Shape from Contours and Multiple Stereo – A Hierarchical, Mesh-Based Approach

Hendrik Kück and Wolfgang Heidrich
Department of Computer Science
University of British Columbia

Christian Vogelgsang
Computer Graphics Group
Universität Erlangen-Nürnberg

Abstract

We present a novel method for 3D shape recovery based on a combination of visual hull information and multi image stereo. We start from a coarse triangle mesh extracted from visual hull information. The mesh is then hierarchically refined and its vertex positions are optimized based on multi image stereo information. This optimization procedure utilizes 3D graphics hardware to evaluate the quality of vertex positions, and takes both color consistency, and occlusion effects as well as silhouette information into account.

By directly working on a triangle mesh we are able to obtain more spatial coherence than algorithms based entirely on point information, such as voxel-based methods. This allows us to deal with objects that have very little structure in some places, as well as small specular patches.

Keywords: image based modeling, surface reconstruction, multi view stereo, triangle meshes, visual hull, mesh deformation, hardware accelerated

1 Introduction

Visual hull algorithms (e.g. [11, 13]) provide a fast way of extracting approximate shape information from multiple images. Under the assumption of Lambertian surfaces, these approximate shapes can be refined with voxel coloring [18] and other space carving methods [10, 19] that make use of color consistency of points across the images. Unfortunately, these voxel-based methods have problems in the presence of larger surface areas with little or no structure or even small specular patches. Noisy image data also presents a challenge. These limitations are due to the use of a very local color consistency test that does not make use of neighborhood information.

To address these issues, researchers have looked into cross-correlation of color values in some local neighborhood representing a continuous approximation of the target shape. The work in this area can roughly be grouped into two categories: methods based on triangle meshes, and methods based on level sets. Both approaches have their distinct advantages and disadvantages: level set methods [4, 20] easily handle topology changes and thereby avoid self intersections. Mesh-based approaches, on the other hand, are more suitable for adaptive refinement. They start with an initial mesh, providing a rough approximation of the general shape of the object, such as a planar grid [5], a sphere [21], or the visual hull [2, 14, 12, 6, 3]. The mesh is then deformed as to maximize the consistency of the shape with the given images. Because vertex positions in meshes are not confined to a regular grid, the mesh-based methods can handle thin structures such as sheets of cloth (see Figure 5 for an example) more easily than approaches based on voxels or level sets.

In this paper, we describe a hierarchical, mesh-based algorithm for extracting geometry from a combination of silhouette and stereo information from calibrated cameras. We first extract a coarse triangular mesh for the visual hull from silhouette information. To this end, we develop a new implicit formulation of the visual hull that we subsequently convert into a triangle mesh (Section 2). We then optimize the position of all vertices in the mesh taking stereo information into account. For this step we develop a cost function for the location of every vertex that is based on color consistency on the fan of triangles surrounding that vertex. In addition to color consistency, we take visibility and mesh quality (both smoothness and triangle shape) into account, and prevent interpenetrations. We also make sure that the optimization process preserves the silhouettes in the images by preventing the mesh from growing beyond the silhouette, but also from shrinking away from it. The details of the cost function are presented in Section 3. Based on this cost function, we then perform a hierarchical optimization, where we recursively subdivide the mesh once the positions for the coarser mesh resolution start to converge (Section 4). Finally, to make all this feasible,

we utilize 3D graphics accelerators for helping with the evaluation of the cost function (Section 5).

By employing this triangle-based approach, we can obtain good geometry even for objects that have virtually no structure in some areas. In contrast to voxel-based approaches, a small number of features or a slight gradient within the triangle fan around a vertex is sufficient to optimize the position of that vertex (although additional features do help in practice). Because we also employ a hierarchical approach, we can still represent fine details in areas where enough structure is available. In contrast to other mesh-based algorithms, our approach combines the correct treatment of occlusions, the use of silhouette information *in the optimization stage*, prevention of self-intersections, and hierarchical subdivision. While most other methods evaluate color consistency and visibility only at points during the mesh optimization (or even ignore the visibility), we integrate over a local region of the surface, yielding a more robust optimization process with fewer local minima.

