Hardware Accelerated Voxel Carving

Miguel Sainz Nader Bagherzadeh
Image Based Modeling and Rendering Lab.
Dept. Electrical and Computer Engineering
University of California, Irvine, USA
{msainz,nader}@ece.uci.edu

Antonio Susin
Dynamic Simulation Lab.
Dept. Matematica Aplicada 1
Universitat Politecnica de Catalunya, SPAIN
toni.susin@upc.es

Abstract

In this paper we present a fast hardware accelerated volumetric method for the reconstruction of real world objects from a set of calibrated images. The approach 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 texture mapping and shadow polygons are used to avoid redundant calculations. Hardware texture mapping is also used to reach subpixel precision. Moreover, if the image data can be segmented into foreground and background, a real time implementation of the visual hull algorithm is added to achieve a substantial reduction in computation time. Experimental results are presented on both synthetic and real imagery, showing excellent visual quality of the volumetric reconstruction.

Keywords

Volumetric reconstruction and modeling, voxel carving, visual hull, hardware acceleration, OpenGL.

1. INTRODUCTION

In this paper we present a method for extracting a 3D volumetric representation of an object from a set of unstructured images taken with a still camera or handheld camcorder. 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 ([Debevec96], [Wood00], [Buehler01]) that reach photorealistic 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.

The 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. Recently a set of volumetric techniques based on spatial carving algorithms ([Kutulakos00], [Broadhurst01], [Culbertson99]) was used to reconstruct complex object shapes with excellent results. The common characteristic for these approaches is that they carve a piece of voxelized virtual material that contains the object, similar to an artist sculpting a raw block of marble. A common trade-off for all these methods is the extended computational cost.

The approach presented in this paper was inspired by the space carving theory [Kutulakos00] and addresses the performance problems by improving the data structures and extensively using the hardware support of the OpenGL API implementation on today's video cards.

2. RELATED WORK

The first image based volumetric model reconstruction algorithms were introduced in the mid 80's, and used a *visual hull* [Laurentini94] to approximate the objects. The visual hull can be described as the maximal shape that generates the same silhouette for every reference view outside of the object's convex hull. Beneficial properties of the visual hull include (1) it is guaranteed to enclose the real object, (2) depending on the number of views and the geometry of the object, it can be a tighter approximation than the convex hull, and (3) the size of the visual hull decreases monotonically with the number of views of the object, although there is no guarantee that it will converge to physical object.

On the other hand, Seitz and Dyer [Seitz99] presented a method that can reconstruct scenes with sufficient color variation starting from a volume of voxels encompassing the scene. This method, *Voxel Coloring*, works by testing each voxel on the surface of the volume for color consistency among all the images in which it is visible. The color consistency test consists of projecting the voxel to all the reference images and performing a similarity test of all the projections. If the voxel color value is similar in all the images the voxel is consistent and will be kept. Otherwise, the voxel is rejected and removed from the volume. The algorithm will stop when all the visible vox-

els are consistent with the images.

A limitation is that the visibility of each evaluated voxel has to be calculated. Since this operation is performed many times during the process it must be done efficiently. To overcome this, Seitz and Dyer imposed constraints on the camera locations such that the scanning of voxels was done in increasing distance to the cameras. To avoid evaluating voxels that are occluded by consistent voxels previously evaluated, a mask image is kept per view storing which pixels of a specific view have been validated.

Overcoming the camera placement limitation, Kutulakos and Seitz [Kutulakos00] presented *Space Carving*, an implementation of *Voxel Carving* with no camera constraints. This approach uses multiple scans, typically across the positive and negative axes. Each scan is forced to be near-to-far relative to the *active cameras* at each iteration. Occlusion maps are kept to avoid the redundant evaluation of occluded voxels.

Another approach, *Generalized Voxel Carving*, described in [Culbertson99] presents an efficient implementation of the unconstrained voxel carving algorithm. Culbertson et al. implement two different approaches that use buffers to store surface voxel visibility for each pixel. In addition the Z-Buffer algorithm is used to speed up the carving process by rendering the voxels depth sorted onto the images. This approach obtains very good results, at the cost of performance and rapidly growing data structures.

Eisert, Steinbach and Girod [Eisert99] proposed *Multi-hypothesis Voxel Coloring*, which is quite similar to *Voxel Coloring* and *Space Carving*. The advantage is that instead of evaluating a voxel in each image with the additional cost of visibility estimation, all the voxels are evaluated at once for every image, establishing a color hypothesis for each of them, and then comparing all the hypotheses obtained per voxel at the end. The trade-off of avoiding the visibility computation is that it is necessary to maintain hypotheses for all voxels (interior ones included), increasing the number of hypothesis evaluations.

A more extensive survey of the different carving methods is presented in [Slabaugh01]. While the different approaches to the voxel coloring problem suggest the use of some sort of hardware acceleration to improve runtime performance, none of them have extensively explored the possibilities that graphic processors provide.

3. ACCELERATED VOXEL CARVING

The algorithm presented in this paper extensively uses the OpenGL API, greatly supported in hardware by most videocards.

The presented technique performs six progressive carvings steps by sweeping a plane, the *carving plane*, along the positive and negative direction of each axis of the bounding box that contains the object to be reconstructed (Figure 1). At each iteration step only the cameras visible to the carving plane are used to test which voxels in that plane will be kept.

