MTMesh: Image Based Mesh Reconstruction and Rendering

M. Sainz, N.Bagherzadeh
Department of Electrical and Computer Science
University of California Irvine
USA
{msainz,nader}@uci.edu

A. Susin
Departament de Matemàtica Aplicada 1
Universitat Politècnica de Catalunya
Spain
toni.susin@upc.es

ABSTRACT
We present a new image based method of reconstructing and rendering models from a set of calibrated input images. First an improved hardware accelerated voxel carving method is used to obtain the voxels corresponding to the volume occupied by the model. Then a new method of real-time multitextured mesh is proposed in order to obtain realistic renders of the recovered models. This representation uses a polygonal relaxed surface mesh obtained from the final set of voxels and overlapping projective texture maps to achieve photo-realistic appearances.

KEY WORDS
Volumetric reconstruction, Voxel carving, Hardware acceleration, Overlapping textures.

1 Introduction

In this paper we present a method for extracting a 3D volumetric representation of an object from a set of calibrated images taken with a digital camera or handheld camcorder. This reconstruction is then processed using a novel view dependent multi-textured mesh model paradigm and we provide an efficient rendering algorithm obtaining high quality realtime renders.

In recent years Image Based Rendering techniques (IBMR) have demonstrated the advantage of using real image data to greatly improve rendering quality. New rendering algorithms have been presented that reach photo-realistic quality at interactive speeds when rendering 3D models based on digital images of physical objects and some shape information (i.e. a geometric proxy). While these methods have emphasized the rendering speed and quality, they generally require extensive preprocessing in order to obtain accurately calibrated images and geometric approximations of the target objects. Moreover, most of these algorithms heavily rely on user interaction for camera calibration and image registration or require expensive equipment such as calibrated gantries and 3D scanners.

Our goal is to extract 3D geometry of the target objects in the scene, based on given camera locations and their respective images. Different approaches such as photogrammetry, stereo vision, contour and/or shadow analysis techniques work with similar assumptions. The complete pipeline to produce 3D models from images starts with a calibration process of the images themselves. There exist several approaches to achieve such calibration as shown in [3]. Throughout this work we assume that this calibration is well known and the objects in the scene are contained within a finite volume.

The next step in the pipeline is a scene reconstruction to obtain a complete model representation that can be used to render novel views of the scene. Depending on the chosen representation for the mode, solutions ranging from point based approaches to complete 3D textured meshes are available in the literature ([2]). We propose a novel model representation that obtains a polygonal mesh from a voxelized reconstruction and uses the images as overlapping texture maps. During rendering, our approach efficiently combines all the images as view dependent projected texture maps obtaining photorealistic renders at interactive speeds.

Summarizing, in this project we present a method, based on image based modeling techniques, that allows the automatic reconstruction of physical objects with all their properties (shape, color and texture) properly recovered. Figure 1 illustrates the block diagram of the suggested pipeline [3] for the 3D model reconstruction from images problem

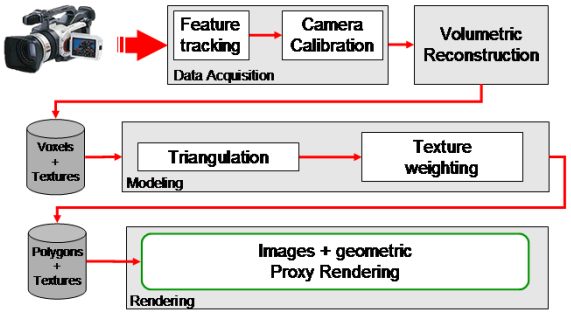

Figure 1. Image Based Modeling pipeline.

The input to the proposed system is a video sequence or set of images taken with an off-the-shelf digital camera or camcorder, by moving it around the physical object that is going to be reconstructed. Then, we use the calibration process presented in [5] to reconstruct the 3D structure of the scene and the motion of the camera analyzing how a set of 2D tracked points move in the image sequence. A volumetric reconstruction step fits a volume of voxels to

the object in the scene using the information of the calibration and the input images. The last stage of the pipeline is a modeling process that transforms the voxelized volume into a polygonal mesh suitable to be rendered using our proposed algorithm in any 3D pipeline.

2 Volumetric Reconstruction

In order to reconstruct the volume occupied by the objects in the scene we have improved the approach presented in [4], that is based on carving a bounding volume using a color similarity criterion. The algorithm is designed to use hardware accelerated features from the videocard. Moreover, the data structures have been highly optimized in order to minimize run-time memory usage. Additional techniques such as hardware projective texture mapping and shadow maps are used to avoid redundant calculations.

