OCTREE VOXEL MODELING WITH MULTI-VIEW TEXTURING

Karsten Müller, Aljoscha Smolic, Birgit Kaspar, Philipp Merkle, Tobias Rein, Peter Eisert, and Thomas Wiegand

Fraunhofer Institute for Telecommunications - Heinrich-Hertz-Institut
Image Processing Department
Einsteinufer 37, 10587 Berlin, Germany
{kmueller/smolic/kaspar/merkle/rein/eisert/wiegand}@hhi.de

ABSTRACT

The reconstruction of 3D models of real world scenes and objects and photo-realistic rendering in interactive free viewpoint applications is a challenging task combining image processing, computer vision and graphics. In this paper, we present a reconstruction pipeline for cultural heritage applications. Starting with a number of photographs of a scene, calibration information is obtained and a 3D model is reconstructed using a hierarchical voxel approach. This octree reconstruction generates a 3D geometry that is further transformed into a wire frame model for smoothing and surface primitive reduction. Finally, texture mapping in the form of view-dependent multi-texture rendering, where original views at original camera positions and automatic texture interpolation at intermediate positions during scene navigation is shown. Furthermore, the algorithm was adapted to exploit automatic graphics card processing and interpolation.

1. INTRODUCTION

Multi-view scene and object reconstruction have become widespread in recent years, due to a number of approaches that were introduced recently. Among the 3D geometry reconstruction methods are shape-from-silhouette voxel approaches, which operate on a set of mask information from all views and a set of provided calibration data [4]. To limit the complexity of voxel reconstruction, a hierarchical approach was introduced, namely octree reconstruction [7], that is also exploited for the cultural heritage reconstruction, introduced in this paper. For texturing or coloring of 3D objects, two main classes of algorithms have been developed. The first class contains approaches for voxel-wise coloring [6], while the second class deals with texture mapping. Some of the texture mapping algorithms also make use of hardware acceleration and have been specifically developed to fit the graphics-processing unit's functionalities including view-dependent multi-texturing [3], Light-field mapping [2], or Lumigraph mapping [1]. However, voxel coloring is only suited for a high-resolution voxel model, since each voxel can only be associated with one constant color value. For the chosen hierarchical octree approach, such coloring is rather unsuitable. In this case, some of the voxels might be quite large, depending on the object surface. Moreover, a constant coloring causes the object surface to look identical no matter which viewpoint is selected. Different lighting, reflections, or even varying colors from different viewpoints are not well represented. Moreover, due to the limited voxel resolution, fine color details on the surface cannot be reproduced. Therefore, we suggest texture mapping onto voxels, as it projects a texture patch onto the surface with the texture patch size adapted to the voxel size. Often the reconstructed voxel object is transformed into a wire frame to obtain a smoother surface approximation of the real object and at the same time to reduce the complexity. However, applying a constant color to a triangle in the mesh does not give the desired result and texture mapping is an applicable algorithm for the representation of fine structures with coarse polygonal meshes. Therefore, we present a reconstruction pipeline that uses a number of photographs of a scene, constructs a 3D geometry as hierarchical octree voxel model, and applies a multi-view texturing to exploit automatic graphics card processing and interpolation.

2. CAMERA SETUP AND CALIBRATION

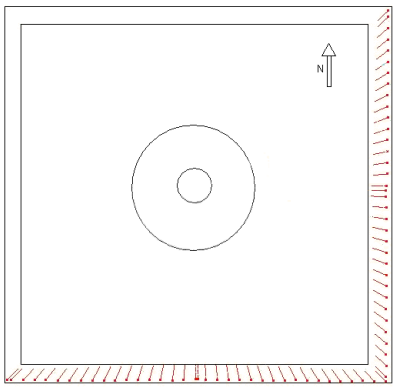

Fig. 1: Camera Positions along the temple

The modeling starts with the calibration of original camera images. Fig. 1 shows the camera positions and directions (as small red arrows) that were used to capture images of a temple in Angkor Vat (Cambodia) that we want to reconstruct in 3D. From this, a projection of significant points into 3D space and resulting calibration and projection information is generated using available calibration software, see Fig. 2.

