Incremental Visual Hull Reconstruction

Ali Bigdelou
http://campar.in.tum.de/Main/AliBigdelou
Alexander Ladikos
http://campar.in.tum.de/Main/AlexanderLadikos
Nassir Navab
http://campar.in.tum.de/Main/NassirNavab

Chair for Computer Aided Medical Procedures
Technische Universität München
Boltzmannstr. 3, 85748 Garching, Germany

Abstract

We propose a novel algorithm for incrementally reconstructing the visual hull in dynamic scenes by exploiting temporal consistency. Using the difference in the silhouette images between two frames we can efficiently locate the scene parts that have to be updated using ray casting. By only concentrating on the parts of the scene that have to be checked for changes, we achieve a significant speed up over traditional methods, which reconstruct each frame independently. Our method does not use any kind of scene model and works regardless of the number and shape of the objects in the scene. We test our algorithm on sequences taken with a multi-camera system and perform a detailed performance analysis showing that our method outperforms other existing methods.

1 Introduction

Visual hull reconstruction is the process of generating three dimensional scene geometry from silhouette images captured from different views of a scene. The visual hull is the shape maximally consistent with its silhouette projection [10]. It has been popular due to its good approximating qualities for many objects and its ease and speed of implementation. It is commonly used as a starting point for more elaborate 3D reconstruction methods [14] and finds application in many real-time 3D reconstruction systems due to its good performance [9, 16]. However, current reconstruction systems do not take temporal consistency in the scene into account. Instead, they favor to highly parallelize the computation of the visual hull in order to obtain real-time performance. In contrast to these approaches, we propose an incremental voxel based visual hull reconstruction method for dynamic scenes using ray casting. By exploiting the fact that there usually occur only small changes between consecutive frames, we can reduce the runtime and calculation complexity in these environments. This allows us to perform real-time reconstruction on a standard processor without having to parallelize the computations.

In the remainder of the paper we give an overview of related work on visual hull reconstruction. Then we propose our new approaches followed by a mathematical analysis. Finally a comparison of different visual hull reconstruction algorithms including our incremental approach is presented.

BMVC 2009 doi:10.5244/C.23.35

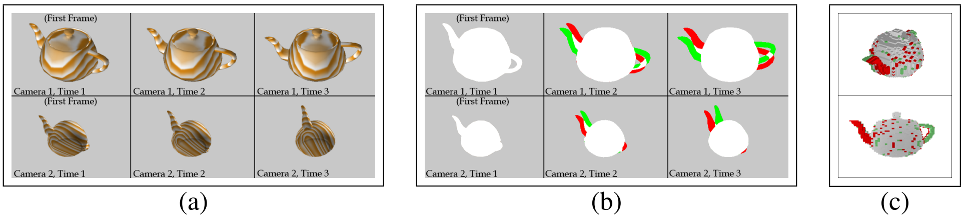


Figure 1: The rotating teapot model: (a) Rendered views from two different cameras in three successive time steps; (b) Corresponding silhouette and difference images with highlighted changed areas. Red pixels have been removed from the foreground since the previous time step and green pixels have been newly added in the current time step; (c) Colored recently removed and added voxels from two novel views.

2 Related Work

Most visual hull reconstruction methods use a volumetric approach [13, 15, 19]. In this approach the reconstruction volume is discretized into voxels, which are then projected into the silhouette images. If the area that the voxel projects into belongs to the foreground in all silhouette images then the voxel is marked as occupied otherwise it is marked as empty. One common approach for speeding this approach up is to use an octree representation [13, 19]. There also exist other approaches for visual hull reconstruction that do not use a volumetric representation of space. These are generally refered to as surface-based approaches [5, 8, 11].

Although there are methods for performing time consistent reconstructions in the context of multi-view reconstruction [17, 20], there is little work dealing with temporal visual hull reconstruction. Cheung [6, 7] consider the problem of aligning multipe silhouette images of a non-rigidly moving object over time in order to improve the quality of the visual hull. However, this is quite different from our approach. More recently Aganj [1] proposed a method for spatio-temporally consistent visual hull reconstruction by performing a 4D reconstruction over the whole sequence using Delaunay meshing. The drawback of their method is that it only works offline, requiring the whole sequence to be available for performing the reconstruction. This precludes a real-time online use of their method. In addition the runtime of their method is far from real-time due to its high computational complexity.