We assume that the calibration parameters of each image are well know, that is, we know the camera location, orientation and its intrinsic parameters. Different methods for the recovery of these parameters are available. For this implementation these values were obtained for the synthetic images computed from the 3D datasets and used for validation purposes. In the case when real imagery is used we apply a method presented in [Sainz02] to perform an autocalibration of the cameras based on 2D features of the images.

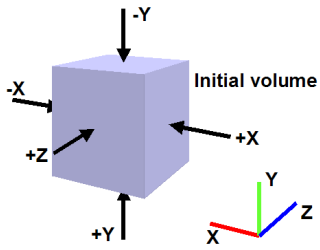


Figure 1: Multiple sweep space carving

We use an octree data structure to keep track of the set of consistent voxels throughout the carving process. The approaches [Eisert99] and [Culbertson99] start with an initial solid voxel volume, which during the carving process will shrink to the final number of voxels. For higher resolutions (*n=256* or more) the amount of memory required to store the data becomes very large (Culbertson et al. report up to 462Mb for a 167x121x101 voxels dataset). Instead of using a preallocated volume only the root node of the corresponding octree is initialized. The octree is then adaptively subdivided as consistent voxels are found.

The main steps of the algorithm are:

```
1 CARVING(n)
2 initialize carving plane to n^2 voxels and
3 empty octree root node
4 for dir = 1 to 6 sweep directions
5   for coord = 1 to n
6     do select all active cameras
7       for c = 0 to all active cameras
8       do project image c onto plane
9          project shadow voxels onto plane
10         extract visible voxel statistics
11         perform consistency check on voxels
12    for i = 1 to length[new_voxels]
13      if new_voxel[i] exists in octree
14        do check consistency with old data
15          if new_voxel[i] still consistent
16            do merge values
17          else
18            do reject voxel and remove
19               voxel from octree
20      else
21        do add new_voxel[i] to octree
```

Figure 2: Pseudocode for the carving algorithm

The following subsections will provide more detail about the primary components of the algorithm.

3.1 Image Projection

The consistency check criterion evaluates the similarity of the color values of a voxel *footprints* for each of the views. A common way to determine how a voxel projects to an image is to rasterize the voxel in the image plane. Since this is a basic operation supported by the OpenGL API, it may seem straight forward to render the voxel from the camera's point of view. However, when the angle of the camera in respect to the normal of the carving plane is large and the voxel resolution is high, some voxels might be projected to the same pixel locations due to rounding from 3D floating point to integer pixel coordinates. This may result in some missing projected voxels, or some voxels with smaller footprints than they should have, contributing erroneously to the consistency check. The presented approach overcomes this by projecting the images onto the carving plane using texture mapping techniques.

The front faces of the voxels on the carving plane are rendered from a virtual viewpoint perpendicular to the plane. To maximize the rendered area of the carving plane on the framebuffer, the view frustum is set to 90 degrees and the virtual view point is located such that the carving plane rendering fills the entire image plane. The projection of the image is then performed with projective texture mapping (Figure 3).

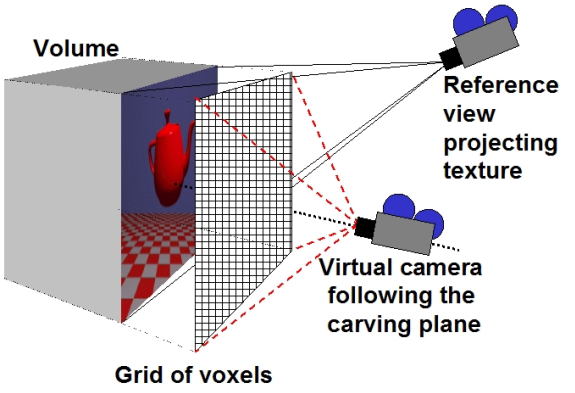


Figure 3: Texture projection onto carving plane

During the initialization process all images are stored as textures in the video card's memory. For each image that has to be evaluated, the voxels of the plane are then rendered with texturing enabled. To assign the proper texture coordinates to a voxel, first the texture matrix stack is set to the full reference camera projection matrix, that is the *intrinsic*extrinsic* matrices of the camera. This generates texture coordinates that are a perspective projection of image coordinates from the camera's location. By assigning the same 3D coordinates of the voxel corners as texture coordinates, the projection of the texture onto the voxel surface is achieved.

This approach has two advantages, (1) each voxel presents a constant footprint for each view, facilitating the registration of the footprints and the statistical calculations and (2) since OpenGL texture mapping interpolates the textures, subpixel accuracy can be reached at no extra cost.

3.2 Shadow Voxels

To simplify the computation and avoid evaluating voxels that are occluded by consistent voxels of previous iterations of the same sweep, Kutulakos and Seitz used bitmasks for each image to mark which pixels were already assigned to a voxel footprint.

In our approach textures are projected onto the voxels of the carving plane. Textures can then be modified to reflect which voxels were already assigned by coloring them black or creating a transparency mask. Unfortunately, modifying the textures in OpenGL can be costly as it implies rasterizing the voxels on each image, rescaling the images to the right texture size and transferring them back to texture memory. Furthermore, if an assigned voxel is deleted later on in another sweep direction, the original colors in the texture have to be restored. We have found a better approach that draws the assigned voxels as shadow polygons onto the carving plane (view Figure 4) instead.