The proposed implementation of the space carving methodology consists of an iterative algorithm (shown in Algorithm 1) that has an outer loop that performs a progressive plane sweep along one axis of the bounding box that contains the object to be reconstructed until the set of non-carved voxels remains constant.

```
begin
    do
        generate occlusion maps
        for i=0 to i = maxSlice
            find surface voxels on slice i
            for each surface voxel
                for each reference view
                    if visible and not occluded
                        store footprint data
                end
                evaluate voxel consistency
                if voxel is not consistent
                    mark voxel for deletion
            end
        end
        delete voxels
    while (totalCarved > 0)
end
```

Algorithm 1. Proposed voxel carving algorithm.

On each iteration of the sweep, only the voxels that belong to the surface of the volume slice are processed. Then, during the color consistency test, each voxel is tested against each of the views to determine the set of contributing reference images. Then the voxel footprint is calculated and the color consistency function decides, whether the voxel is consistent and kept, or is inconsistent and removed. This color test is performed by correlating each pair of color histograms of the contributing views, and if more than a threshold of images are correlated (i.e. similar color information), the voxel is considered to be consistent.

The following subsections present more details of the key steps of the carving algorithm.

2.1 Surface Voxels and Occlusion Maps

The main data structure used to store the voxels of the volume is a three dimensional bitmask, that encodes only if a voxel is present or not. The main advantage of such structure is the memory footprint at runtime: a 512x512x512 structure has 16Mbytes of storage.

The voxels on the surface can be easily determined by performing a 6-neighbors test and selecting those voxels with an active bit that have at least one face to face contact with empty voxels.

During the plane sweep and for each of the analyzed voxels, we want to determine the subset of reference images where each of them are visible. Since parts of the current filled volume may occlude the voxels for some of the views, it is necessary to define a set of occlusion maps and use them during visibility determination.

We propose to create at the beginning of each iteration a depth mask for each of the reference views that can be used in a zBuffer test to determine visibility of any given voxel in that view. To determine if a voxel v is visible from reference view i, one can transform the coordinates of the center of the voxel to the view coordinate system and do a simple depth test for visibility determination: if the voxel depth from the reference view is larger than the depthmap value at that projection, the voxel is occluded by the active volume of the scene.

We use the ϵ-z-$buffer$ test concept as presented in [3] which consists of generating a depthmap with an offset of ϵ units by applying a translation of the voxel vertices along their projection rays as shown in Figure 2c. This perspective offset can be applied by exploiting the OpenGL vertex programmability of the new videocards.

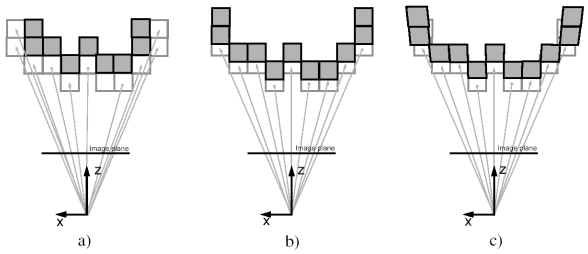

Figure 2. Comparison of ϵ-z-$buffer$ perspective offset (c) versus an one-level eroded volume (a) and a standard depth offset (b).

2.2 Consistency Test

The consistency test is the key step in the voxel carving algorithm. Its function is to decide whether a voxel is going to be kept because its appearance is similar in the reference views were it is visible, or is going to be carved away.

We extend the proposed criterion of [7] consisting in storing the 3D color histogram structure, $CH_j^i(r, j, b)$ for

each footprint j of a given voxel i, defined as

$$CH_j^i(r,g,b) = \frac{\#(r,g,b)_j^i}{\#p_j}, \quad (1)$$

where $\#(r,g,b)_j^i$ stands for the number of times the color value (r,g,b) appears in the footprint j of voxel i, and $\#p_j$ is the total number of pixels included in the footprint.

Instead of looking only for histograms intersections as proposed in [7], we calculate the normalized correlation,

$$N_{j,k}^i = \frac{\sum CH_j^i(r,g,b) \cdot CH_k^i(r,g,b)}{\sqrt{\sum (CH_j^i(r,g,b))^2 \cdot \sum (CH_k^i(r,g,b))^2}} \quad (2)$$

between each pair of histograms and count how many are above a minimum correlation value. When the ratio of high versus low correlated pairs is above a threshold, the voxel is considered as consistent.