Our approach does not make any assumptions about the number or the shape of the objects in the scene and does not use any additional information other than that provided by the silhouette images. Our results are identical to the results one would obtain when using a normal reconstruction method for each frame but the computation time is considerably reduced leading to real-time performance on an off-the-shelf computer.

3 Incremental Reconstruction Approach

In dynamic scenes such as those captured by multi-camera systems the changes between frames are limited by the speed at which the objects in the scene move or deform. Hence, the number of voxels that change in the reconstruction does not alter dramatically. Therefore it is inefficient to reconstruct each frame independently. Theoretically the only voxels that have to be updated to transform the previous reconstruction into the current one, are the ones with changed occupancy. In the following we will call this subset of voxels the *Changed Set*. In practice we normally cannot compute the *Changed Set* directly. Therefore, we approximate it by a superset, which we call the *Search Space*.

In this work we propose a *Search Space*, whose size is close to the *Changed Set*. The main idea is to consider the changes in the silhouette images between the previous and the current frame, since they are caused by the change in voxel occupancy. Using ray casting [2] we efficiently find these changed voxels. Since most of the search process is done in two dimensional image space, the required amount of operations is reduced noticeably compared to performing a full reconstruction. Figure 1 illustrates our method for three frames of a rotating teapot. Figure 1(b) shows the change in the silhouettes (green being newly added foreground pixels and red being the removed foreground pixels), while Figure 1(c) shows the added (green) and removed (red) voxels between the frames. It is clear that the number of changed voxels is small compared to the total number of voxels.

The visual hull in the first frames can be reconstructed using any existing reconstruction method. The first step for updating the reconstruction is to compute the difference images. This allows us to determine which pixels were added or removed from the foreground. We then find the corresponding voxels by traversing the rays starting from the camera center and passing through the changed pixels.

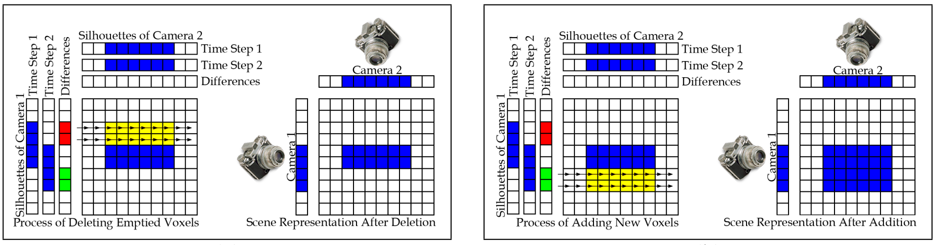
However, there usually is not a one-to-one relation between voxels and pixels due to different resolutions and the effects of perspective projection. Normally the voxel resolution is chosen to be smaller than the pixel resolution; hence the projection of most voxels extends to more than one pixel. In this situation the occupancy of each voxel may be checked multiple times, when using ray casting. Despite that, this redundancy can be reduced considerably by using an appropriate sub-sampling depending on the scene configuration and resolution. In other words, the changed state can be computed for a subset of pixels, achieved by sub-sampling silhouette images, to obtain better correspondence between voxel and difference images pixel resolutions.

The information about which pixels were added and removed is used to find the set of removed and added voxels. While the set of voxels corresponding to the removed pixels only have to be set to empty, the set of voxels corresponding to the added pixels have to be checked for occupancy. This is done using the original silhouette images. Since we only change the traversing process in the voxel space, the reconstructed visual hull produced by our approach is equivalent to the ones produced by other visual hull reconstruction methods. The implementation of the deletion and addition phase depends on the algorithm used. We have developed two different algorithms based on this approach, which we will discuss in the following two sections.