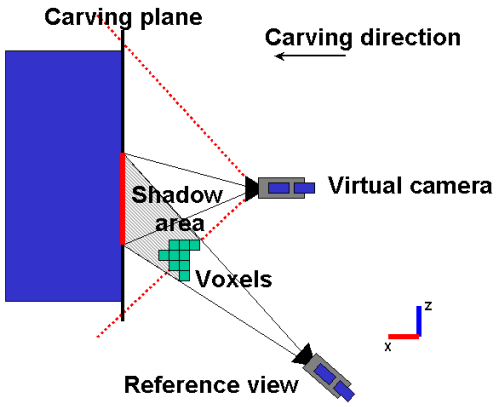


Figure 4: Shadow voxels projection

For a specific view, the camera location is considered as a light source, and the set of accumulated voxels for that specific carving sweep is rendered in black using the planar projected shadow method described in [Blinn88]. Given the ground plane equation (our carving plane) and the homogenous position of the light (the camera location in 3D) a 4x4 matrix, called planar projected shadow matrix, can be constructed that projects any 3D polygon onto the ground plane based on the light position. If this matrix is concatenated with OpenGL's modelview matrix, the shadows will be rasterized on top of the ground plane. The complete projection of a reference view onto the carving plane can then be achieved as follows:

1) Render the color coded carving plane and store the voxel mask.
2) Render the projective texture onto the plane.

3) Calculate the view specific shadow matrix of the plane.
4) Concatenate the shadow matrix to the view transform.
5) Render voxel geometry in black.
6) Capture the framebuffer for color consistency check.

Our *shadow voxels* significantly improve the overall performance of the carving process, by removing useless voxels before the consistency check is performed.

3.3 Footprint Registration

As described earlier, the textured voxels are painted from a virtual view point perpendicular to the carving plane covering the full image space. The color information is then captured from the framebuffer for each of the active views once the corresponding texture has been projected. Next the correct pixel to voxel mapping is determined.

During initialization, a generic color coded mask is created to encode the ID of the voxels on the carving plane. Since the plane contains *N x N* voxels, a checkered pattern is created matching the size of the carving plane. Each square of the mask is then assigned a color that encodes its location in the mask. Assuming 32-bit color buffers are supported, a position can be encoded with up to 16 bits per coordinate (e.g. X = RG and Y = BA). In our implementation we use an 8-bit encoding, using the red channel as X and the green channel as Y, limiting the maximum resolution to *N=256*. Since the virtual view of the carving plane always remains constant, the *voxel mask* is created once and reused throughout the multisweep carving process.

To perform the footprint registration we raster scan the framebuffer and accumulate the red, green and blue values on the corresponding voxel data structure, identified by the color ID in the voxel mask. The values stored in each voxel are the sum of values of red, green and blue and the number of pixels per voxel.

3.4 Consistency Check

Once a list of the different footprints of all the visible voxels is available, a comparison of the footprints is performed to determine if a voxel is color consistent with all the images where it appears.

The original algorithm presented by Seitz ([Seitz99]), considered the color of the projection of the voxel center as the color value of the footprint. Unfortunately, using a single pixel color per voxel, important information about the color distribution in the footprint is lost and the method is very sensible to the sensor's noise. Broadhurst and Cipolla [Broadhurst00] propose to use a statistical criterion based on the variance analysis and the *f-snedecor* statistical function. However, when voxel resolution is large, the footprints have a small area in pixels, and the low number of samples causes the statistical analysis to be inaccurate.

In the presented approach we determine the consistency of a voxel by performing a distance measurement in normalized color space of the pixels of the footprints [Steinbach00]. That is, between each image pair *i, j* we apply the following mapping

$$dR_{i,j} = \left| \frac{\bar{R}_i(X_i, Y_i)}{\overline{RGB}_i(X_i, Y_i)} - \frac{\bar{R}_j(X_j, Y_j)}{\overline{RGB}_j(X_j, Y_j)} \right|$$

$$dG_{i,j} = \left| \frac{\bar{G}_i(X_i, Y_i)}{\overline{RGB}_i(X_i, Y_i)} - \frac{\bar{G}_j(X_j, Y_j)}{\overline{RGB}_j(X_j, Y_j)} \right|$$

$$dB_{i,j} = \left| \frac{\bar{B}_i(X_i, Y_i)}{\overline{RGB}_i(X_i, Y_i)} - \frac{\bar{B}_j(X_j, Y_j)}{\overline{RGB}_j(X_j, Y_j)} \right|$$

$$dR_{i,j} + dG_{i,j} + dB_{i,j} \leq threshold$$

with $\overline{RGB}_i(X, Y) = \bar{R}_i(X, Y) + \bar{G}_i(X, Y) + \bar{B}_i(X, Y)$

The normalization of the colors by the sum of components increases the robustness in respect to varying illumination conditions between the different images.