The inclusion of the cost term that forces the mesh to maintain the original silhouette proves essential for dealing with objects that have large areas of very little structure, since it counters the tendency of the smoothing term (similar to curvature flow in level set methods) to excessively shrink the mesh in these areas.

2 The Visual Hull

As the starting point for our mesh-based optimization method we use a triangle mesh generated from silhouette information. There are several approaches available that generate a triangle mesh for the visual hull directly by intersecting polygonal silhouette cones (e.g. [13]). Unfortunately, the meshes produced by these algorithms tend to be unsuitable for our purposes, since they contain many long, skinny triangles, and triangles of strongly varying size. These meshes are also not water-proof, in general.

We therefore developed a new way of generating an approximate triangle mesh of the visual hull by using an implicit formulation of the silhouette cones. The visual hull is described by the Boolean operations

$$VH = \bigcap_i \left[\left(\bigcup_{l \in OC_i} C_{i,l} \right) \cap \left(\bigcap_{m \in HC_i} C_{i,m} \right) \right], \quad (1)$$

where OC_i is the set of outer object contours in image i, HC_i is the set of hole contours and $C_{i,j}$ is the silhouette cone for contour j in image i.

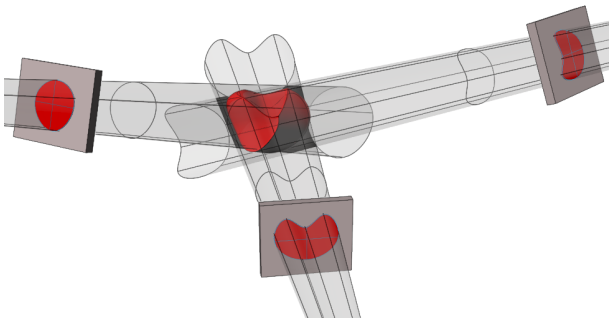

Figure 1: Visual Hull as the intersection of 3 contour cones

Our implicit formulation makes it straightforward to express these operations. On every point of a voxel grid, we can compute the *signed distance function* $D(V, \mathbf{x}, \mathbf{d})$, which represents the distance of a point $\mathbf{x}$ from the boundary of a volume V *along direction* $\mathbf{d}$ (where $\mathbf{d}$ is any of the three coordinate axes in our case). The distance is negative inside the volume, and positive outside. We can compute $D(C_{i,j}, \mathbf{x}, \mathbf{d})$, the signed distance of any point from the silhouette cone $C_{i,j}$ efficiently by projecting $\mathbf{x}$ and $\mathbf{d}$ into image i and performing a 2D ray intersection with contour j. The sign of the result is determined by a point in polygon test.

The directed distance function for the visual hull can then be expressed by transforming the Boolean operations from Equation 1 into min/max operations:

$$\begin{aligned} D(VH, \mathbf{x}, \mathbf{d}) &= \max_i [\max(D(O_i, \mathbf{x}, \mathbf{d}), D(H_i, \mathbf{x}, \mathbf{d}))] \\ &\text{with} \\ D(O_i, \mathbf{x}, \mathbf{d}) &= \min_{l \in OC_i} D(C_{i,l}, \mathbf{x}, \mathbf{d}) \\ D(H_i, \mathbf{x}, \mathbf{d}) &= \max_{m \in HC_i} D(C_{i,m}, \mathbf{x}, \mathbf{d}) \end{aligned}$$

To generate a triangle mesh from this representation, we sample this implicit function at a regular voxel grid, and apply the extended marching cubes algorithm [8], which makes use of the signed distance function to preserve sharp features.

Note that the resulting mesh is not the exact visual hull, and that it also is not conservative (i.e. it does not necessarily enclose the true visual hull or the actual object) because of the discretization: only feature sizes larger than the voxel spacing will be captured. This is not a big concern in our application, since we only use the resulting mesh as a starting point for an optimization. It does mean, however, that our optimization algorithm will also have to consider locally growing the object instead

of just carving parts away. For objects that have a lot of detail in some places, but are smooth in other areas, we can also create the visual hull surface at a high resolution initially, but then use polygonal simplification to reduce the complexity in the smooth areas before going to the next step of our algorithm.

3 Cost Function

The triangle mesh computed in the first stage represents a first approximation of the shape of the object. In the optimization stage, we use both color and silhouette information from the images to improve this initial approximation.