4 Ray Casting

Here, we explain our first algorithm for incremental visual hull reconstruction in dynamic scenes, which uses ray casting to update the visual hull. For each changed pixel in the difference images, we create a ray from the optical center of the camera passing through the pixel and traverse all voxels, which the ray passes, using ray casting [2]. The size of the search space in this implementation is equal to the number of voxels projecting into the changed regions of the difference images.

All changes after the first frame can be reconstructed by an updating phase using this approach. Even in the first frame, the whole visual hull can be computed using blank silhou-


Silhouettes of Camera 2
Time Step 1
Time Step 2
Differences
Silhouettes of Camera 1
Camera 2
Camera 1
Process of Deleting Emptied Voxels
Scene Representation After Deletion
Process of Adding New Voxels
Scene Representation After Addition

(a) (b)

Figure 2: The voxel traversing process: (a) Rays from removed pixels. Yellow voxels are removed; (b) Rays from new pixels. Yellow voxels are added.

Algorithm 1 Pseudo code for the deletion phase in the ray casting method

```
1: for each image i in the sub-sampled difference images list do
2:     for each removed pixel p in image i do
3:         Create ray r from optical center passing through p
4:         Set r to the first voxel in the reconstruction volume
5:         repeat
6:             Set occupancy state of the voxel at r to empty
7:         until step r to the next occupied voxel is unsuccessful
8:     end for
9: end for
```

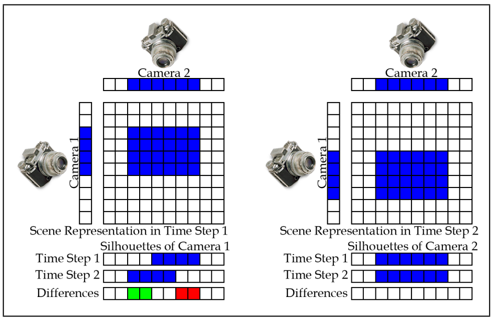
ettes as the images from the previous time step. But the computation using this approach, for the whole scene reconstruction, is redundant and the occupancy of some voxels might be checked more than once.

In the deletion phase, the occupancy of all voxels lying on rays passing through removed pixels (red in the images such as figure 1) is set to empty. This is because these areas are the projection of removed voxels since the previous time step. This is illustrated in figure 2(a). Algorithm 1 shows the pseudo code for the deletion process.

Algorithm 2 Pseudo code for the addition phase in the ray casting method

```
1: for each image i in the sub-sampled difference images list do
2:     for each added pixel p in image i do
3:         Create ray r from the optical center passing through p
4:         Set r to the first voxel in the reconstruction volume
5:         repeat
6:             if voxel v at r is occupied then
7:                 Mark v as occupied
8:             end if
9:         until step r to next empty voxel is unsuccessful
10:    end for
11: end for
```

To complete the updating process, after removing deleted voxels, newly occupied voxels are added in the addition phase. Figure 2(b) and the pseudo code given in algorithm 2 show the process of creating newly occupied voxels in the reconstruction results of the previous



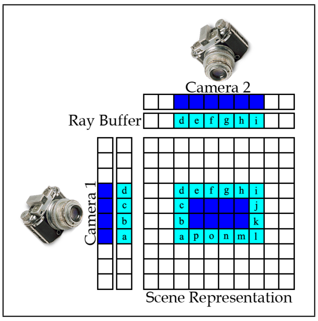


Figure 3: Two consecutive time steps of a dynamic 2D scene viewed by two cameras: (a) Silhouettes and calculated difference images for each camera; (b) Ray buffer, storing information about surface voxels.

phase. The addition phase of this method is similar to the deletion phase. However, before creating new voxels, their occupancy has to be checked. A voxel is marked as occupied if its projection on all silhouette images, maps to a foreground pixel.

Since ray casting is completely implementable on a GPU [12, 18], this algorithm also can be implemented on modern graphical processors.

5 Ray Buffers

Although the ray casting approach described in the previous section is already very efficient, we can further improve the performance by storing more data from the previous time step. To this end we store per pixel ray information as described in [3, 4] and shown in figure 3 for each time step. We save the basic ray information such as starting point, direction and current voxel for later use in successive time steps. We call this stored information ray buffers. This information helps us to define an even smaller search space.