All the voxels that pass the footprint consistency check are stored in a list and later added to the carving volume octree. It can occur that a voxel has been evaluated under different views in previous carving sweeps and already exists in the octree. To guarantee that all cameras visible to a voxel are taken into account an extra consistency test is performed between the color values stored in the octree and the new ones. If the test passes, we merge the color values for that voxel by averaging them. A failed test indicates that a subset of the cameras visible to that voxel is inconsistent and that both the new and existing voxel can be safely removed.

3.5 Visual Hull Computation

When the reconstruction is applied to a set of views around a target object, most of the voxels in the initial volume will be carved away as they are outside the object. Generally, this is not taken into account and all voxels will still be analyzed and subsequently carved, increasing the runtime of the algorithm. However, it is possible to segment the images into *foreground* (the object itself) and *background*. This process can be automated if background reference images are available, a case commonly encountered when the object is mounted on a turntable and the camera is fixed. Alternatively, the user can manually segment the images with standard image manipulation software. Assuming that a set of views of an object is available, with the background marked in a specific color or transparency value, it is possible to check if a voxel belongs to the *visual hull* of the object. This test will be true when the footprints of the voxel on all images are in the foreground pixels. If there exists at least one image in which the footprint falls into the background pixels, that voxel is outside the visual hull and will not be considered during the carving process (Figure 5).

Using transparency, projective texture mapping and stenciling, Lok [Lok01] described a real time algorithm to obtain the visual hull from a set of segmented images. Our implementation performs the visual hull calculation right before projecting the image textures onto the carving

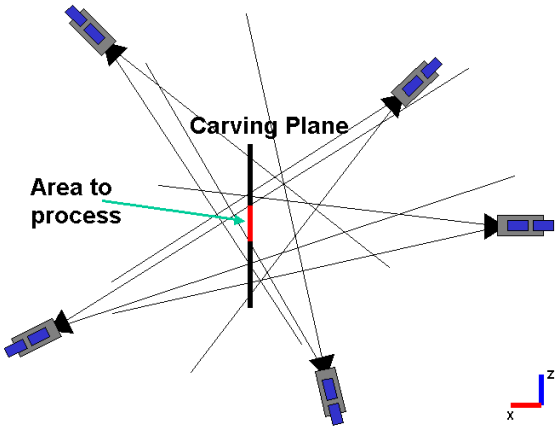


Figure 5: Visual hull test

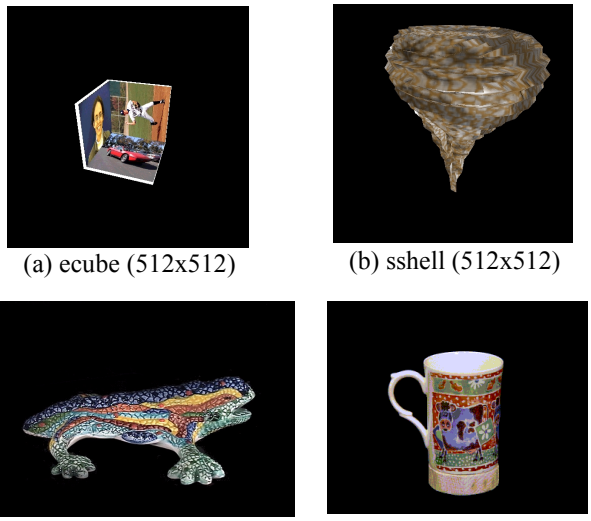
plane. The images are projected onto the carving plane using the projective texture mapping described in Section 3.1. Initially buffers are cleared, stencil and alpha tests enabled to paint only the foreground of the images and background pixels are marked with an alpha value of 1 during the initialization stage. Next all views are rendered for active and inactive cameras. During this process the stencil buffer will accumulate the number of times a given pixel was painted. If all *n* views project foreground pixels to a voxel on the plane, the corresponding stencil values will be equal to the number of cameras and the voxel belongs to the visual hull. Once all *n* views have been projected we change the stencil test function to pass only those pixels that have the maximum value in the stencil buffer. This way, every time a primitive is rendered using the calculated stencil buffer, the visual hull is queried to determine the part of the primitive that is within the real object. Finally the active views will be rendered one by one and stored for the consistency test as described in Section 3.4.

4. EXPERIMENTS AND RESULTS

Several validation and verification tests were performed with real and synthetic images (Figure 6). The validation examples consist of a set of segmented images, the camera calibration parameters (orientation, translation and focal length) and a bounding box encapsulating the object.

The synthetic datasets *ecube* (Figure 6 (a)) and *sshell* (Figure 6 (b)) were constructed defining virtual camera paths around synthetic models and capturing the images from the computer screen. The segmentation of the foreground and the calibration of the cameras were performed automatically. Identical camera parameters were used for both synthetic datasets to allow for the comparison of the results.

Two sets of camera motions were defined consisting of circular paths on the surface of a sphere centered around the center of gravity of the synthetic object. The first motion describes two circles, one at 0 degrees elevation and the other one at 30 degrees. The second motion adds a third circle at an elevation of 45 degrees. At all times the



(a) ecube (512x512) (b) sshell (512x512)

(c) dragon (710x480) (d) cup (352x288)

Figure 6: The datasets

cameras point towards the center of the sphere. All the images in the synthetic sets have a size of *512x512* pixels.