To optimize storage of the footprint histograms we quantize the RGB values into a set of only 8 bins for each color channel. This way, the storage needed for a 256 color resolution, 256^3 (more than 16 million), is reduced to only 512 levels. Finally, to account for small color disparities due to sensor sensibility, the bins are extended so they overlap by a given amount (usually a 15%).

3 MTMesh Models

At this point of the proposed pipeline, the reconstruction of the scene is a set of voxels, and the reference images. In order to render the model in a 3D application, we need to define a proper model representation that integrates the 3D recovered information together with the images to achieve a realistic result. In the following subsections we present a novel multitextured mesh, *MTMesh* representation and its hardware accelerated realtime rendering algorithm. The overall algorithm can be summarized in the block diagram of Figure 3.

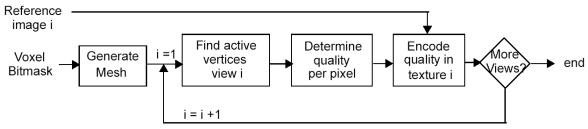

Figure 3. MTmesh creation block diagram.

3.1 MTMesh Generation

An initial smooth triangular mesh covering the surface of the reconstructed scene is created using a variation of the SurfaceNet algorithm [1], that creates a globally smooth surface model from the binary segmented volume data retaining the detail present in the original segmentation. The mesh M is constructed by linking nodes on the surface of the binary-segmented volume and relaxing node positions (see Figure 4) to reduce the energy in the surface net while constraining the nodes to lie within a surface cube defined by the original segmentation.

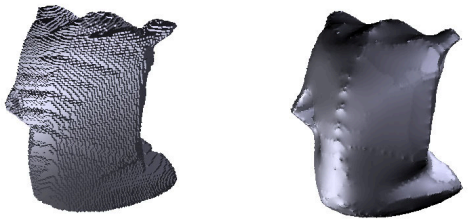

Figure 4. Surface Net reconstructed from the voxel model. On the right we have the same mesh after the relaxation step.

After that, we construct the MTMesh model by defining a set of indexed face lists F_i for each reference view i. Each of these lists contains the faces of the mesh M visible from the corresponding view and the image is used as a projective texture map for the faces.

Moreover, since the relative orientations of the faces and the corresponding image planes are variable, we can define a projection quality factor α_i that weights the contribution of a reference view i to each of the faces of the initial mesh. We calculate this weight in a per vertex basis using $\theta_i(v)$, the relative angle between each vertex and the reference view center of projection

$$\alpha_i = 1 - \frac{\theta_i(v)}{90^\circ} \quad (3)$$

In order to obtain a quality per pixel of the reference image, we can use the OpenGL API to render and interpolate a vertex-colored mesh using the weight factor as the R,G,B and A values. The resulting framebuffer is captured and used to modulate the texture colors and alpha channel as shown in Figure 5.

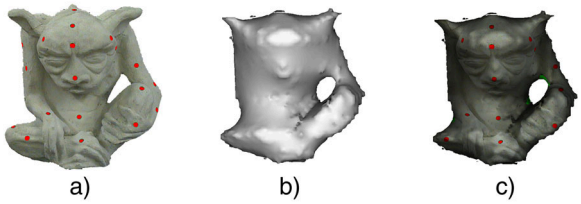

Figure 5. Quality map for a given reference view. a) shows the original image of an object, b) is the quality factors encoded as a greyscale map, and c) is the image with a per-pixel quality factor measure multiplying the R,G and B channels.

The final model consists of the vertices and faces of the mesh M and, for each reference view, the indexed list of visible faces as well as a modified texture map that contains the original image modulated on a pixel basis by the α_i weights.

3.2 MTMesh Rendering

During rendering time, the proposed strategy, given a new viewpoint, is to determine which reference views contribute to the new one and then render them and since they overlap, combine them to produce the final render.

