy.

On a system level our approach is similar to that of Niem [1]. Like Niem we use a calibration object to ensure accurate camera parameter estimation for arbitrary objects. Similarly to ensure the system can handle untextured or reflective objects and uncontrolled lighting we also use a "shape from silhouettes" approach rather than rely on stereo feature matching [2] or colour consistency [3]. In contrast to Niem, our system benefits from a new robust calibration object, a novel batch technique for exact computation of the "visual hull" and a novel texture blending scheme.

Currently we extract a simple texture map to represent the appearance of the 3D surface. Structured or controlled lighting techniques such as Bernardini [4] allow the recovery of albedo and normal maps for more accurate rerendering of an object in new lighting conditions. However, we chose an uncontrolled lighting solution to reduce the system cost and simplify image acquisition.

An alternative to explicitly modeling the surface reflectance properties is to use an image based rendering approach. Macmillan describes such a technique where images of an object are acquired from multiple viewpoints and with multiple lighting conditions [5]. This approach can produce very high quality results but is not well suited for producing small sized models suitable for low bandwidth applications. The image acquisition step is also expensive and potentially slow. Proprietary image-based solutions such as QTVR suffer from large file size and also require large numbers of input images to ensure smooth viewpoint changes when rotating the model.

2. System Overview

In order to engineer a reliable commercial modelling system we chose to base our solution on two well understood simple techniques – a calibration object is introduced into the scene and we use a "shape from silhouettes" approach.

To automate silhouette detection we utilise a uniformly coloured backdrop placed behind the object. The segmentation is further improved with the novel use of a stand to separate the object from the calibration object.

To summarise, the main steps involved in the system are as follows:

1. **Photograph object** - The object is placed on a stand on a planar calibration mat with a coloured backdrop placed behind the object. The object and stand are then rotated by hand and around 15 to 50 photographs taken of the object from various different angles.
2. **Estimate Cameras** - As the calibration mat and object are moved together, we can estimate the camera parameters (intrinsics and extrinsics) in a global object-centred coordinate frame for each view. The mat has been carefully designed with easily detectable features (see section 3.1).
3. **Detect Binary Silhouettes** - The coloured backdrop is automatically detected and the remaining image is labelled as foreground for each view. The segmentation scheme is a fairly standard colour-key technique [6].
4. **Generate Surface Mesh** - The surface model is generated using a "shape from silhouettes" approach. We have implemented a novel batch method for fast and efficient calculation of the exact "Visual Hull" (see section 4).
5. **Texture Map Generation** - We have implemented a robust and efficient technique for blending the image data to generate a consistently textured model without an associated loss of texture detail (see section 5).

The system is semi-automatic allowing the user to recover from individual component failures or shortcomings. We have provided an easy-to-use, simple GUI that provides additional editing functionality.

3. Estimating the Cameras

3.1. The 3D S.O.M. Calibration Target

We require a calibration target which can be placed in a scene to enable the accurate and robust recovery of the unknown camera parameters.

Previously seen targets (e.g. Niem [1], Gortler [7]) require coloured features (less reliably detected in uncontrolled lighting) or complicated detection schemes. Niem for example requires the reliable detection of thin line features which can easily become obscured by shadows, highlights etc. The Niem pattern consists of two concentric thin circles joined with radial line segments printed on a coloured background. A complex non-linear ellipse fitting scheme is required along with robust line fitting with pairs of opposite line segments. A major drawback of Niem's approach is that an erode-dilate operator is used to separate the mat pattern from the object silhouette which prohibits thin structures in the object silhouettes being successfully modelled. Similarly, Gortler requires the accurate detection of dark-cyan thin circles on a light-cyan background.

In contrast our target consists of very simple features that can be reliably detected with a very straight-forward detection scheme. The main design criteria were – *Accurate feature detection*, *Robust to lighting variation*, *Robust to object and background detail*, *Low cost*.