The real images were obtained using handheld cameras and differ in the number of reference images and their size. The *dragon* dataset (Figure 6 (c)) consists of 5 images and the *cup* dataset (Figure 6 (d)) of 14 images in total. The camera calibration was performed using an implementation of the method of linear factorization described in [Sainz02] that performs an autocalibration of the cameras based on 2D features in the images. The segmentation of the images was done manually using standard image manipulation software.

The experiments consisted in reconstructing all the datasets using three different thresholds, to evaluate the influence on the computational complexity and quality of the reconstruction. All tests were performed on a PC system with a 2.2Ghz P4, 1Gb of RAM and a 64Mb NVIDIA Quadro Pro videocard running MS Windows XP with Detonator XP drivers (version 23.11).

The results for the real imagery are highlighted in Table 1 and the results for the synthetic datasets in Table 2. From these values we can conclude that the following factors greatly influence the overall performance and quality

Consistency threshold. The value of the threshold can vary between 0 and 3. Three values 0.2, 1.0 and 1.5 were experimentally chosen. The results indicate that when the threshold reaches a certain value, it does not alter the carving results. The reason is that the volumetric reconstruction converges to the visual hull after a certain limit is reached.

Shape of original model. Another factor that plays an important role is the convexity of the object. When reconstructing an almost convex object such as *sshell* (Figure 6 (b)), the computation time is low compared to a concave object such as *ecube* (Figure 6 (a)). In the later case, because of the large concavity of the object, no shadow voxels exist to occlude successive iterations, increasing

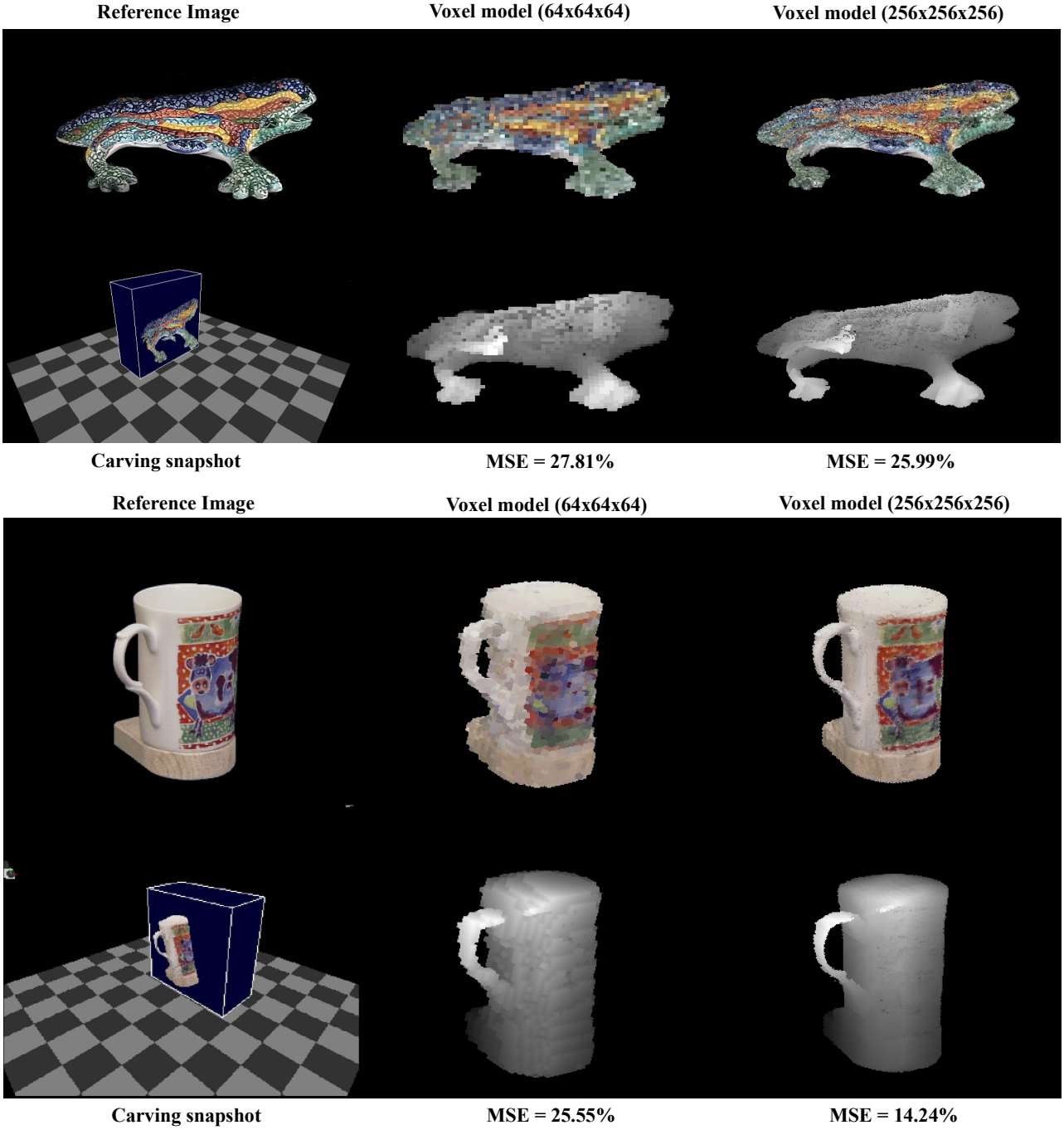


Figure 7: Snapshots of the carving process and recovered models of the real datasets. For each of the datasets, the first row shows from, left to right, one of the reference images, the same view of the reconstructed voxel model at different resolutions. The second row of each dataset shows a snapshot of the carving process and the two corresponding depth-maps of the reconstruction (brighter color indicates closer to the camera). The mean square error of each reconstructed view and the reference image is also shown.

the computation time.