This data has to be taken into account during both the deletion and the addition phase of the algorithm and updated, whenever any change happens on the surface voxels. In ray buffers, we only store information for surface voxels for reducing the discussed per voxel occupancy checking redundancy. Although by using this method we check fewer voxels, it is not always faster than the ray casting approach due to the extra processing for updating the ray buffers.

The deletion phase is similar to the ray casting method, however now we directly find the first surface voxel which should be removed using the information in the ray buffers. In large scenes this reduces the search space size and consequently the amount of required computations for finding the first visible surface voxel noticeably.

On the other hand, in the adding phase, we cannot directly find new voxels using the available data in the ray buffers, because we only store information about the current surface voxels. However, for improving performance, after finding the first new voxel - using the same approach as in the addition phase of the ray casting method - we obtain and update the neighboring new voxels using a 3D region growing method in voxel space, starting on the newly added voxel. Then, by projecting, we mask out all corresponding changed pixels to recently added surface voxels from all difference images. This helps us to avoid further redundant processing.

6 Analysis

Our proposed approaches update the scene based on the changed areas in the images, so each change in the visual hull of a model should be reflected in at least one of the silhouette images used for reconstruction. In this section we show the correctness of this assumption through detailed mathematical analysis.

As described before, the scene volume is divided into voxels. We define the set V such that it contains all voxels in the scene. We will refer to each voxel with v. For constructing the visual hull, we use n calibrated cameras viewing the scene. The set C contains all cameras and we will refer to each individual camera in this set using c_1, c_2, ..., c_n. Each of these cameras has an associated projection matrix P_{c_i}. $P_{c_i}(U)$ is the 2D projection of a 3D point U using camera c_i. Silhouette images for each of these cameras are referred to using I_{c_i}. The value in the silhouette image at location u is given by $I_{c_i}(u)$. In the silhouette images pixels can have two distinct values, foreground and background. We refer to foreground with value one and to background with value zero. Extending these notations for dynamic scenes requires adding a time step index. For example for referring to the camera image c_i in time step t we use $I^t_{c_i}$ and so on.

Using this notation, we will represent the visual hull reconstruction process. The reconstructed visual hull of a scene is defined as $D = \left\{ v \in V | \underset{c\in C}{\forall} (I_c(P_c(v)) = 1) \right\}$, which is a subset of V including only occupied scene voxels. D contains all voxels, whose projections are mapped to foreground pixels in all images. The state of a voxel can change between two time steps t and $t+1$ (*Changed Set*). Unchanged voxels are either empty or occupied at both times, while changed voxels are occupied at t and empty at $t+1$ or vice versa.

For creating the voxel representation of the scene at time $t+1$ it is sufficient to only invert the state of the voxels which are in the *Changed Set*. Formally the *Changed Set* is given by:

Lemma 1: The set of voxels S, defined as

$$S = \left\{ v|(v \in D^t) \wedge \underset{c\in C}{\exists} (I^{t+1}_c(P_c(v)) = 0)) \right\} \bigcup \left\{ v|(v \notin D^t) \wedge \underset{c\in C}{\forall} (I^{t+1}_c(P_c(v)) = 1) \right\}$$

contains all voxels in the changed set.

Proof: The equation defining the set S is the union of two sets. The left is the set of voxels occupied at time t ($v \in D^t$) and empty at time $t+1$. The second term on the left ($\underset{c\in C}{\exists} (I^t_c + 1(P_c(v)) = 0)$) is based on the observation that there should be at least one camera in which the projection of v at time $t+1$ is zero if this voxel is actually empty. The right part adds the set of voxels, which are empty at time t ($v \notin D^t$) and occupied at time $t+1$. The second term on the right ($\underset{c\in C}{\forall} (I^{t+1}_c(P_c(v)) = 1)$) is the set of occupied voxels, since by definition their projection on the images of all cameras at time $t+1$ is one. □

Lemma 1 gives us a clear definition for the optimum set of voxels, which only contains the changed voxels at time $t+1$ with respect to time t. However, computing the subset S itself is not possible directly. Therefore, we propose a superset, which approximates S closely.