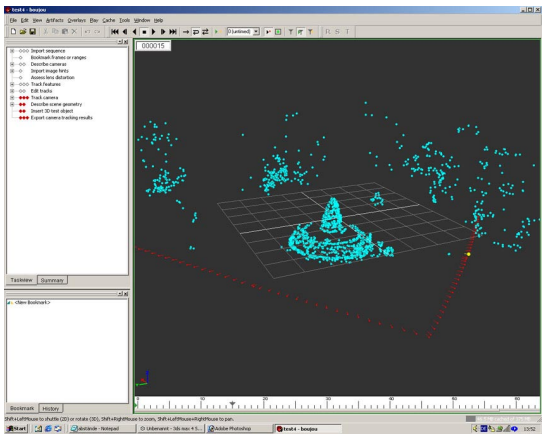

Fig. 2: Projected points in 3D-space

3. OCTREE-VOXEL-GENERATION

After obtaining the projection information, 3D geometry reconstruction starts by applying an octree voxel approach. The algorithm operates on the silhouette information of all views. A user-assisted segmentation is used to obtain this silhouette information for each view. For 3D modeling, an initial cube is placed into the 3D scene such that its projection into all silhouette views covers the interior area completely. In the next stage, the cube is divided into eight equivalent cubes one in each octant. Each subcube is further projected into each view and the following rules are applied:

1. A cube that is completely inside the silhouettes of all views, is not subdivided further, i.e. it is completely inside the object to be reconstructed
2. A cube that is completely outside of at least one silhouette is omitted.
3. All other cubes are further subdivided.

Subdivision is applied, until a certain minimum voxel size is achieved. Fig. 3 shows the different voxel resolutions resulting from the subdivision process, starting with a coarse voxel resolution. Here only remaining voxels of a certain stage are shown, that are also part of the final model.

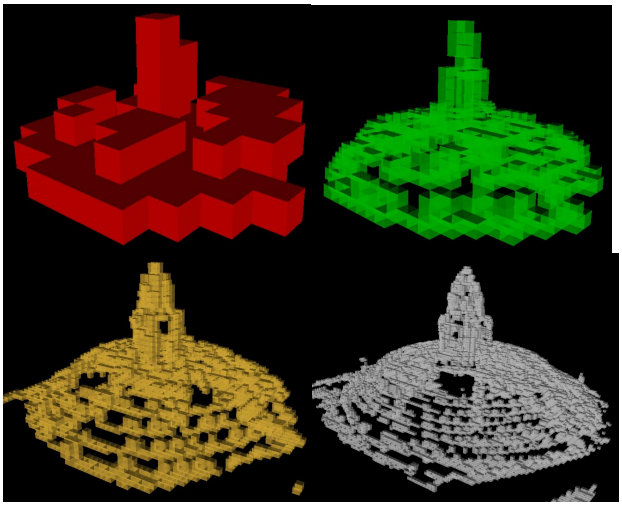

Fig. 3: Different voxel resolutions during the octree generation process

Finally, the octree model is obtained, including all resolution stages, as shown in Fig. 4. For better visualization, the coarsest voxel are drawn in opaque red, while all other resolution stages are drawn in transparent white.

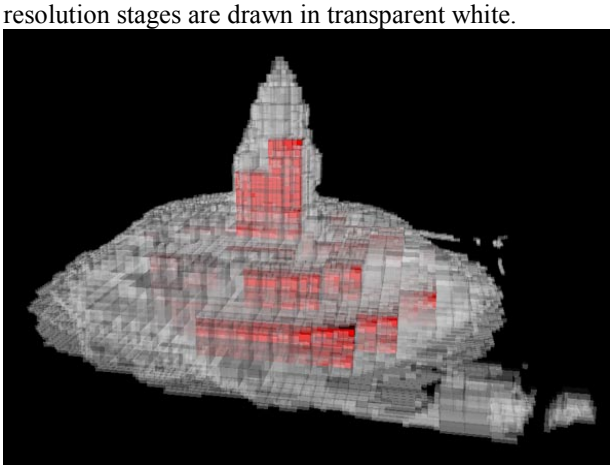

Fig. 4: Reconstructed octree model, coarsest voxel resolution red, finer resolutions transparent

4. SURFACE SMOOTHING

Although the voxel geometry is already suited for texturing, visible artifacts would occur, if texture mapping is applied onto the model at this stage. Therefore, the voxel model is transformed into an irregular triangular wire frame model that better approximates the original 3D object and provides surface smoothing at the same time. A general approach for voxel model transformation is the marching cubes algorithm [5], where all cubes be-longing to the object surface are triangulated. However, to obtain a better approximation, also neighboring cubes need to be considered, such that larger triangles can be formed in surface areas, which are relatively planar. Fig. 5 shows a

wire frame transformation example, followed by a smoothing operation.