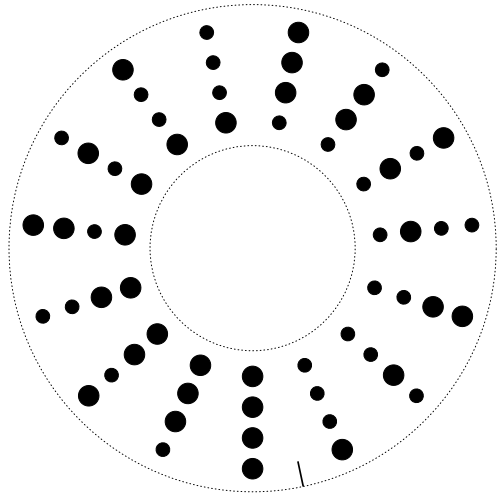


Figure 1: *3D S.O.M. Calibration Target*

The target is shown in figure 1. We used a planar target because it is cheap. The target can be printed using an ordinary black and white printer.

The pattern consists of 15 groups of 4 dots arranged radially around a large circle. Dots are used because they are easy to identify, even when projected arbitrarily. The size and number of dots is a compromise between larger features (easier to detect) and a large number of features (give a better estimate of the camera parameters).

Groups are used to distinguish between dots on the object (or background) and the dots on the target - dots are common, sets of four dots are rare. A line of four dots is used because this feature also possesses a projective invariant, the cross-ratio.

3.2. Detection Algorithm

The target is detected in an image using the following steps,

- Detect Dark Blobs

- Cluster Blobs
- Centre Voting
- Label Groups and Output Correspondences

which are described in more detail in the following sections.

3.2.1. Detect Dark Blobs

Dots are dark on light, therefore for a pixel with colour (R,G,B) it is part of a dot if each component has intensity less than 50%. This gives a binary image with 1 where there are dots (and other dark objects) and 0 elsewhere.

A standard 4-connected region labeling algorithm is used to extract blobs from the binary image. The algorithm keeps track of the number of pixels N and the sums of the x-coordinates $\sum x$ and y-coordinates $\sum y$ to determine the centroid $(\sum x/N, \sum y/N)$ and area N of each dot.

3.2.2. Cluster Blobs

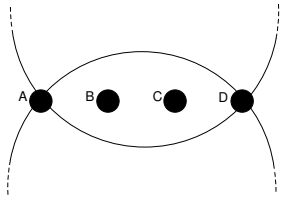


Figure 2: *Cluster Cross Ratio And Proximity*

Typically there will be several hundred blobs found in the thresholded image, whereas at most 60 are of interest to us. Object and background blobs are eliminated by clustering blobs into sets of four which lie on a straight line. Additionally, the cross ratio (a projective invariant) of the points (shown in figure 2) is known,

$$\text{cross ratio} = \frac{AC}{AD} : \frac{BC}{BD} = \frac{2}{3} : \frac{1}{2} \qquad (1)$$

and this is a powerful test for rejecting accidentally aligned dots. A further criterion is that there should only be two dots between the endpoints i.e. within both circles in figure 2 which approximate a 4×1 ellipse.

3.2.3. Centre Voting

To determine the orientation of the clusters (2-way rotational ambiguity) we need to know the centre of the big circle (very approximately). This is done using a voting method similar to the Hough transform. Each cluster votes for all the points on a line through the blob centres. The point in the (20 times subsampled) accumulator with the most votes is determined to be the centre of the big circle.

The four dots in a cluster are easily ordered so that the first is the farthest from the centre and the last is the nearest the centre. At this point we also eliminate clusters (if any) which do not point towards the centre.

3.2.4. Label Groups and Output Correspondences

Labelling the groups is straightforward. The integer label is simply given by,

$$\text{label} := (\text{dot}_1 = \text{big}) + 2(\text{dot}_2 = \text{big}) + 4(\text{dot}_3 = \text{big}) + 8(\text{dot}_4 = \text{big})$$

where big dots are ones that have an area more than 1.5 times one of their neighbours (the flat area ratio is 2.0) or have an area less than 1.5 times a neighbour that is known to be big. This is resolvable because there is a big-big-big-big group but no small-small-small-small group i.e. each group has at least one big dot.

Once each cluster has been labelled (1-15) we generate a table of matches between 3D mat coordinates ($z = 0$) and 2D image coordinates.