Number of Images. Expectedly the impact of a higher number of images is noticeable when the shape of the object presents concavities. Generally, the image projection and voxel shadowing can be performed in hardware quite efficiently. However, if shadows do not play a significant role, the increased image area that has to be processed leads to the expected performance loss.

Projected size. Another important factor is the amount of area the object occupies on the image, leading to longer processing times for larger projections.

Voxel resolution. It has a direct impact on the computation time and the visual quality of the result. Although this parameter can be set to any desired value, a bound directly related to the size in pixels of a voxel footprint exists. This value should be at least 1 pixel per footprint and small enough such that the color variance in a footprint is insignificant

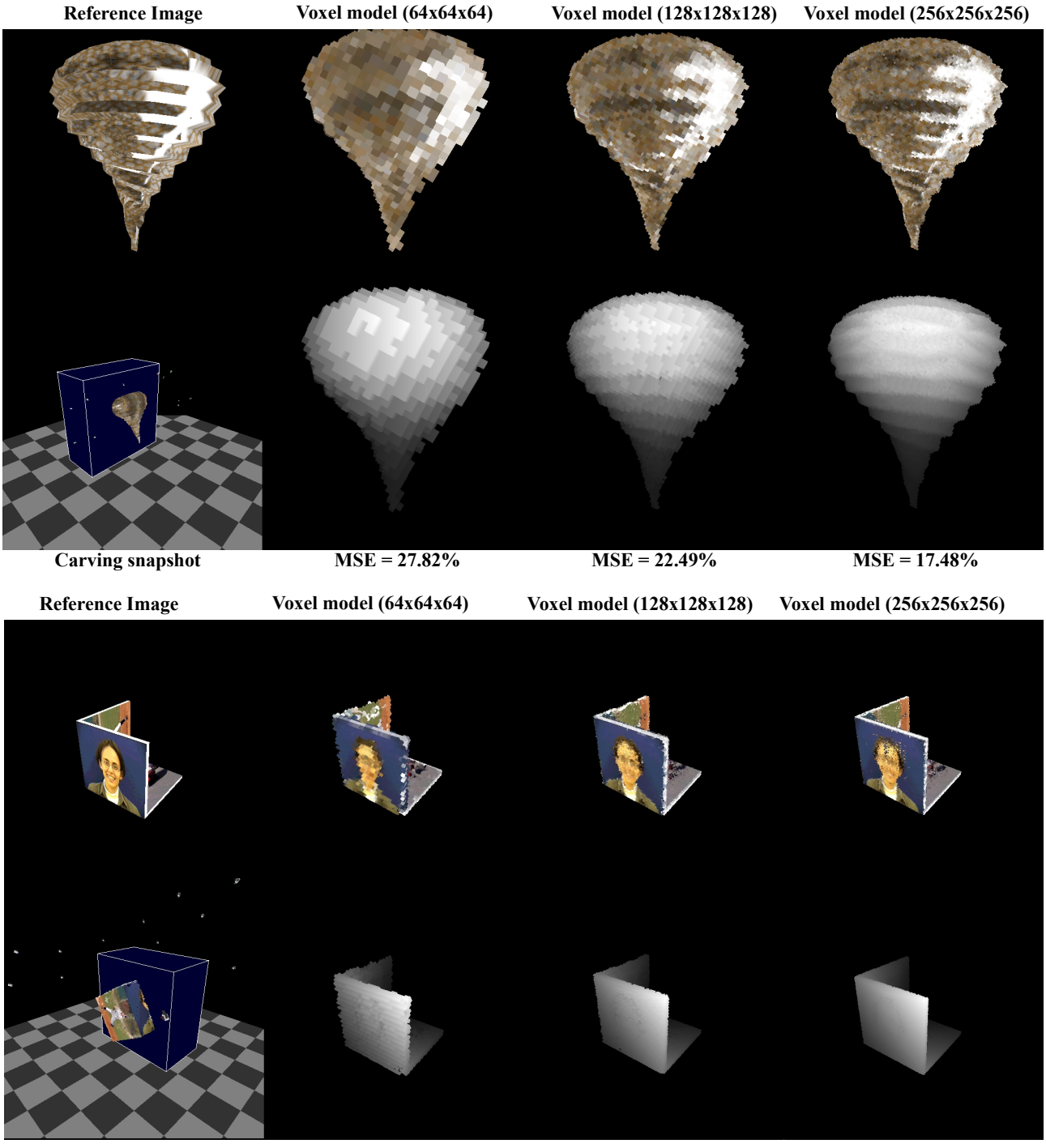


Figure 8: Snapshots of the carving process and recovered models for the synthetic datasets. For each of the datasets, the first row shows from, left to right, one of the reference images, the same view of the reconstructed voxel model at different resolutions. The second row of each dataset shows a snapshot of the carving process and the three corresponding depthmaps of the reconstructions.