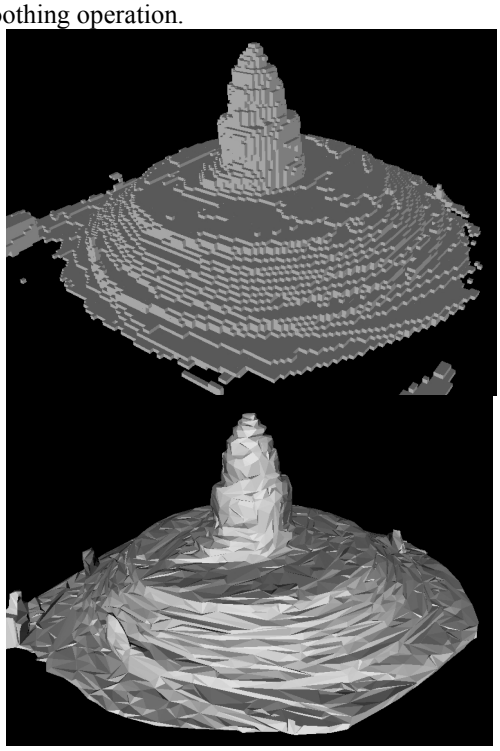

Fig. 5: Transformation of a voxel model into smoothed wireframe

The algorithm begins with the triangulation of all cube faces that belong to the object surface. These faces are first detected applying a marching cubes algorithm. The result is a very dense wire frame, which is smoothed and reduced, using automatic DirectX functionality for mesh processing. To combine also perpendicular faces, resulting from adjacent cube sides, normal vector comparison needs to be neglected for the mesh simplification routines.

5. MULTI-VIEW TEXTURING

Finally, the 3D geometry needs to be textured. As already mentioned, a constant coloring of geometric primitives, i.e. voxel or triangle, is not suited for good viewing quality, as the triangles are rather large and constant coloring does render individual aspects from different views. Therefore, a specific form of multi texturing is applied, that guarantees relatively constant lighting conditions when navigating through the scene and shows original views, if the appropriate viewing direction is reached during navigation. In general, texture interpolation is achieved by weighting the available textures. These weights are usually obtained by taking the dot product of view vector and surface normal vector. For an untextured model, this weighting is shown in Fig. 5 bottom. Each triangle of the surface shows a different lighting, according to its normal vector direction. Unfortunately, the triangular surface structure is clearly visible not only for the untextured but also for the textured model. Therefore, we use an approach, where all surface triangles are illuminated equally for a specific view. Instead of using the individual normal vectors of each triangle separately, a common vector is used for all triangles for a constant texture weight for texture mapping. This common vector equals the camera vector of that view, since we want to have maximum weighting, if the 3D scene is seen from the associated original camera viewpoint. Note that although constant illumination is now achieved for one texture, all other textures have their own texture weight. This results in a texture blending behavior, where textures are blended together, considering their original camera direction. An example of the textured temple model is shown in Fig. 6, where an intermediate view was selected.

Fig. 6: Intermediate view of the textured temple model

For a more detailed reconstruction example, the temple model is shown in detail from two original and one intermediate viewpoint in Fig. 7.

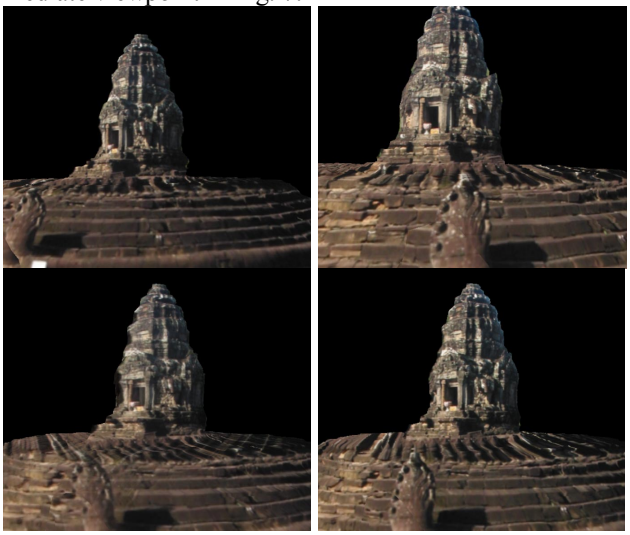

Fig. 7: Model from original viewpoints (*top*), interpolated and corresponding view at that position (*bottom*)

Fig. 7 top shows two original viewpoints of the temple model and Fig. 7 bottom-left an intermediate view position. Furthermore, the original image at that intermediate position is also shown in Fig. 7 bottom-right.

For the texture interpolation, it is necessary to analyze the camera vectors w.r.t. their directions. Consider different textures with their camera vectors C_i and viewing direction v. If texture weights would be calculated for each view independently, e.g. as $\cos\theta$ with θ being the angle between v and C_i, lighting varies considerably during scene navigation. Thus an approach similar to unstructured lumigraph rendering was taken from [8] and applied as shown in Fig. 8.