3.3. Camera Parameter Estimation

The matches obtained are then used as input to a RANSAC based camera solver module which determines the camera focal length and extrinsics.

3.4. Results

Figure 3: *Identified Features*

Figure 3 shows an example of the features detected. White squares indicate big dots and red squares indicate small dots. In this example 48 features have been found with no outliers - this is the typical performance on a very large range of examples. Note that all the dark blobs on the toy dog object have been rejected as target features by the pattern constraints.

On a test set of 131 images taken over a wide range of locations, times and lighting conditions we found only 3 images with incorrect clusters detected. These clusters were

then rejected by the RANSAC camera solving. In a further 4 images, the system failed to detect a significant number of mat features (due to poor lighting) and these images were rejected by the system. For the images with no incorrect clusters detected we obtained an accuracy of between 0.5 and 1.5 pixels (root mean square projection error). The image sizes varied from 800×600 up to 2160×1440.

4. Efficient and accurate shape from silhouettes

4.1. Background

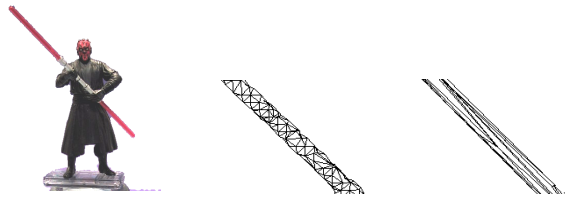
Many 3D modeling systems use the "shape from silhouette" approach to computing the shape of an object from a set of images taken from known positions [1]. The approach uses the "visual hull" approximation to the shape, which is the maximum volume that reproduces all the silhouettes of an object [8]. A good approximation to the visual hull can be obtained by intersecting the back-projection of a finite set of silhouette images. 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.

4.2. Previous work in this area

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" [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 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 [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 aliasing artifacts at sharp features on the extracted surface (see Figure 4). An additional problem with volumetric sampling methods is that they tend to generate models containing an excessive number of faces and vertices. When run with a high-resolution grid these methods are computationally expensive, requiring a large memory overhead.

Direct intersection avoids these problems by directly generating a polygonal mesh representation of the visual hull without using a regular grid [11]. This is generated by first approximating each silhouette by a polygon 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.



(a) (b) (c)

Figure 4: *Drawbacks of the volumetric sampling approach: (a) Example image, (b) Close-up of long thin structure on mesh produced by volumetric sampling has aliasing effects, (c) Close-up of mesh produced by direct intersection.*

4.3. 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 have implemented a new approach called SAVANT ("Silhouette Approximation, Vertex ANalysis and Triangulation") which takes into account all the polygon cones simultaneously.

Every vertex in the final model can only arise from the intersection of 3 of the polygon cone faces, and the intersection must lie inside all the polygon cones. Hence, we could generate the complete set of candidate vertices by back-projecting and intersecting triples of polygon edges, and only keep the candidate vertices whose projection is inside the silhouette in all of the images. However this is extremely computationally expensive (of the order n^3 in the total number of polygon edes n).

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 sections describe the steps involved in more detail.

4.4. Calculating the vertices

4.4.1. 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. We start from an intial cube that encloses the scene. We then recursively subdivide any cubes which are not totally in front of all the cameras and discard those that lie totally behind any camera until all the cubes are in front of all the cameras or a minimum size is reached.

4.4.2. Processing cubes

The region being processed consists of a set of cubes stored in a stack. 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:

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 5.

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:

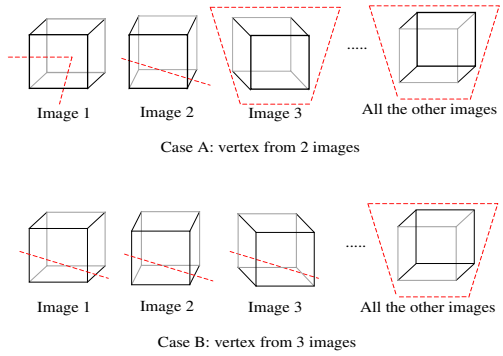


Figure 5: *Two cases that generate a vertex.*

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.