The goal of the optimization is to move the vertices to obtain a mesh that is *photo consistent* with the original images. As defined by Kutulakos and Seitz [10], this requires that two conditions are met:

- The projection of the mesh *exactly* covers the silhouettes of the object in all images.
- A radiance function can be assigned to every point on the mesh, which is consistent with the color information in every image that sees this point.

For Lambertian materials it follows that each point on the mesh has to project to pixels of the same color in all images in which that surface point is not occluded. Reversing the direction of projection, this means that the color information in all images, projected onto the mesh, must be consistent on the mesh's surface. We use graphics accelerators to perform this occlusion aware projection of the images onto the mesh (see Section 5). At the same time we also enforce consistency with the silhouettes, avoid interpenetrations, and include regularization terms for curvature and triangle shape for improved stability. All these aspects are combined into a single function describing the cost of any vertex v in dependence of its position:

$$E(v) = \lambda_i E_i(v) + \lambda_s E_s(v) + \lambda_d E_d(v) + \lambda_c E_c(v) + E_p(v),$$

where E_i measures the inconsistency of the intensity/color information in a neighborhood of the vertex. E_s associates a cost for the difference between the object's and the mesh's silhouettes in the images with the vertex. E_d measures the distortion (skinniness) of triangles and E_c the local curvature of the mesh. E_p is a penalty associated with self-penetration of the mesh. The λs are user defined weights. In general, they are chosen so that E_i and E_s dominate. The individual terms are described in more detail in the following.

3.1 Color Term

We measure the inconsistency of the color information from the different images for the neighborhood of a given vertex v by using a criterion similar to the one introduced by Fua and Leclerc [5]. We sample the surface and then for each sample $\mathbf{x}_j$ first average the color information in the images that see $\mathbf{x}_j$:

$$\bar{I}(\mathbf{x}_j) = \frac{\sum_{i=1}^{n} s_i(v)\gamma_i(\mathbf{x}_j)M_i(\xi_i(\mathbf{x}_j))I_i(\xi_i(\mathbf{x}_j))}{\sum_{i=1}^{n} s_i(v)\gamma_i(\mathbf{x}_j)M_i(\xi_i(\mathbf{x}_j))},$$

where $s_i(v)$ is the sampling rate at which image i samples the triangle fan around v and γ_i is the visibility function for image i; it is 1 if the sample is visible in image i and 0 otherwise. ξ_i is the projection that maps from world space to the image space of image i, and M_i is the binary silhouette image. The value of M_i is 1 for object pixels and 0 for background pixels. Finally, I_i denotes the color information in image i.

The contributions of the images are weighted by their sampling rate (image pixels per mesh area) for this part of the mesh. The error for each sample is then taken to be the sum of the squared differences between the color in the individual images and the average color, again weighted by the sampling rates:

$$e(\mathbf{x}_j) = \frac{\sum_{i=1}^{n} s_i(v)\gamma_i(\mathbf{x}_j)M_i(\xi_i(\mathbf{x}_j))[I_i(\xi_i(\mathbf{x}_j)) - \bar{I}(\mathbf{x}_j)]^2}{\sum_{i=1}^{n} s_i(v)\gamma_i(\mathbf{x}_j)M_i(\xi_i(\mathbf{x}_j))}.$$

To generate the sample locations, we first determine the largest sampling rate for the triangles adjacent to v in any of the images which actually see this part of the mesh. We then sample the triangle fan around v accordingly using this rate. Given the $e(\mathbf{x}_j)$ for all samples, E_i is then computed as

$$E_i = \frac{1}{|V^m|} \sum_{j \in V^m} e(\mathbf{x}_j),$$

where V^m is the set of samples that are visible in multiple images.