.Overall, the algorithm performs well even when high voxel resolution is requested. Figure 7 and Figure 8 show snapshots of the process and the results for the four datasets. The computation time varies between 0.5 to 13 minutes. The *ecube* was the worst model, for which due to the large concavity, penalties are incurred resulting in a similar computation time as the *cup* dataset at half of the resolution.

The final voxelized models present good visual quality from the original calibrated camera positions, and for any arbitrarily chosen location around those cameras. For each of the reconstructed views presented in the figures, we provide the mean of the squared error (MSE) in color difference with respect to the original polygonal rendering for the same view. The color difference is measured between two corresponding pixels as the 3D distance in

RGB color space. The reported MSEs are normalized and denote the percentage of the maximum error in 24bit RGB color space.

The obtained MSE values are higher than expected as the images of the reconstructed models present a clear pixelization due to the voxel size and a unique color assignment to each voxel.

For arbitrary views that are too far apart from the original views the models are not that accurate. This is a well know limitation of image based model reconstruction algorithms, since information not captured in the images can not be reconstructed without additional assumptions. However, with a set of images that provide a good coverage of the object a fairly accurate model can be reconstructed easily as shown in this work. The algorithm has been tested on additional objects with large amount of concavities similar to the one shown in Figure 9. The images and in particular the depthmaps, show that the accuracy of the reconstruction is excellent for complex objects

5. CONCLUSIONS

We have developed a new optimized space carving algorithm that extensively uses OpenGL hardware acceleration. Moreover, an adaptively subdivided octree based data structure is used to limit the required memory footprint.

The system was tested with both synthetic and real world objects and excellent reconstructions results were obtained, eventhough some of the models had non convex surfaces. The proposed method is computationally fast and is suitable for standard PC's, making it a very attractive solution.

		dragon		cup	
resol.	thresh	time[s]	nVoxels	time[s]	nVoxels
64	0.2	33	9146	24	7089
	1.0	33	9727	26	7578
	1.5	32	9748	26	7578
128	0.2	91	39091	105	32328
	1.0	95	40340	120	33867
	1.5	96	40177	120	33867
256	0.2	439	158490	677	128546
	1.0	472	163765	763	130243
	1.5	465	163222	768	130385

Table 1: Carving results for real images

			sshell		ecube	
resol.	#Views	thresh	time[s]	nVoxels	time[s]	nVoxels
64	20	0.2	69	6435	75	11676
		1.0	70	6591	86	16772
		1.5	70	6591	86	16813
128		0.2	228	28946	311	71314
		1.0	236	30416	406	81142
		1.5	236	30416	406	81065
64	30	0.2	103	6898	101	8954
		1.0	106	7057	130	16766
		1.5	106	7057	130	16740
128		0.2	360	32294	380	56863
		1.0	385	32775	671	85313
		1.5	379	32771	670	84967