Definition: We define two new binary images I^{RV}_c, I^{AV}_c for any given camera at times t and $t+1$ and compute their corresponding values at the location u as described by the following formulas:

$$I^{RV}_c(u) = I^t_c(u) \wedge \neg I^{t+1}_c(u) \text{ , } I^{AV}_c(u) = \neg I^t_c(u) \wedge I^{t+1}_c(u)$$

I_c^{RV} represents the projection of the removed voxels, whose disappearance causes differences between I_c^t and I_c^{t+1} and I_c^{AV} represents the projection of the added voxels, whose appearance causes differences between I_c^t and I_c^{t+1}. In figure 3(a) the red and green colored areas are showing I_c^{RV} and I_c^{AV} respectively.

Next we will show that if any change happens in the scene, which can be captured by the visual hull, we will see it in the difference images. The next theorem proves that our proposed search space is a superset for the *Changed Set*.

Theorem 2: The subset E, which is defined as described below, contains all changed voxels between times t and $t+1$:

$$E = \left\{ \bigcup_{c\in C} \{v | I_c^{RV}(P_c(v)) = 1\} \right\} \bigcup \left\{ \bigcup_{c\in C} \left\{ v | I_c^{AV}(P_c(v)) = 1 \right\} \right\}$$

Proof: Again, the above equation contains the union of two sets. The left part is denoting all voxels for which there exist at least one camera with value one at the voxel projection coordinate on the corresponding image I^{RV} of that camera. Similarly on the right side of the union operator, we have a subset of voxels, denoting all voxels for which there exist at least one camera with value one at the voxel projection coordinate on the corresponding image I^{AV} of that camera. By rewriting the above equation we obtain:

$$E = \left\{ v | \underset{c\in C}{\exists} (I_c^{RV}(P_c(v)) = 1) \right\} \bigcup \left\{ v | \underset{c\in C}{\exists} \left(I_c^{AV}(P_c(v)) = 1 \right) \right\}$$

By replacing I_c^{RV}, I_c^{AV} from the above definitions we get:

$$E = \left\{ v | \underset{c\in C}{\exists} \left(I_c^t(P_c(v)) = 1 \wedge I_c^{t+1}(P_c(v)) = 0 \right) \right\} \cup \left\{ v | \underset{c\in C}{\exists} \left(I_c^t(P_c(v)) = 0 \wedge I_c^{t+1}(P_c(v)) = 1 \right) \right\} \supseteq$$

$$\left\{ v | \underset{c\in C}{\forall} \left(I_c^t(P_c(v)) = 1 \right) \wedge \underset{c\in C}{\exists} \left(I_c^{t+1}(P_c(v)) = 0 \right) \right\} \cup \left\{ v | \underset{c\in C}{\exists} \left(I_c^t(P_c(v)) = 0 \right) \wedge \underset{c\in C}{\forall} \left(I_c^{t+1}(P_c(v)) = 1 \right) \right\} =$$

$$\left\{ v | (v \in D^t) \wedge \underset{c\in C}{\exists} \left(I_c^{t+1}(P_c(v)) = 0 \right) \right\} \cup \left\{ v | (v \notin D^t) \wedge \underset{c\in C}{\forall} \left(I_c^{t+1} P_c(v)) = 1 \right) \right\} = S$$

For reaching the third line, we have used the fact that if a voxel can be seen from all cameras at time t, ($\underset{c\in C}{\forall} (I_c^t(P_c(v)) = 1)$), it is part of the reconstructed volume at time t, which gives $v \in D^t$. Similarly, if there exists at least one camera from which the voxel can not be seen ($\underset{c\in C}{\exists} (I_c^t(P_c(v)) = 0)$) at time t, that voxel is empty at time t ($v \notin D^t$). Using the result of lemma 1 and the fact that $S \subseteq E$ the correctness of this theorem can be stated. $\square$

Conclusion 1: For any newly removed voxel from the scene at time $t+1$, which alters the visual hull, there should be at least one I^{RV} image, that reflects this change.

Conclusion 2: For any newly added voxel into the scene at time $t+1$, which alters the visual hull, there should be at least one I^{AV} image, that reflects the change.