3.2 Silhouette Mismatch

The term E_s penalizes a mismatch between the silhouettes of the current mesh and the actual object silhouettes. A penalty is associated with vertex v, if that vertex is assumed to be (at least partially) responsible for this difference. E_s is composed of two contributions

$$E_s = \lambda_s^{>} E_s^{>} + \lambda_s^{<} E_s^{<},$$

where $E_s^{>}$ penalizes the mesh growing outside the original silhouettes, whereas $E_s^{<}$ indicates shrinkage

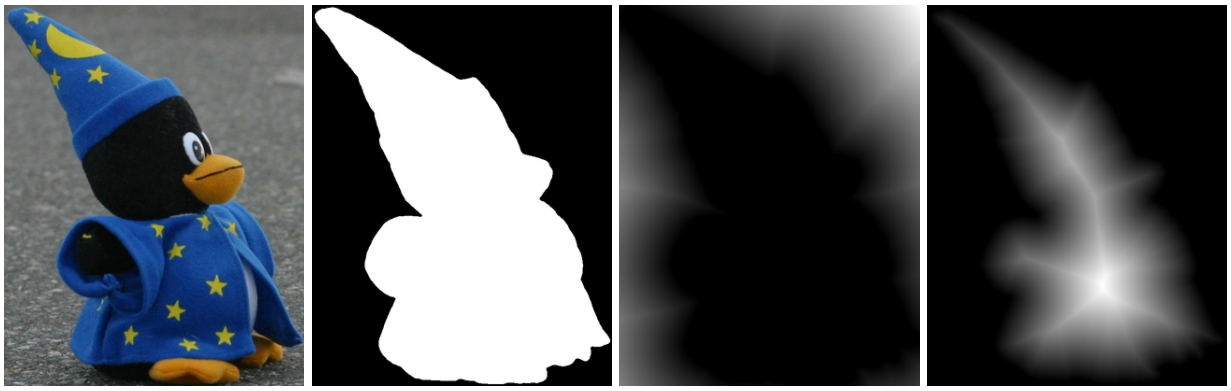

Figure 2: From left to right: original image, silhouette mask, outside distance field D_i, and inside distance field $d_i(v)$.

that results in the mesh's silhouette being smaller than the true object silhouette. The λs are user defined weights.

As no part of the mesh is allowed to project outside the original objects silhouette, it is straightforward to determine which vertices are in violation of this condition. We use the same sampling as described in Section 3.1 and define a distance field D_i for each image, which encodes the distance from the object's contour in image space. Its value is 0 inside the contour and linearly increasing outside. $E_s^>$ is then computed as

$$E_s^> = \frac{1}{n_s \cdot n} \sum_{j=1}^{n_s} \sum_{i=1}^{n} D_i(\xi_i(\mathbf{x}_j)),$$

where n_s is the number of samples, so that the number of samples outside the silhouette, weighted by their distance to it, forms the cost of the vertex.

Similarly, for $E_s^<$, we define a distance field $d_i(v)$ *inside* the silhouettes, which measures the distance of a point in image space from the object's contour. Its value linearly increases from 0 on the contour to 1 on the inside. We compute $E_s^<$ for each vertex v as

$$E_s^< = \min_i d_i(\xi_i(v)).$$

This term causes the mesh's vertices to be attracted to the closest contour cone. However, not all vertices are required to lie on a contour cone. Photo consistency only requires that every part of the real silhouette from every image is covered by the projection of some part of the mesh. For this reason, we choose $\lambda_s^<$ to be small compared to λ_i, so that the $E_s^<$ term is clearly dominated by E_i in textured areas of the object. In areas without color variation however, this term effectively keeps the mesh close to the visual hull surface and prevents it from shrinking.

3.3 Curvature

We use a mesh based measure for the local curvature in the neighborhood of vertex v. For all vertices in a 1-ring neighborhood around v, we compute the sum of the dihedral angles across the edges adjacent to them. E_c is then taken to be the average of these angles. As a consequence, E_c depends on the position of vertices in a 2-ring neighborhood of v. This measure has the advantage that it is more efficient to compute than for example a curvature measure based on polynomial interpolants, and it is also independent of local parameterizations.

3.4 Triangle Distortion

For the stability of the optimization algorithm, but also for the quality of the resulting mesh, it is essential to prevent degenerate triangles. We use the following triangle quality measure, which is based on the Frobenius norm of the triangle edge matrix [15]:

$$\tau = \frac{a^2 + b^2 + c^2}{4\sqrt{3}A} - 1,$$

where a,b and c are the lengths of the triangles edges and A is the area of the triangle. For an equilateral triangle the value of this measure is 0. For degenerate triangles it approaches infinity. In combination with the color consistency term, the distortion term tends to push vertices sideways across the surface, so that the overall geometric shape remains unchanged, but the distribution of vertices on the geometry is relatively uniform. To compute E_d for a vertex v, we average the τ values for the triangles adjacent to v.

