Marching Intersections: An Efficient Approach to Shape-from-Silhouette

M. Tarini, M. Callieri, C. Montani, C. Rocchini
Istituto di Scienza e Tecnologie dell'Informazione
Consiglio Nazionale delle Ricerche
Via G. Moruzzi 1, Pisa - Italy
$tarini@di.unipi.it, \{callieri \mid montani \mid rocchini\}@iei.pi.cnr.it$
K. Olsson, T. Persson
Media Technology and Engineering
University of Linköping - Sweden
$\{karol033 \mid thepe562\}@student.liu.se$

Abstract

A new shape-from-silhouette algorithm for the creation of 3D digital models is presented. The algorithm is based on the use of the *Marching Intersection (MI)* data structure, a volumetric scheme which allows efficient representation of 3D polyhedra and reduces the boolean operations between them to simple boolean operations on linear intervals. MI supports the definition of a direct shape-from-silhouette approach: the 3D conoids built from the silhouettes extracted from the images of the object are directly intersected to form the resulting 3D digital model. Compared to existing methods, our approach allows high quality models to be obtained in an efficient way. Examples on synthetic objects together with quantitative and qualitative evaluations are given.

1 Introduction

The acquisition of natural looking 3D digital models from real objects is becoming an inalienable issue in a variety of 3D multimedia applications like 3D tele-shopping, virtual studio production, 3D teleconferencing, 3D information systems, 3D archiving, etc..

Between the many different techniques and tools for shape acquisition [2], the optical systems, that are based on the acquisition of images of the object to be scanned, are particularly developed and differentiated. Optical systems can be classified in two large categories: active systems, in which the scanning process takes advantage of some kind of structured light cast over the object, and passive systems, where the shape information is obtained merely from the analysis of the images of the object.

Shape-from-silhouette is certainly among the cheapest and more robust passive methods. The earliest work based on this approach is by Martin and Aggarwal [7]. The needed instrumentation is limited to a digital camera [12], with the possible integration of an economical turntable (Szeliski [23] uses a spring-loaded microwave turntable for his shape-from-silhouette scanner). However, the shape-from-silhouette algorithms are computationally expensive and generally produce low quality results.

In this paper we present a new approach to shape-from-silhouette based on the use of a new data structure and algorithms: the *Marching Intersections* [19]. The new method improves the efficiency of the shape-from-silhouette solutions and the quality of the generated models.

2 Shape-from-Silhouette basics and related work

The basic idea of the shape-from-silhouette algorithms (also known as occluding contours methods) is very simple. Let us suppose to have multiple views V of the 3D object to be scanned. From each view v we extract the silhouette s_v, which is the region including the object's interior pixels and delimited by the line(s) separating the object from the

background. Generally s_v is not convex and can present holes due to the geometry of the object. For each s_v we generate a cone-like volume c_v (called *extended silhouette* or *truncated extended silhouette* in the case we consider the portion of the conoid limited by a couple of near and far planes chosen by the user), defined by all the rays starting at the point of view and passing through all the points of the silhouette on the image plane. The 3D object is definitely internal to c_v and this is true for every view $v' \in V$; it follows that the object is contained into the volume $c_V = \cap_{v \in V} c_v$. As the size of V goes to infinity, and includes all possible views, c_V converges to a shape known as the *visual hull* vh of the original geometry.

The visual hull [5] of an object rarely coincides with the object itself; in general we can only state that the following relationship holds:

$$obj \subseteq vh_{obj} \subseteq ch_{obj} \quad (1)$$

where vh_{obj} and ch_{obj} represent the visual hull and the convex hull of the object obj, respectively.

Even though the visual hull, and therefore the digital model obtained from a limited number of views c_V, is a superset of the 3D object under examination, in practical cases and for many applications the geometric model c_V is considered satisfactory. The level of satisfaction obviously depends on the kind of object and on the number and position of the acquired views. The generated model can be sensibly improved from the appearance point of view by means of color textures obtained by the original images [13] or it can be refined from a geometric point of view by means of further (less robust) techniques like, for example, shape-from-shading [25], etc.

In this paper we propose an efficient and precise method to generate the digital model c_V of a real 3D object. We will refer to c_V as the *inferred visual hull* [21] of the object.