There are a number of ways of speeding up the search, for example utilising the fact that if a cube projects completely inside a silhouette then this will be true for all of it's child cubes and enumerating all possible triples of edges if the number of edges intersecting a cube is less than a threshold (see our previous work [12] for details). For efficient intersection testing the silhouette edges are stored in a quad-tree data structure.

4.5. Wiring up the vertices

The polygon faces are now obtained by using the labels associated with each hull vertex 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 until we return to 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 [12]. Finally the polygons can be triangulated.

Figure 6: *Example data sets*

4.6. Visual Hull Results

We carried out tests comparing the runtime performance of the SAVANT visual hull algorithm with an optimised incremental visual hull algorithm (detailed description [12]). 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).

A table comparing the timings for the two algorithms for each of these examples is shown in Table 1. 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.

	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	16.1s	519s	921s
SAVANT algorithm timing	**15.8s**	**102s**	**282s**

Table 1: *Comparison of algorithm performance*

A more detailed breakdown of comparative performance is given in our previous work [12]. A typical result showing how SAVANT (or direct intersection) compares to a volumetric approach is shown in Figure 4.

5. Extracting the textures

5.1. Background

In practice the measured colour and intensity for a surface element observed in different photographic images will not agree. This is due to the interaction between real world lighting effects (such as highlights) and variations in the camera gain settings as well as registration and surface modelling errors.

A common approach for blending image data for texturing is to use a triangle-based scheme [1, 13]. In general these techniques rely on a regular triangular mesh model (with a fairly uniform size and shape for each triangle). Each triangle is assigned to the "best" camera by considering the viewing angle and visibility. Blending is restricted to the boundary triangles and simple weighted averaging used to "blur" the seams. The size of the transition region across seams is crucial. If the transition region is small the seams between regions will only be slightly blurred and still visible. If the transition region is too large, this results in blurring away high frequency detail and ghosting.

An alternative to triangle based schemes is to use per-pixel weighted filtering schemes [4, 14]. For example, Pighin et al describe a system for generating 3D face models from photographs [14]. Cylindrical weight maps are constructed for each camera image that satisfy a number of requirements – visibility (zero weight for hidden surface points), smoothness (the weights should vary smoothly), positional certainty (oblique views have less weight). The images are then blended by weighted averaging to produce the final texture map. Similarly, Bernardini [4] blends albedo and normal texture data obtained from multiple structured light 2.5D scans.

We have developed a new multi-band 3D "splining" approach to preserve high-frequency detail in the transition regions of the surface textures. (The classical 2D image splining approach is described by Burt and Adelson [15]).

5.2. Texture representation

There are many possible ways to represent the texture data for an arbitrary surface. Our algorithm does not require any specific texture representation. We currently use a combination of 6 orthographic views of the object from “canonical” viewpoints (front, back etc) and a packed set of padded triangle strips to represent the triangles that were not fully visible from any canonical view. For each triangle on the mesh we have a unique triple of 2D texture coordinates.

5.3. Outline of 3D multi-band blending

The approach we take is to process each camera image sequentially. We generate a smooth weight function across the surface for each camera which we represent using an image-based representation in the camera image frame. The input

camera image data is divided into several frequency bands and each band projected into the texture map representation along with the weight function. We blend each band in the texture map representation separately using different (pixel-wise) filters. The low-frequency data is blended using a simple linear averaging filter whereas the high-frequency data is blended with a non-linear ("maximum weight") filter. Finally, the texture map bands are recombined to generate the final texture map.

The key idea here is that we wish to average the low-frequency information from all cameras (to even out lighting variation) but preserve the high-frequency information from the best view (to preserve image sharpness).

5.4. Building a camera weight function

The first step in our algorithm is to build a smooth weight function for each camera. By construction our weight function will only be non-zero for visible parts of the surface. Hence we can represent the non-zero part of the weight function using a camera image based representation. We build a greyscale weight image in the camera frame which is then applied to the surface using texture mapping. The weight image is constructed by rendering each triangle flat-filled with an intensity proportional to the texture resolution. The weight image is then blurred to ensure it is smooth. Care must be taken to ramp down the weights near occlusion edges (see our previous work [16] for details).