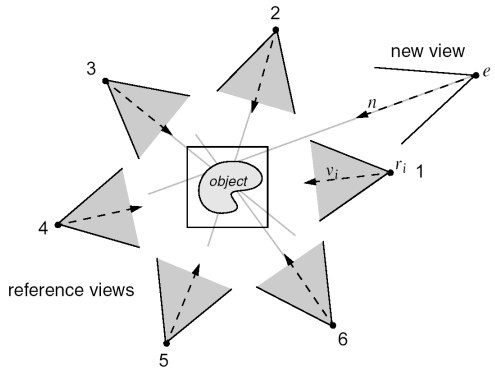

Figure 6. Reference view selection. In this case views 1,2 and 6 will be selected to render the novel view e

As illustrated in Figure 6 given a novel viewpoint e with viewing direction n, a reference view i is selected if the angle $\beta_{i,n}$ between the reference view direction r_i and the new view direction n is less that $135°$, which is a conservative value that will include enough reference views to guarantee a render without holes. Based on this angle we can calculate an angular weight

$$w_i(\beta_{i,n}) = \begin{cases} 1.0 & \beta_{i,n} < 45° \\ \cos(\beta_{i,n} - 45°) & \beta_{i,n} \geq 45° \end{cases} \quad (4)$$

Furthermore, an additional positional weight $w_i(d_{i,e})$ based on the Euclidean distance $d_{i,e} = |e - o_i|$ from the origin of the reference viewpoint o_i to the novel viewpoint is calculated. A final blending weight for each contributing reference image can be obtained as $w_i = w_i(\beta_{i,n}) \cdot w_i(d_{i,e})$ and then normalized such as that $\sum_i w_i = 1.0$.

During rendering, a first pass creates an occupancy z-buffer that only allows the rendering of the triangles closest to the new viewpoint. In the second pass each reference view list of faces F_i is rendered using the calculated weight w_i as a modulating color for the texture T_i. This process gives a final color for each pixel of

$$C_i(x, y) = w_i \cdot \tilde{\alpha}_i \cdot (R_i(x, y), G_i(x, y), B_i(x, y), 1) \quad (5)$$

where $\tilde{\alpha}_i$ is the quality factor, initially associated to each vertex (3), and interpolated for the considered pixel.

By using the appropriate OpenGL blending functions we can render each selected view separately and accumulate the color values obtaining a final pixel color of $C(x, y) = \sum_i C_i$. However, the accumulated weights, which are stored in the alpha channel of the final image do not necessarily sum up to one, requiring a post-render normalization process.

Without any hardware extensions to perform complex per-pixel manipulations this normalization step has to be performed in software. However, widely available graphics accelerators now offer per-pixel shading operators that can be used to perform this normalization more efficiently. In our implementation, we use nVIDIAs OpenGL Texture Shader extension compensating the color intensity deficiency by remapping of the R, G and B values based on the value of α. During initialization time we construct a texture (see Figure 7) encoding in (s,t) of a look-up table of transparency and luminance values respectively, from 0 to 256 possible values. Based on this alpha-luminance map,

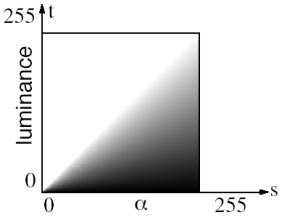

Figure 7. Alpha-Luminance map.

we proceed to correct each of the R,G and B channels of every pixel of the blended image I. Using NVIDIAs texture shader extension operation the graphics hardware can remap the R and α by a texture lookup with coordinates s = α and t = R into our alpha-luminance map. At this point, rendering a quadrilateral with the intermediate image I as texture-one and the alpha-luminance map as texture-two, and setting the color mask to block the G, B and α channels will compensate the red channel by α^{-1} and store it will compensate the red channel by α^{-1} and store it in a new image I_R. Note that only the R and α channels are used by this dependent texture replace operation. Therefore, we need to remap the G and B channels to the R channel of two additional buffers I_G and I_B while copying the α channel as well and is achieved by rendering two quads and using NVIDIAs register combiners. Then the dependent texture replace operation is also performed on the images I_G and I_B. By separating the RGB channels into three different images and using the αR-dependent texture replace operation we get the corrected RGB values in the red channel of three new images that are finally composited into the the final image using NVIDIAs register combiners. Figure 8 illustrates this normalization process.

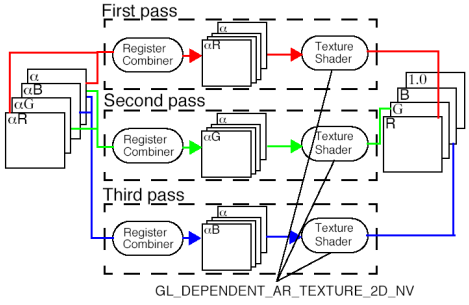

Figure 8. Normalization process based on register combiners

4 Results

Several validation and verification tests were performed with images from a home video recording system with auto-focus and auto-exposure enabled. Two datasets are presented here, consisting of video sequences that have been preprocessed in order to remove the interlacing and to enhanced the color saturation, since the original tapes present a high content of overexposed areas. All the tests and timings have been performed in a PC P4 2.0Ghz with 1Gb of RAM and a NVIDIA GeForce4 Ti 4600 videocard with Detonator drivers 40.03.