3.5 Self Intersection

During the optimization it is possible for originally distinct parts of the mesh to approach each other

and intersect. While it would be possible to adapt the topology of the mesh accordingly [1], we assume that the topology of the object has been correctly established by the visual hull computation. Thus, we penalize self intersections by associating a very large cost with them. After vertex v has been moved during the optimization, we test the adjacent edges and triangles for intersection with parts of the mesh that are close in 3D space. We use an axis-aligned bounding box (AABB) tree to efficiently determine a small subset of the triangles in the mesh that have to be tested for interpenetration.

4 Optimization Procedure

The actual optimization procedure consists of two parts: given an initial mesh, we first optimize the vertex positions to minimize the total cost function across all vertices

$$\mathcal{E} = \sum_{v=1}^{n_v} E(v),$$

where n_v is the number of vertices in the mesh. Once the average local improvement drops below a given threshold we refine the mesh using $\sqrt{3}$-subdivision [7] and start optimizing the higher resolution mesh. This is repeated until the desired degree of detail has been reached.

At every mesh resolution, we have $3n_v$ degrees of freedom, where n_v is typically fairly large. It is therefore impractical to solve for all unknowns in a global optimization approach similar to [17]. We thus perform local optimization for one vertex at a time, iterating over all vertices until convergence, or until a certain maximum of iterations has been reached. The local optimization is implemented using the downhill simplex method [16]. This method moves the selected vertex v at each iteration and evaluates $E(v)$ at the new position. As $E(v)$ depends not only on the position of v, but also on those of its neighbors and potentially any other vertex in the mesh, it does not make sense to run the local optimization until convergence. Instead, we terminate once the step size of the simplex method gets small enough (smaller than a fixed fraction of the average length of the edges connected to v).

While we could simply process the vertices in order, we exploit the fact that the interdependencies between vertex positions due to visibility and curvature are mostly local, and prioritize vertices accordingly. We choose an approach that takes into account that the interdependencies between the vertices are mainly, but not exclusively local. After a vertex has been moved a substantial amount, it therefore makes sense to first optimize the positions of its neighbors. This is especially true if the change in position resulted in a large improvement of the cost function, since then it is likely that a similarly large improvement is possible for adjacent vertices. At the same time, we also have to make sure all vertices are visited eventually. To this end, we organize the vertices in a priority queue and update their priorities as follows: processed vertices are inserted into the queue with an initial priority that is linearly decreasing over time. This is equivalent to increasing the priority of all unprocessed vertices and ensures that each vertex will be processed eventually. After optimizing the position of a vertex, we furthermore increase the priority of all its neighbors proportional to both the moved distance and the achieved improvement in terms of the cost function.

5 Utilizing Graphics Hardware

The cost function (see Section 3) contains several terms that are expensive to compute. As the cost has to be calculated multiple times during each of the local optimization steps, it is crucial to evaluate it efficiently. To this end, we utilize the computing power and specialized features of graphics accelerators by expressing some of the computations involved in terms of OpenGL operations.

One especially expensive part is the computation of color consistency E_i in the presence of possible occlusions. We implement the sampling of the triangle fan as well as the occlusion aware projection of color information from the different images onto the mesh using hardware accelerated operations. To sample the surface around vertex v, we render the triangle fan around v into an orthographic view, looking onto the fan along the estimated surface normal. The scale of the projection is chosen such that the sampling rate of the triangles matches the maximum sampling rate in the original images. For each image that could potentially see the triangle fan (given the triangle normals and the viewing directions of the image's camera) we then generate a shadow map that encodes the occlusion information. The AABB tree already mentioned in Section 3.5 can be used to restrict the geometry to relevant parts during the creation of this map.