The main problem in the computation of the inferred visual hull is the difficulty in designing a robust and efficient algorithm for the intersection of the extended silhouettes; due to these difficulties the few direct methods proposed present strong usage limitations [9]; the other existing methods range from image based to volume carving approaches.

Matusik *et al.* [8] propose an image based solution which closely resembles the computation of the intersection between CSG models by means of a ray casting algorithm; however, the method does not generate a geometric model of the inferred visual hull but it produces new views starting from multiple reference views.

On the other hand, most of the authors adopted a volume carving approach: once a volume space which surely contains the object has been defined [11], a regular grid is adopted; the spatial resolution of the grid is selected by the user and generally represents a compromise between memory needs and execution times with respect to output precision and quality. Volume carving is simple: if a voxel of the grid is internal to all the cone-like extended silhouettes c_v, then it is certainly internal to the inferred visual hull c_V. Each voxel overcoming the membership tests is set to the in value; the assigned value is out as soon as a test fails. The surface of the inferred visual hull, that is the surface separating in and out values in the grid, can be generated, for example, by means of a Marching Cubes [6] (MC for short) technique.

Volume carving methods differ for three main aspects:

- reduction of the number of membership tests by means of hierarchical design of the volume grid. Niem [14, 12] adopts the pillars, a stack of voxels extending from the first to the last horizontal slice of the grid, as base information unit; Hong and Shneier [4], Potmesil [17], Noborio *et al.* [16], Srivastava and Ahuja [22], and Szeliski [23] use the hierarchical octree data structure;
- reference space in which the membership test is carried out, i.e. the 2D space of the images ([14, 12, 4, 17, 23]) or the 3D space of the extended silhouettes ([16, 22]);
- accuracy of the membership test.

We think that many of the mentioned limitations are due to the adoption of an inverse approach based on volume carving.

Independently of the *geometric* characteristics of the used shape-from-silhouettes approach, all the methods are based on the acquisition of multiple images and on the extraction of the silhouette of the object from each of these images. It follows that important aspects of the chosen method refer to the image acquisition environment (a single digital camera [12] which changes position at each shutter release or a digital camera used in conjunction with a turntable [3, 11, 14, 23]), to the camera model adopted, to the camera calibration procedure (the

method used to identify the interior and exterior parameters of the camera like, for example, the focal distance, position, orientation [1, 24]), and to the silhouette detection algorithm.

We will not describe these aspects here because they do not represent the innovative part of the paper and we are mainly interested in demonstrating the accuracy and the efficiency of our method. Moreover, the examples reported do not derive from real objects. We simply skipped the acquisition phase of the method and we generated the starting images as perspective views of digital models. The image acquisition environment is supposed to have a digital camera in fixed position and the object standing on a turn-table with known rotation angles between subsequent images.

3 The Marching Intersections Representation Scheme

The *Marching Intersections (MI)* [19] is a representation scheme for 3D surfaces and polyhedra based on the use of a virtual 3D uniform grid; the resolution of the grid is selected by the user depending on the application needs. MI is suitable for the resampling of surfaces, the removal of high frequency details, the topological and geometric simplification of huge 3D meshes, for the fusion of multiple range maps acquired by means of 3D range scanners [20]. Moreover, MI turns out to be really efficient for the applications in which scheme conversions and boolean operations between free-form complex 3D polyhedra are required. These last characteristic represent the basis for the computation of the inferred visual hull as proposed in this paper.

A 2D example of the data structure and the different steps of the conversion algorithm is shown in Fig. 1. Given the curve to be processed, the user chooses the reference grid which meets her/his approximation requirements (top-left); all the intersections of the input curve with the horizontal and vertical lines of the grid are stored in proper data structures (conversion step) and these structures actually represent the MI representation scheme.

Similarly, in the 3D case, the MI structure consist in three orthogonal set of *rays*, parallel to the X, Y or Z axis. Each ray represent a line of the 3D grid, and is recorded as a list of its *intersections* with the object represented by the MI structure. Each intersection is represented in turn by a scalar value, storing the position of that interception along the ray. In the original implementation of MI that value was coupled by the "sign" of the observation (if we suppose to walk along the ray, a "+" sign means that we are entering the object, a "-" means that we are leaving it). In our case, we deal only with closed objects (both truncated extended silhouettes and visual hulls are), so we do not need to store the sign explicitly: every odd intersection is a "+" and every even one is a "-".