The first dataset corresponds to a non-professional recording of a statue in the ruins of Persepolis (Iran), the *two-head-horse* (see Fig. 9, left column) and it contains 702 frames of 720x480 pixels each. The initial sequence is decimated to a final set of 12 frames that will be used during the volumetric reconstruction. The carving process is set to use an initial resolution of 200x176x138 voxels and it performs 165 iterations in 70 minutes, reducing the occupancy of the initial volume to a 43%, generating a final solid model of 4857600 voxels. If the same reconstruction if performed using half the initial resolution, the computation time is less than 10 minutes. The mesh generation takes 12 seconds to produce a colored triangulated mesh of 203391 vertices and 409978 faces. The rendering time is 6.5 frames per second on the above mentioned platform without any level-of-detail simplification.

The second dataset presented here, the *monster* (see Fig. 9, right column) consists of a set of 16 still images of 1024x1024 pixels each taken from an object on a turntable. A manual tracking process of the fiducials on the surface of the object is performed to obtain the proper calibration of the images using [5].

The volumetric reconstruction starts with an initial volume of 250 x 225 x 170 (9562500 voxels), and using five manually selected frames from the initial set, it produces in 43 iterations and 3.5 min. a final reconstruction of 1349548 voxels (a 14% of the initial volume). The mesh model has 150714 vertices and 305516 faces and is rendered at 6 frames per second. The computation time of the smoothed and colored mesh is 4.3 seconds.

5 Conclusions

In this paper we have presented a novel complete pipeline for image based modeling that can be used to reconstruct 3D objects and scenes. Moreover, we propose a new model representation that weights and combines the original images onto a single triangular mesh and can be rendered in realtime using a new rendering paradigm.

We have applied the proposed pipeline to successfully reconstruct objects captured in controlled environments, as well as in field trips to archeological sites where lighting conditions and environment interactions are not ideal.

Currently we are working towards getting more reconstructed datasets and also to improve speed and reliability of the reconstruction algorithm.

The sensor device used for capturing the images is an off-the-shelf digital camcorder, and we have found out that the automatic settings of this type of cameras are not well suited for the purposes of object reconstruction from images because the automatic exposure compensation plays against the voxel carving algorithm by changing significantly the surface color of the objects. We expect to find a feasible way to compensate this on the actual tapes, and for future acquisitions we will design the proper protocol and sensor adjustments to avoid such problems.

Acknowledgements

This research was partially supported by the National Science Foundation under contract CCR-0083080 and by the Comissio Interdepartamental de Recerca i Innovacio Tecnologica, Gaspar de Portola grant C02-03. We would like to thank Dr. Farrokh Shadab for kindly providing his personal video tapes of his trip to Persepolis as data for our experiments.

References

[1] S. Gibson, Constrained Elastic SurfaceNets: Generating Smooth Surfaces from Binary Segmented Data, in *Proc. Medical Image Computation and Computer Assisted Interventions*, pp. 888- 898, October 1998.

[2] M. Pollefeys, L. Van Gool, M. Vergauwen, K. Cornelis, F. Verbiest, J. Tops, 3D recording for archaeological fieldwork, in *IEEE Computer Graphics and Applications*, **23**(3):20–27, May-June 2003.

[3] M. Sainz. *3D Modeling from Images and Video Streams*. PhD. Thesis, University of California Irvine, July 2003.

[4] M. Sainz, N. Bagherzadeh and A. Susin, Hardware Accelerated Voxel Carving, in *1st Ibero-American Symposium in Computer Graphics (SIACG 2002)*, Guimaraes, Portugal. pp 289-297, July 2002.

[5] M. Sainz, A. Susin and N. Bagherzadeh. Camera Calibration of Long Image Sequences with the Presence of Occlusions, in *Proc. IEEE International Conference on Image Processing*, September 2003.

[6] J. Shi and C. Tomasi, Good Features to Track, in *Proc IEEE Conference on Computer Vision and Pattern Recognition*, pp. 593-600, 1994.