Table 2: Carving results for synthetic images

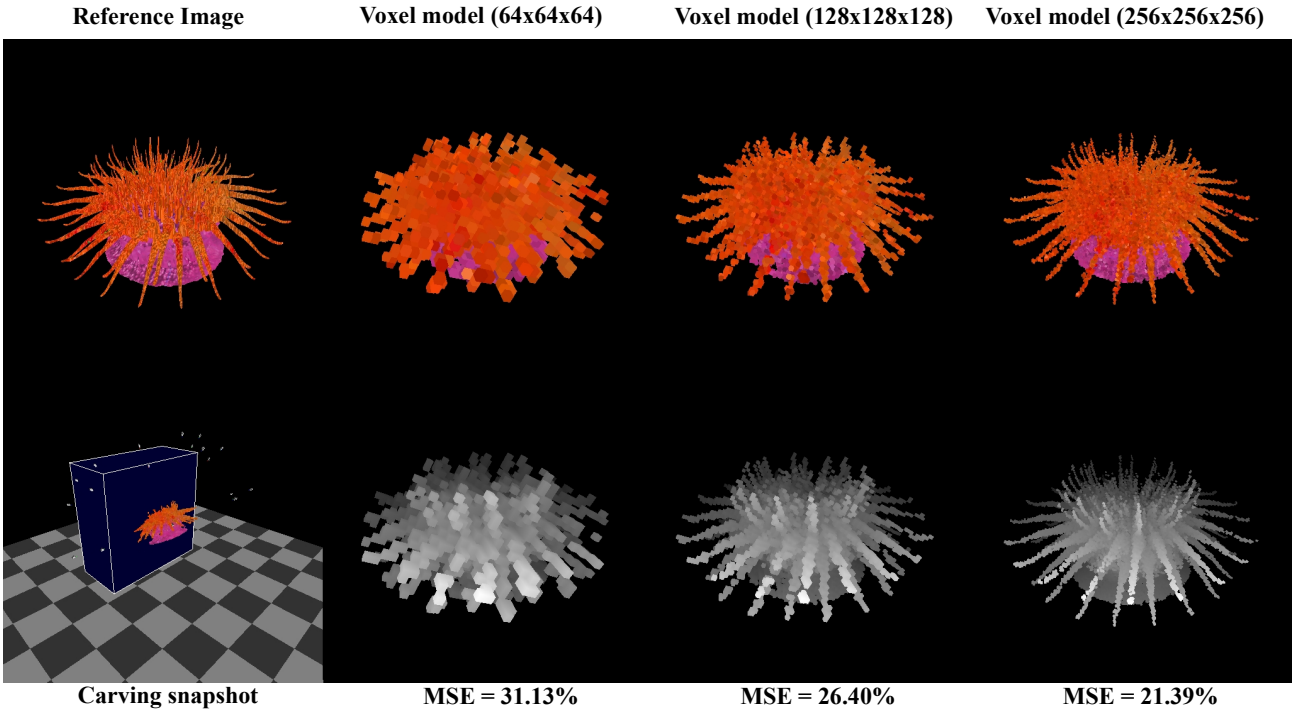


Figure 9: Snapshots of the carving process and recovered models for another synthetic object with high degree of concavities.

The quality of the reconstructed voxel-based models can be further enhanced by using a surface extraction algorithm, such as marching cubes, to obtain a polygonal model representation, which then can be texture mapped with the original images.

6. ACKNOWLEDGMENTS

This research was supported by the National Science Foundation under contract (CCR-0083080) and by the Comissió Interdepartamental de Recerca i Innovació Tecnològica (CIRIT), Gaspar de Portola grant C01-02.

We thank Professor Falko Kuester for his help and fruitful discussions. The *cup* dataset is courtesy of Peter Eisert (www.lnt.de/~eisert/reconst.html). Some of the synthetic models were downloaded from www.mitch3dseite.de.

7. REFERENCES

[Blinn88] J. Blinn, *Me and my (fake) shadow.* In IEEE Computer Graphics and Applications, vol. 9, no. 1, pp. 82-86, 1988.

[Broadhurst00] A. Broadhurst and R. Cipolla, *A statistical consistency check for the space carving algorithm*. In Proc. 11th British Machine Vision Conference, pp. 282-291, 2000.

[Broadhurst01] A. Broadhurst, T.W. Drummond and R. Cipolla, *A probabilistic framework for space carving*. In Proc. of Int. Conference on Computer Vision, I, pp. 282–291, 2001.

[Buehler01] C. Buehler, M. Bosse, L. McMillan, S. Gortler and M. Cohen, *Unstructured Lumigraph Rendering*, In Proc. SIGGRAPH'01, pp. 425-432, 2001.

[Culbertson99] W. B. Culbertson, T. Malzbender, and G. Slabaugh, *Generalized voxel coloring*. In B. Triggs, A. Zisserman, and R. Szeliski, editors, Vision Volumetric Scene Reconstruction 485 Algorithms: Theory and Practice (Proc. Int. Workshop on Vision Algorithms) , volume 1883 of Lecture Notes in Computer Science, pp. 100-115. Springer-Verlag, 2000.

[Debevec96] P.E. Debevec, C.J. Taylor, and J. Malik, *Modeling and rendering architecture from photographs: a hybrid geometry and image-based approach*. In Proc. of SIGGRAPH'96, pp 11-20, 1996.

[Eisert99] P. Eisert, E. Steinbach and B. Girod, *Multi-Hypotheses Volumetric Reconstruction of 3-D Objects From Multiple Calibrated Camera Views*, In Proc. of the Int. Conference on Computer Vision, Vol. 1, pp. 415-425, 1999.

[Kutulakos00] K. Kutulakos and S. Seitz, *A theory of shape by space carving*. In Int. Journal of Computer Vision, 38(3), pp. 198-218, 2000.

[Laurentini94] A. Laurentini, *The Visual Hull Concept for Silhouette-Based Image Understanding*. In IEEE Transactions on Pattern Analysis and Machine Intelligence, 16(2), pp. 150-162, February 1994.

[Lok01] B. Lok, *Online Model Reconstruction for Interactive Virtual Environments*. In Proc. of Symposium on Interactive 3D graphics, 2001.

[Sainz02] M. Sainz, N. Bagherzadeh and A. Susin, *Recovering 3D Metric Structure and Motion from Multiple Uncalibrated Cameras*. In Proc. of IEEE Conference on Information Technology: Coding and Computing, pp. 268-273, 2002.

[Seitz99] S. Seitz and C. Dyer, *Photorealistic scene reconstruction by voxel coloring.* In Int. Journal of Computer Vision, 35(2), pp. 1067-1073, 1999.

[Slabaugh01] G. Slabaugh, B. Culbertson, T. Malzbender and R. Schafer, *A Survey of Methods for Volumetric Scene Reconstruction from Photographs*. In Proc. of Int. Workshop on Volume Graphics, pp. 81-100, 2001.

[Steinbach00] E. Steinbach, B. Girod, P. Eisert and A. Betz, *3-D reconstruction of Using Spatially Extended Voxels and Multi-Hypothesis Voxel Coloring*. In Proc. of Int. Conference on Pattern Recognition, Vol. 1, pp. 774-777, 2000.

[Wood00] D. Wood, D. Azuma, K. Aldinger, B. Curless, T. Duchamp, D. Salesin and W. Stuetzle, *Surface Light Fields for 3D Photography.* In Proc. of SIGGRAPH'00, pp. 287-386, 2000.