Rays along the X, Y or Z direction are arranged in 2D regular arrays, called the *X-rayset*, *Y-rayset*, and *Z-rayset* respectively. For efficiency reasons, we want the origin of all the rays to lay on integer values. For example, the *ray* at position (i, j) of the *Z-rayset* refers to the line parallel to the Z axis and passing through the point $[i, j, 0]$ (an intersection value k in its list would represent the point $[i, j, k]$). The surface represented by the MI is scaled accordingly before being immersed in the grid.

3.1 Using the MI structure for shape from silhouette

For each silhouette, we first get a MI structure representing the corresponding truncated conoid. Next Sections 4 describes how this operation is done.

Intersection of conoids (as MI structures) is then computed performing an AND operation. In a MI scheme, boolean operations can be easily performed and they boil down to simple ray-by-ray operations on the linear intervals delimited by couple of intersections (in ray's intersections list). Actually, to save memory, rather than computing several MI representations and then intersecting them, we keep a single MI structure, initialized as the conoid for the first silhouette and updated as we process each subsequent silhouette.

As in most shape from silhouette approaches, we consider the object as fixed and the camera as moving around it (even if in typical physical settings is the other way round, with a fixed camera and the object rotating). This way conoids computed from each silhouette will be expressed as MI structures sharing the same position and axis orientation, which is a necessary condition to intersect them.

Finally, once we obtain a MI data structure representing the intersection of all conoids, we want to extract a triangle mesh out of it. This is done by means of a time efficient technique [19] which tra-

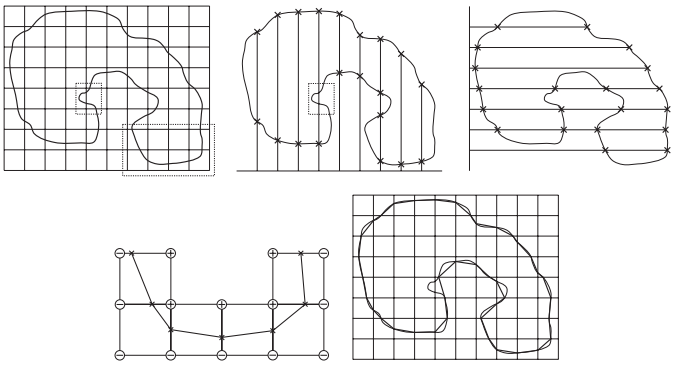

Figure 1: The *MI* data structure and conversion algorithm in a 2D example: (top-left) the curve to be processed and the MI data structures collecting the intersections between the input curve and the vertical (top-center) and horizontal (top-right) lines of the user selected grid; (bottom-left) based on the horizontal and vertical intersections, an MC look up table entry code is located for each not empty (virtual) cell; (bottom-right) the reconstructed curve. The vertical intersections inside the small dotted box are collected and then deleted from the algorithm because belonging to the same virtual cell. This removal is a prerequisite for the correct operation of the algorithm and it leads to the removal of high frequency details.

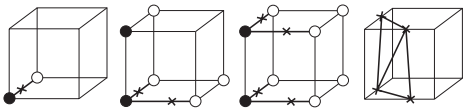

Figure 2: Reconstruction of a virtual cell and of the corresponding triangular patch performed by the *MI* method, starting from a signed intersection on an edge.

verses the "virtual cells" implicitly defined by the MI structure (see Fig. 2), building a proper MC entry for them (assigning in/out status to grid vertices) that in turn is then used to index a MC's lookup table [10].

Note that the mesh extraction phase is required only to find mesh connectivity: in fact, since each intersection along any ray define a 3D point, a MI data structure already contains all the vertices of the mesh. This is an advantage over standard MC approaches, where intersections are found with approximations, for example by interpolation of voxel values.

4 Representing a polyhedron in MI

In order to represent a given polyhedron in a MI structure, we need to perform a conversion step consisting in the detection of all the intersections between that polyhedron and the reference grid.

In the typical, most general use of MI [19], the input polyhedron is a general mesh, and its conversion occurs on a per-face basis. Each face is scan converted three times, once per MI ray-set (X, Y, and Z ones), finding and storing all its intersections with rays in the respective ray-set. Once this is done for each face, the X Y and Z ray-sets will contain all the intersections between the mesh and the grid lines.