[7] M. Stevens, B. Culbertson, T. Malzbender. A Histogram-Based Color Consistency Test for Voxel Coloring in *Proc. of International Conference on Pattern Recognition*, Vol. 4, pp. 118-121, 2002.

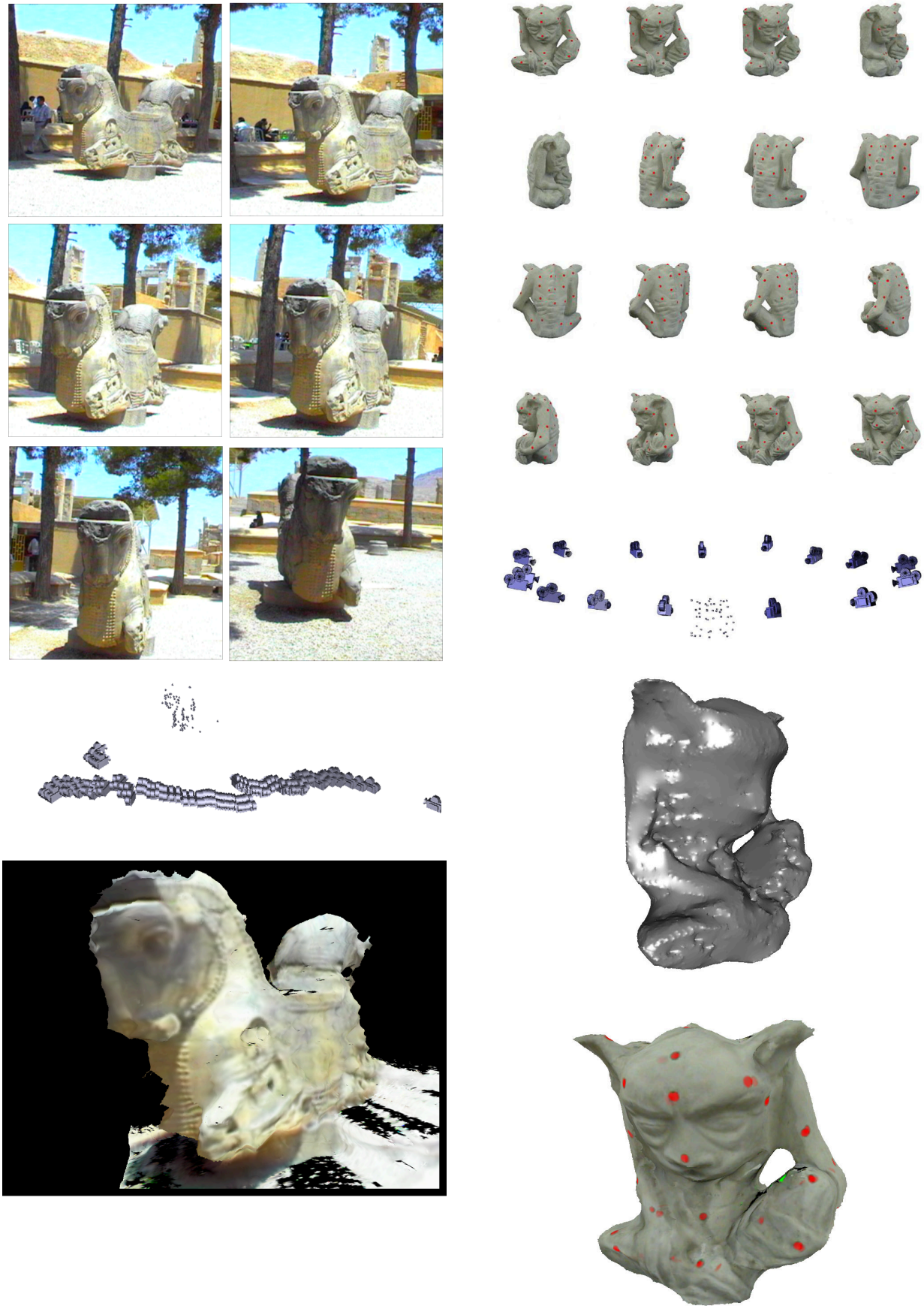

Figure 9. Reconstruction results. The left column shows some of the the *two-head-horse* dataset images, the reconstructed camera path and a novel view rendered using the reconstructed mesh. The right column shows the original 16 frames of the *monster* dataset, the calibrated camera path, the reconstructed and relaxed triangular mesh without coloring and a novel rendered view of the object.