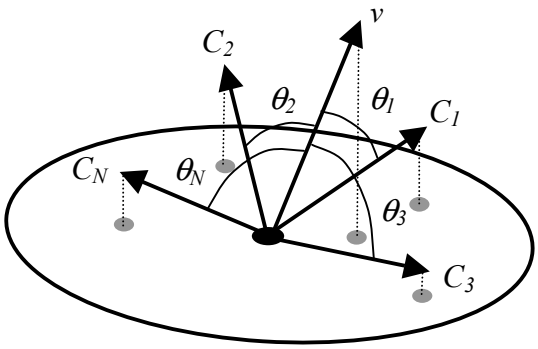

Fig. 8: Unstructured Lumigraph Rendering [8]: Calculate all $\cos\theta$ between viewing direction v and camera vectors C_i

First, cosines are calculated as already described. However, relative weighting functions are calculated as follows.

$$w_i = \begin{cases} \dfrac{\cos\theta_i}{1-\cos\theta_i}, & \cos\theta_i \neq 1 \\ Float\_Max, & \cos\theta_i = 1 \end{cases} \quad (1)$$

This function provides a weight that reaches infinity, if viewing direction and camera vector are parallel and drops to 0, if both directions are perpendicular. For angles larger than 90°, weights are assumed to be 0, since back face clipping is provided. After calculating all relative weights w_i, absolute weights a_i are obtained by normalizing them as follows.

$$a_i = \frac{w_i}{\sum_{\forall i} w_i} \quad (2)$$

This approach results in absolute weights that guarantee constant lighting during rendering and smooth interpolation. Moreover, the original texture is shown when an original camera position is reached. The last condition results from the relative weights being nearly infinite at these positions. Due to the normalization, this weight is finally divided by the sum of all weights, which causes an absolute weight a_i of one, while all other absolute weights are neglected. Rendering is finally carried out, using the texture weights for the associated textures and applying view-dependent multi-texturing as multistage blending.

6. CONCLUSIONS

In this paper, we have shown a 3D reconstruction pipeline from a number of camera images. The images are used to extract calibration information for building a hierarchical octree voxel model. For surface smoothing and surface primitive reduction, a transformation step for obtaining a triangular wire frame model is applied. In the last step, texture mapping is carried out in the form of multi-texturing using adaptively calculated texture weights to provide constant lighting when navigating through the scene. Thus, a 3D model is built using only a limited subset of views for texturing and automatically interpolated remaining intermediate views. Currently, investigations about reconstruction quality are carried out, if different numbers of input textures are used. Furthermore, we also work on 3D geometry extraction with multiple video textures, where a wireframe is required for each time instance.

9. REFERENCES

[1] C. Buehler, M. Bosse, L. McMillan, S. Gortler, and M. Cohen, "Unstructured Lumigraph Rendering", *Proceedings of SIGGRAPH 2001*, pp. 425-432, 2001.

[2] W. C. Chen, J. Y. Bouguet, M. H. Chu and R. Grzeszczuk, "Light Field Mapping: Efficient Representation and Hardware Rendering of Surface Light Fields", *Proceedings of ACM SIGGRAPH*, pp. 447-456, 2002

[3] P. Debevec, C. Taylor and J. Malik, "Modeling and rendering architecture from photographs: A hybrid geometry- and image based approach", *Proceedings of SIGGRAPH 1996*, pp. 11-20, 1996

[4] P. Eisert, E. Steinbach, B. Girod, "Multi-hypothesis, Volumetric Reconstruction of 3-D Objects from Multiple Calibrated Camera Views", *ICASSP*, pp. 3509-3512, Phoenix, Mar. 1999

[5] W. E. Lorensen, and H. E. Cline, "Marching Cubes: a high resolution 3D surface reconstruction algorithm", *Proceedings of SIGGRAPH*, vol. 21, no. 4, pp 163-169, 1987.

[6] S. M. Seitz and C. R. Dyer, "Photorealistic Scene Reconstruction by Voxel Coloring", *Proc. Computer Vision and Pattern Recognition Conf.*, pp. 1067-1073, 1997.

[7] R. Szeliski, "Rapid Octree Construction from Image Sequences", *CVGIP: Image Understanding*, Vol. 58, No. 1, July, pp. 23-32, 1993

[8] D. Vlasic, H. Pfister, S. Molinov, R. Grzeszczuk and W. Matusik, "Opacity Light Fields: Interactive Rendering of Surface Light Fields with View-dependent Opacity", *Proc. Of 2003 symposium on Interactive 3D graphics*, pp. 65-74, 2003.