At the end of the scan conversion process, in each ray the intersection list is sorted with respect to the intersection value. Sorting ensures fast detection of nearby intersections and fast search for specific intervals. The only geometric operation the intersections undergo is the *removal*: each pair of consecutive intersections $i1$ and $i2$ which lie on the same cell edge (that is, $\lfloor ic_{i1} \rfloor = \lfloor ic_{i2} \rfloor$) and have discordant signs is removed from the structure it belongs to. As shown in Fig. 3, this operation corresponds to a resampling which implies the removal of high frequency details and has the effect to get

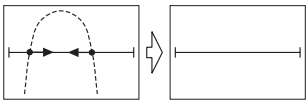

Figure 3: An example of the *Removal* of two discordant intersections belonging to the same virtual cell.

```
1 for each ray-set (X,Y,Z)
2  for each ray (i,j) in it
3   compute the projection on I of ray(i,j)
4   scan-convert the corresponding 2D line:
5   for each intersection k with silhouette found
6     remove perspective distortion, obtaining k'
7     add k' to ray(i,j) of the current ray-set
```

Figure 4: A pseudo code for the conversion into a MI structure of a truncated conoid implicitly defined by a 2D silhouette on an image **I** and by a camera position.

an arrangement of the intersections data structures in a MC-*compliant* manner, i.e. in such a way that a MC-like surface reconstruction algorithm could be successfully applied.

In our case the input surface is the one of a truncated extended silhouette. A possible approach would be to compute a mesh representing the truncated extended silhouette (composed by a front face, a back face[1] and a set of trapezoids connecting each edge of the former with one of the latter), and then convert it as above. Still, a shortcut is possible here.

4.1 Conversion of a truncated extended silhouette

The MI data structure proves ideal for representing extended silhouettes. In fact it turns out to be easy to scan convert an extended silhouette surface directly into the X, Y and Z ray-set structures.

Having a silhouette, a direct way to compute the MI structure representing its 3D extension is shown with the algorithm in Fig. 4. We will now describe some of its phases in more detail.

Step 3 requires the knowledge of camera internal parameters and its position (that is, the camera matrix to project 3D points into the image). In

[1] Apparently the frontal front and back faces are not needed because they will not contribute to the final surface, but in order to define the AND operator the volume need to be closed.

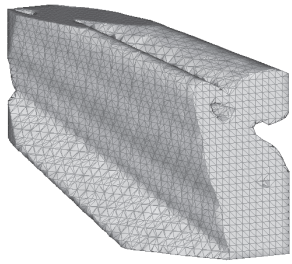

Figure 5: Surface of a conoid extracted from a MI structure obtained with algorithm of Fig. 4.

real world implementations these parameters can be found with calibration methods such as [1, 24]. At the end of Step 3 we have the 2D line that is the projection on the image of the currently processed ray of the MI.

In Step 4 we scan-convert that line to find its intersections with the 2D silhouette (see Fig. 6). This is a purely 2D operation: the actual implementation depends on the structure used for representing the 2D silhouette. In our case, we keep the silhouette as a 2D array of bits, 1 for inside and 0 for outside, and to find intersections we traverse the line with a Bresenham-like line scan algorithm. Such a 2D silhouette representation is a very natural one, and can be obtained from real pictures thresholding a 2D array of RGB distances computed between a background picture without the object and one with the object. Scan converting the line in the proper direction, we get the positive side effect of producing intersections already in the right order, so that we can avoid to sort intercepts, as the MI algorithm would usually require.

In Step 6 we "unproject" the 2D intersections found, that is, we compute the position k' on the current (3D) ray of the intersection between that ray with the conoid, starting from the value k which is the x (or y) coordinate, on the image, of the intersection between the 2D projected line and the silhouette. Basic projective geometry gives:

$$k' = -\frac{Ray_s \cdot (Row_x - k \cdot Row_4)}{Ray_d \cdot (Row_x - k \cdot Row_4)}$$

Where Row_x and Row_4 are the 1st and the 4th row of the view projection matrix, Ray_s is starting point of current ray with one as 4th component, and Ray_d is its direction, with zero as 4th component.