In addition to the shadow map, we also store the color and contour information of the original image in the RGB and alpha channels of a texture. This allows us to project the color information onto the triangle fan using texture mapping. By combining shadow mapping and the contour information in the alpha channel we then can identify all samples that are both unoccluded and project to the inside of the

Figure 3: Stuffed puppy mesh before and after optimization. The blurriness of the texture on the original mesh indicates stereo misalignments due to geometric error.

contour in the given image, giving us the terms γ_i, M_i, and I_i for each sample.

At basically no additional cost the same process also provides us with the distances $D_i(\xi(x_j))$ and $d_i(\xi(v))$ required to compute E_s. To this end, we store an Euclidean distance map (instead of a binary contour mask) in the alpha channel of the textures. This map encodes the distances from the contour (both inside and outside) as described in Section 3.2 and is computed in a preprocessing step using the technique described in [9].

After rendering the triangle fan in this fashion, we read out the framebuffer, and compute the actual error terms E_i and E_s in software.

6 Results

We tested our method with several datasets from which we will show results in the following. The first data set is a stuffed puppy, of which we took 32 images. The visual hull surface generated from the silhouettes consisted of 2808 triangles. To illustrate the reconstruction quality, we show the meshes texture-mapped, where the texture of every triangle is generated by averaging the projected color information from all images that see that triangle. This way, blurry results demonstrate geometric error, since they indicate a misalignment between the projected information from different images.

Figure 3 shows the mesh before and after our optimization procedure. The blurriness of the texture on the visual hull image demonstrates the geometric error in that mesh. After the optimization, the textures line up very well, indicating a high-quality reconstruction. Note that even small specular patches such as the eyes and the tip of the nose do not throw the algorithm off.

Our second example, the Tux mascot, has large areas with very little visual structure. In particular, the head is too dark to show any significant shading effect that could be used as an aid in reconstruction. But also fairly large chunks of the cloak and the feet are solid-colored, and the shading is not always distinctive enough. Tux, just like the stuffed puppy, has plastic eyes that can produce specular highlights from certain angles. We took 34 images of this object.

Figure 4 shows the progression of the algorithm for this data set. From left to right, we see the visual hull surface, the optimization thereof (both 512 triangles), and two levels of subdivided and optimized meshes at 1536 and 4608 triangles.

Like the stuffed puppy data, the results show significant improvements as the algorithm progresses, yielding better and better texture alignment by successively shaping smaller features like the rim of the hat, and producing significant concavities such as the space between the belly and the cape. The cape itself is a very thin structure that our triangle based algorithm represents quite well without requiring excessive detail in other areas. Figure 5 shows the triangle

Figure 4: Progression of the optimization algorithm (left to right): visual hull, optimization at original resolution, and two refined levels.

mesh for the cape in a close-up.

Finally, we performed experiments to determine the importance of the term that avoids silhouette shrinkage. To this end, we ran the algorithm on the Tux data set with and without this term. As shown in Figure 6, the omission of the term results in significant shrinkage of the head area, in which the available information is simply insufficient for the color consistency term to take effect. Other areas that are affected are the right foot and some parts of the cape and hat.

7 Conclusion

In this paper we have presented a mesh-based approach for shape recovery based on silhouette and multiple stereo information. Our approach combines the correct treatment of occlusions, the use of silhouette information in the optimization stage, prevention of self-intersections, and hierarchical subdivision. Our new cost term for preventing shrinkage of the mesh away from the contours proves essential for dealing with objects containing large areas with very little structure. Using our mesh-based approach, we were also able to recover very thin structures that would have required fairly large volume resolutions in level set methods.

References

[1] Y. Duan and H. Qin. A novel modeling algorithm for shape recovery of unknown topology. In *International Conference on Computer Vision 2001*, pages 402–409, 2001.

[2] C. H. Esteban and F. Schmitt. Multi-stereo 3D object reconstruction. In *International Symposium on 3D Processing, Visualization, and Transmission*, pages 159–167, 2002.

[3] C.H. Esteban and F. Schmitt. Silhouette and stereo fusion for 3d object modeling. In *Proc. 3-D Digital Imaging and Modeling, 2003*, pages 46–53, 2003.