5.5. Blending the images

Once the camera weight function is constructed it is projected into the texture image representation by texture mapping. Additional care needs to be taken to ensure that the hidden parts of the surface are given zero weight. This is achieved by building a separate masked weight "sub-image" for each partially visible triangle.

The low-frequency camera image is generated by bluring the input camera image. The size of the blur kernel can be determined by mapping a threshold in world units (e.g. 5% of object size) into camera image units. The high-frequency image is a signed image obtained by subtracting the low-frequency image from the original image. These images are now projected into the texture image frame using standard texture mapping.

A low-frequency texture image is maintained using a weighted average filter to blend the projected low-band images. Similarly the high-frequency signed texture image is updated using a "maximum weight" filter (for each pixel keep the signed colour value with the maximum associated weight).

Once all the input images are processed the low and high-frequency texture images are recombined by simple addition. The method can be summarised by the following equation:

$$J(u,v) = \frac{\sum_k \lambda^k(u,v) \mathrm{I}^k_{\mathrm{low}}(u,v)}{\sum_k \lambda^k(u,v)} + \sum_k \mu^k(u,v) \mathrm{I}^k_{\mathrm{high}}(u,v)$$

where $J(u,v)$ is the output texture map image, $\mathrm{I}^k_{\mathrm{low}}$ is the k'th low frequency image $(G_\sigma \cdot \mathrm{I}^k)$ projected into the texture domain, $\mathrm{I}^k_{\mathrm{high}}$ is the high frequency image $(\mathrm{I}^k - G_\sigma \cdot \mathrm{I}^k)$ projected into the texture domain and $\lambda^k(u,v)$ is the projected smooth weight function for the k'th input camera image. The high frequency filter weights μ_k are defined by:

$$\mu_k = \begin{cases} 1 & \text{if } \lambda^k > \lambda^i \ \forall i \neq k \\ 0 & \text{otherwise} \end{cases}$$

The final texture image can then be used to render the textured surface model from any viewpoint.

5.6. Results of texture blending

In order to demonstrate our algorithm we used a "doll" test sequence of 16 images taken of a china doll model. The visual hull model generated contained 4,000 triangles. We also used a further "dino" test sequence of a toy dinosaur with 25 camera images and 10,000 triangles in the mesh model. We compared the multiband technqiue with two simpler techniques that utilise the same image-based weight maps - **Averaging** and **Best camera**. For "averaging", a single band is used and the texture data combined using weighted averaging. For the "best camera" scheme, the camera pixel with the highest associated weight is used for each texture map pixel.

(a) (b) (c) (d)

Figure 7: *Novel views of textured models: (a) "best camera" scheme for doll has noticeable seams, (b) multiband scheme for doll is sharp and seamless, (c) "averaging" scheme for dino is blurred, (d) multiband scheme for dino is sharp and seamless.*

Figure 7 shows the textured mesh for the "china doll" and "dino" input sequences. From these images it is clear that the "averaging" approach can suffer from blurring due to camera misregistration and shape modeling errors. The "best camera" method gives crisper textures but can suffer from seams due to lighting inconsistencies between input camera images. However, our multiband technique gives a good compromise between these two extremes. Typically the time taken is 2-3 minutes on a 800MhZ PC.

6. Results

Although there are obvious limitations with silhouette based surface reconstruction (e.g. modelling concavities) we have found that the synthesised views of the texture mapped model are suprisingly convincing. Figure 8 shows some typical examples of novel views obtained in 3D S.O.M. next to a typical input camera image. More example models are available from the 3D S.O.M. web site (http://www.cre.canon.co.uk/3dsom).

7. Conclusions

We have described a software solution to 3D scanning from photos. Our approach has been to make minimal assumptions about the object and scene lighting by introducing a robust calibration object and utilising an efficient "shape from silhouettes" technique to improve robustness and performance. The key novel contributions of this work are - robust camera estimation using a carefully designed calibration target, batch visual hull calculation that extends the silhouette approach to large numbers of complex silhouettes and multi-band texture blending to ensure consistent looking textured models without loss of detail. Future work will look at extending the system to handle concavities and improvements to model quality.