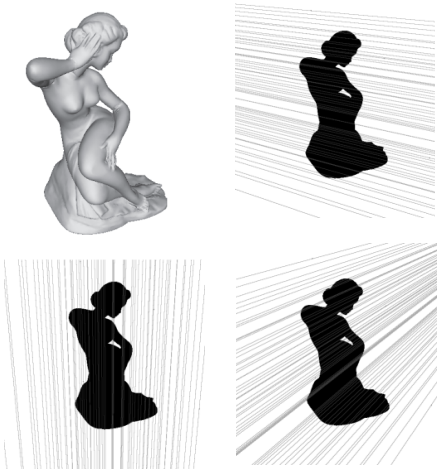

Figure 6: One object (top left) and its silhouette with 2D lines traced over it to find intersections along rays in the X, Y and Z ray-set of the MI, respectively. The number of lines has been reduced for illustration purposes: in typical cases they would cover most of the area of the image.

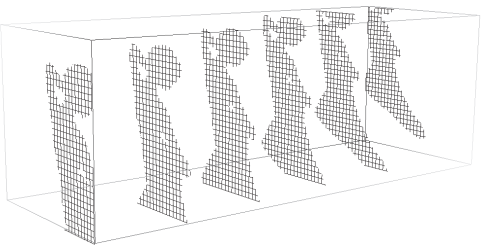

Figure 7: Some slices of the conoid visible in Fig. 5, composed by intervals defined by some of the MI ray along X and Y. Notice that each slice is the replication of another one at a different scale. Cropping is applied on the sides of the volume to make the front and back faces.

For example, for ray (i, j) of the Y ray-set, Ray_s would be $[i, 0, j, 1]$ and Ray_d would be $[0, 1, 0, 0]$. If k represent the y coordinate, then Row_y (the 2nd row of the view projection matrix) should be used instead of Row_x.

When z' is computed, we crop it in $[0..N]$ (N being the size of the volume represented by the MI structure). Cropped intersections will nicely form the front and back boundary of the truncated conoid (see Fig.5 for an example).

The algorithm of Fig.4 is repeated for each silhouette. For the 1st processed silhouette, in Step 7 we build the initial MI structure by storing int its proper ray the unprojected intersections. For subsequent silhouettes, Step 7 consists in merging the (unprojected) intersections found along the line with the ones currently present in the same (i, j) ray of the MI structure, by computing the boolean AND of the two interval sets.

The AND operation between intervals can only reduce them. So if the current ray is empty (no intersections along it from previously processed conoids), then we can skip it in Step 2. For this reason, given a set of silhouettes taken from all around an object (like when a fixed camera is in front of a rotating plate), it is convenient to process 2 orthogonal views first, so that their intersection will be small, and more rays will be empty and skipped to speed up subsequent views.

The *removal* operation (see Section 4) can be done on the fly while storing the interceptions during Step 7.

The asymptotic complexity of the algorithm in Fig. 4 iterated over all silhouettes is $O(KMN^2)$ where N is the grid size of the MI data structure (which therefore represents a volume of $N \times N \times N$), M is the number of pictures, and K is a measure of the complexity of the silhouette, recording the average number of operations needed to find intersection of the silhouette along a given $2D$ direction. (The value of K is influenced by many factors like which data structure is used to represent the silhouette. In our case, the cost of finding intersections is proportional to the resolution of the image.)

4.2 Optimization by caching and reusing scan-converted lines

For each processed silhouette, use of the MI data structure allows to manage a N^3 volume in a N^2 time. Still, an important optimization can be done.

The basic observation is that, slicing a conoid with planes parallel to the XY plane (or YX, or ZY), the sections present the same 2D shape, up to a resizing and a translation (see Fig. 7).

In practice this means that, in our algorithm, we

Size	Hits	Time0	Time1
Lady silhouettes			
64	12.3%	0.692	0.070
128	44.0%	1.805	0.310
256	80.2%	2.643	1.542
512	94.6%	3.190	8.021
Bunny silhouettes			
64	15.3%	1.350	0.091
128	52.8%	3.079	0.511
256	85.1%	4.049	2.604
512	96.0%	4.895	13.189

Figure 8: Some results of our application running on a normal PC (Athlon 1.4 GHz 512 MB). *Size* is the grid size, *Hits* is the percentage of cache hits, that is, how many out of the $3Size^2$ rays composing the structure were found in the cache described in Sec. 4.2 and were not scan-converted. *Time0* is the time in seconds required to convert a silhouette into a MI structure for the conoid, and to intersect it with the previous MI structure. *Time1* is the time in seconds to extract the final mesh from the MI structure. Total time to compute the models shown in Fig. 9 is ((*Time0* $\times$ number of silhouettes) + *Time1*), which is 340 sec. for the Lady and 577 sec. for the Bunny.