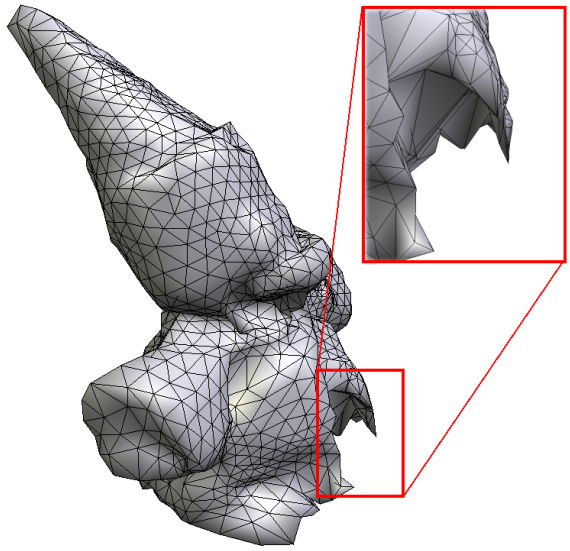

Figure 5: A close-up of the thin geometry representing the cape (also see Figure 4 for textured version).

Figure 6: Optimization with (left) and without (right) the term that prevents shrinkage.

[4] O. Faugeras and R. Keriven. Variational principles, surface evolution, PDEs, surface level set methods and the stereo problem. *IEEE Transactions on Image Processing*, 7(3):336–344, 1998.

[5] P. Fua and Y. Leclerc. Object-centered surface reconstruction: combining multi-image stereo shading. In *Image Understanding Workshop*, pages 1097–1120, 1993.

[6] J. Isidoro and S. Sclaroff. Stochastic refinement of the visual hull to satisfy photometric and silhouette consistency constraints. In *Proc. IEEE International Conf. on Computer Vision (ICCV)*, pages 1335–1342, 2003.

[7] L. Kobbelt. $\sqrt{3}$ subdivision. In *Siggraph 2000, Computer Graphics Proceedings*, Annual Conference Series, pages 103–112, 2000.

[8] L. Kobbelt, M. Botsch, U. Schwanecke, and H.-P. Seidel. Feature-sensitive surface extraction from volume data. In *Proc. of Siggraph*, pages 57–66, 2001.

[9] M. N. Kolountzakis and K. N. Kutulakos. Fast computation of the euclidean distance map for binary images. *Information Processing Letters*, 43:181–184, 1992.

[10] K. Kutulakos and S. Seitz. A theory of shape by space carving. *International Journal of Computer Vision*, 38(3):199–218, 2000.

[11] A. Laurentini. The visual hull concept for silhouette-based image understanding. *PAMI*, 16(2):150–162, February 1994.

[12] T. Matsuyama, X. Wu, T. Takai, and S. Nobuhara. Real-time generation and high fidelity visualization of 3d video. In *Proc. of MIRAGE2003*, pages 1–10, 2003.

[13] W. Matusik, C. Buehler, and L. McMillan. Polyhedral visual hulls for real-time rendering. In *Rendering Techniques '2001*, pages 115–126, 2001.

[14] J. Neumann and Y. Aloimonos. Spatio-temporal stereo using multi-resolution subdivision surfaces. *International Journal of Computer Vision*, 47(1/2/3):181–193, 2002.

[15] P. Pebay and T. Baker. A comparison of triangle quality measures. In *Proceedings to the 10th International Meshing Roundtable*, pages 327–340. Sandia National Laboratories, Oct 2001.

[16] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 1992.

[17] A. Rockwood and J. Winget. Three-dimensional object reconstruction from two-dimensional images. *Computer-Aided Design*, 29(4):279–285, 1997.

[18] S. Seitz and C. Dyer. Photorealistic scene reconstruction by voxel coloring. *International Journal of Computer Vision*, 25(3):1067–1073, November 1999.

[19] G. Slabaugh, W. B. Culbertson, T. Malzbender, and R. Schafer. A survey of volumetric scene reconstruction methods from photographs. In *Proc. of Volume Graphics*, pages 81–100. Springer Computer Science, June 2001.

[20] G. Slabaugh, R. Schafer, and M. Hans. Multi-resolution space carving using level sets methods. In *International Conference on Image Processing (ICIP)*, 2002.

[21] L. Zhang and S. Seitz. Image-based multiresolution modeling by surface deformation. Technical Report CMU-RI-TR-00-07, Carnegie Mellon University, March 2000.