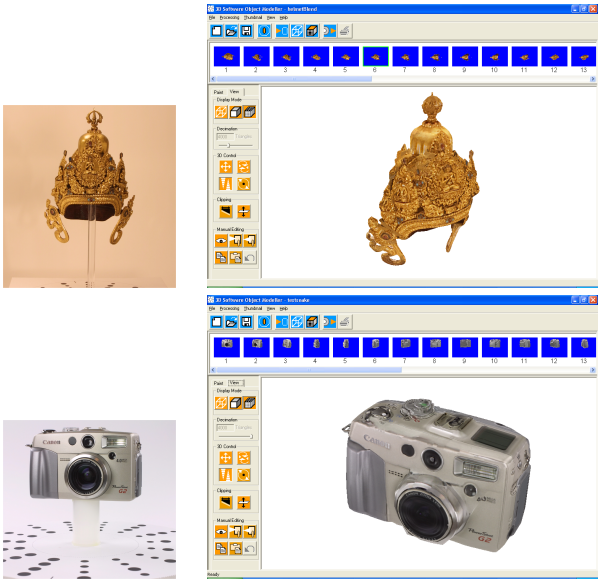


Figure 8: *3D models generated next to example input images*

8. Acknowledgements

The authors wish to thank Simon Rowe and Qi He Hong for useful discussions and comments and Canon Inc. for funding the work.

References

1. W. Niem. Automatic reconstruction of 3d objects using a mobile camera. *Image and Vision Computing*, 17(2):125–134, February 1999.

2. R. Koch, M. Pollefeys, and L.J. Van Gool. Multi viewpoint stereo from uncalibrated video sequences. In *ECCV98*, pages 55–71, 1998.

3. S.M. Seitz and C.R. Dyer. Photorealistic scene reconstruction by voxel coloring. In *CVPR97*, pages 1067–1073, 1997.

4. F. Bernardini, I. Martin, and H. Rushmeier. High quality texture reconstruction from multiple scans. *IEEE Trans. on Visualization and Computer Graphics*, 7(4), Oct-Dec 2001.

5. W. Matusik, H. Pfister, A. Ngan, P. Beardsley, R. Ziegler, and L. McMillan. Image-based 3d photography using opacity hulls. *ACM Transactions on Graphics*, 21(3):427–437, July 2002.

6. A.R. Smith and J.F. Blinn. Blue screen matting. *Computer Graphics*, 30(Annual Conference Series):259–268, 1996.

7. S.J. Gortler, R. Grzeszczuk, R. Szeliski, and M.F. Cohen. The lumigraph. In *SIGGRAPH 96 Conference Proceedings*, pages 43–54. ACM SIGGRAPH, August 1996.

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

9. Richard Szeliski. Rapid octree construction from image sequences. *CVGIP: Image Understanding*, 58(1):23–32, July 1993.

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

11. W. Matusik, C. Buehler, and L. McMillan. Polyhedral visual hulls for real-time rendering. In *Proceedings of the 12th Eurographics Workshop on Rendering*, pages 115–125, London, England, June 2001.

12. A. Lyons, A. Baumberg, and A. Kotcheff. Savant: A new efficient approach to generating the visual hull. In *Short Presentations Proceedings of Eurographics*, pages 18 – 28, 2002.

13. H. Lensch, W. Heidrich, and H. Seidel. A silhouette-based algorithm for texture registration and stitching. *Graphical Models*, 63(4):245–262.

14. F. Pighin, J. Hecker, D. Lischinski, R. Szeliski, and D. Salesin. Synthesizing realistic facial expressions from photographs. In *SIGGRAPH 98 Conference Proceedings*.

15. P. J. Burt and E. H. Adelson. A multiresolution spline with application to image mosaics. *ACM Transactions on Graphics*, 2(4):217–236, October 1983.

16. A. Baumberg. Blending images for texturing 3d models. In *British Machine Vision Conference (BMVC)*, pages 404–413, 2002.