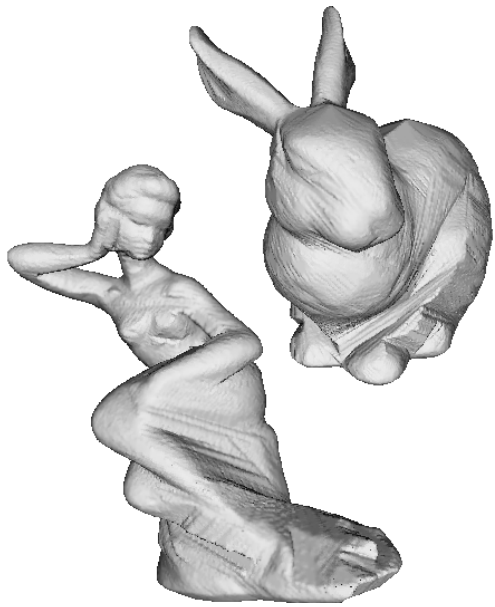

Figure 9: Examples of reconstruction results: the Lady, obtained by silhouettes like the one in Fig. 6, and the familiar Stanford bunny. In both cases a 256x256x256 MI structure has been carved by 128 (synthetic) 1000x1000 silhouettes images taken from a (virtual) camera rotating around it (like when a rotating plate is used). The two meshes are composed by 257K and 451K faces respectively.

can conveniently cache the results of phases 4 to 6; in each ray-set of the MI all rays are parallel, so their 2D projections on the image (found in Steps 3) belong to lines all passing from a single point (the vanishing point of that set of rays). We use a cache consisting of a 1D array of lists of computed intersections found along one of those 2D lines (for best efficiency, in each entry of the cache we store intersection values *after* back projection). Cached results are reused when a different ray of the same ray-set is used, if that ray projects over the same 2D line (up to a fraction of a pixel). In that case we just displace and scale the all previously computed intersection in the list. Especially for high values of N this happens very often. For a $256 \times 256 \times 256$ MI and 1000×1000 images, more than 80% of the times a cached line is used (see Table 8): only just more than a thousands 2D lines are scan converted and reused for 64K 3D rays. This approach makes the system very scalable: as Fig. 8 shows, computation times tend to grow less than linearly (rather than cubically or quadratically) with the size N of the processed volume.

As an additional optimization, when a ray is skipped in Step 2 because it is already empty, we can tag the corresponding line in cache as empty.

5 Results and conclusions

Some qualitative and visual results are shown in in table of Fig. 8 and in Fig. 9.

To conclude, we designed and implemented a volumetric based visual hull extraction technique that uses the powerfully compact structure of Marching Intersection. That proved ideal for the task. The main benefits are twofold:

- first, efficiency in term of memory space (quadratic rather than cubic with the adopted resolution) and computation times (efficient conversion, intersections and mesh extraction) gives the possibility both to use a higher volume resolution, and to merge more extended

silhouettes, each at higher resolution.

- secondly, explicit handling of intercept points (rather than their evaluation by interpolation of voxel values a-la MC) improves, for a given volume resolution, the quality of the results by avoiding part of the grid aliasing.

As a future work direction, it would be interesting to investigate whether and in which way the MI structure could be useful in the open problem of finely tuning the camera positions and parameters for a given set of uncalibrated or roughly calibrated images featuring silhouettes (as in [18]). That would remove, or greatly reduce, the effect of this source of error, which we ignored in this paper (see [15] for an estimation of its magnitude).

Acknowledgements

We acknowledge the financial support of the European Project ViHAP3D (IST-2001-32641)

References

[1] Intel Corporation. Reference manual for the open source computer vision library. Technical report, http://www.intel.com/research/mrl/research/opencv/, 2000.

[2] B. Curless and S. Seitz. 3D Photography. In *ACM Siggraph '00 Course Notes, Course No. 19*, August 24th 2000.

[3] A. W. Fitzgibbon, G. Cross, and A. Zisserman. Automatic 3D model construction for turn-table sequences. *Lecture Notes in Computer Science*, 1506:155–170, 1998.

[4] T. H. Hong and M. O. Shneier. Describing a robot'S workspace using a sequence of views from a moving camera. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7:721–726, 1985.

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

[6] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *ACM Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 163–170, 1987.

[7] W. N. Martin and J. K. Aggarwal. Volumetric Descriptions of Objects From Multiple Views. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-5(2):150–158, 1983.

[8] W. Matusik, C. Buehler, R. Raskar, S. J. Gortler, and L. McMillan. Image-based visual hulls. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 369–374. ACM Press / ACM SIGGRAPH, 2000.

[9] Wojciech Matusik, Chris Buehler, and Leonard McMillan. Polyhedral visual hulls for Real-Time rendering. In *Proceedings of Eurographics Workshop on Rendering*, pages 115–126, London, 2001.

[10] C. Montani, R. Scateni, and R. Scopigno. A modified look-up table for implicit disambiguation of Marching Cubes. *The Visual Computer*, 10(6):353–355, 1994.

[11] A. Y. Mülayim, V. Atalay, O. Özün, and F.Schmitt. On the silhouette based 3d reconstruction and initial bounding cube estimation. In *5th International Fall Workshop on Vision Modelling and Visualization - VMV*, 2000. Max-Planck-Institut fur Informatik, Saarbrücken, Germany, Nov., 22-24.

[12] W. Niem. Automatic reconstruction of 3d objects using a mobile camera. *Image and Vision Computing*, 17:125–134, 1999.

[13] W. Niem and H. Broszio. Mapping texture from multiple camera views onto 3d-object models for computer animation. In *International Workshop on Stereoscopic and Three Dimensional Imaging Conf. Proc.*, 1995. Santorini, Greece.

[14] W. Niem and R. Buschmann. Automatic modelling of 3d natural objects from multiple views. In Yakup Paker and Sylvia Wilbur, editors, *Image Processing for Broadcast and Video Production*. Workshops in Computing Series, Springer, Hamburg 1994, 1994.

[15] Wolfgang Niem. Error analysis for silhouette-based 3d shape estimation from multiple views. In *International Workshop on Syntetic-Natural Hybrid Coding and 3D Imaging (IWSNHC3DI'97)*, Rhodes, Greece, 1997.

[16] H. Noborio, S. Fukuda, and S. Arimoto. Construction of the octree approximating three-dimensional objects by using multiple views. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-10(6):769–782, 1989.

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

[18] P. Ramamanathan, E. Steinbach, and B Girod. Silhouette-based multiple-view camara calibration. In *Proc. of Vision, Modeling and Visualization*, pages 2–10, Saarbrüken, 2000.

[19] C. Rocchini, P. Cignoni, F. Ganovelli, C. Montani, P. Pingi, and R. Scopigno. *Marching Intersections:* an efficient resampling algorithm for surface manipulation. In *Proceedings of the International Conference on Shape Modeling and Applications - SMI 2001*, pages 296–305. IEEE Computer Society Press, 2001. Genova, Italy, 7-11 May 2001.

[20] C. Rocchini, P. Cignoni, C. Montani, and R. Scopigno. The *Marching Intersections* algorithm for merging range images. ISTI-CNR, Pisa, Italy, Submitted for publication, March 2001.

[21] G. Slabaugh, B. Culbertson, T. Malzbender, and R. Schafer. A survey of methods for volumetric scene reconstruction from photographs. In K. Mueller and A. Kaufmann, editors, *Proceedings of the Joint IEEE TCVG and Euro graphics Workshop (VolumeGraphics-01)*, pages 81–100, Wien, June 21–22 2001. Springer-Verlag.

[22] S. K. Srivastava and N. Ahuja. Octree generation from object silhouettes in perspective views. *Computer Vision, Graphics, and Image Processing*, 49(1):68–84, 1990.

[23] Richard Szeliski. Rapid octree construction from image sequences. *Computer Vision, Graphics, and Image Processing. Image Understanding*, 58(1):23–32, 1993.

[24] R. Tsai. A versatile camera calibration technique for high accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses. *IEEE Journal of Robotics and Automation*, RA-3(4), August 1987. a preliminary version appeared in Proc. 1986 IEEE Int. Conf. Computer Vision and Pattern Recognition, Miami, FL, June 22-26, 1986.

[25] R. Zhang, P.-S. Tsai, J. E. Cryer, and M. Shah. Shape from shading: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(8):690–706, 1999.