This theorem helps us to define a new search space E, whose size is close to the real changed set S as described in lemma 1. Moreover, because I^{AV} and I^{RV} are made up from the projection of changed portions of the scene we get a small 2D area to consider in each image. In other words, the size of the search space is directly related to the size of the changed portion of the scene; for instance if the scene is static, the search space will be empty because I^{AV} and I^{RV} are equal to zero.

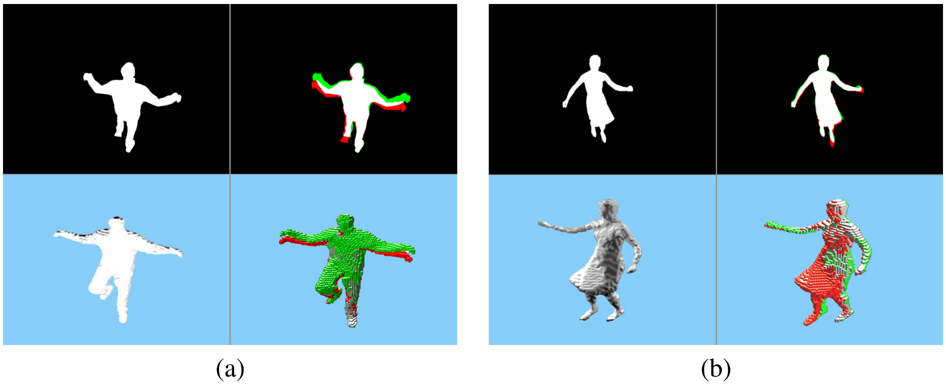


Figure 4: The silhouette and difference images and the reconstructed volume rendered from a novel view: (a) Crane dataset; (b) Samba dataset.

7 Results

We have tested our proposed incremental reconstruction approaches as well as previous methods on both synthetic and publicly available real data sets (provided by Daniel Vlasic [21], see figure 4), comparing them on different aspects such as run time and the number of occupancy checks with various voxel and image resolutions. These results show the significantly improved performance of our incremental approaches. We achieve a speed-up of up to 10 times over the other reconstruction methods.

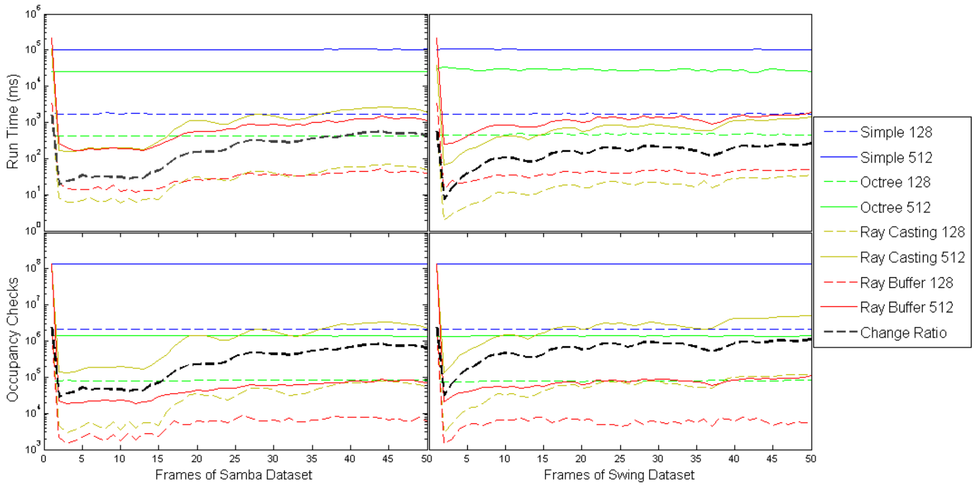


Figure 5: Runtime and occupancy check count in the first fifty frames of the Samba (left) and Swing (right) dataset at two different voxel resolutions. Our incremental approach outperforms the other methods in both categories.

Figure 5 shows the results we obtained on two real data sets with respect to runtime and number of checked voxels at different volume resolutions. Since the size of the search space of our approach is directly related to the size of the changed portion of the model, as

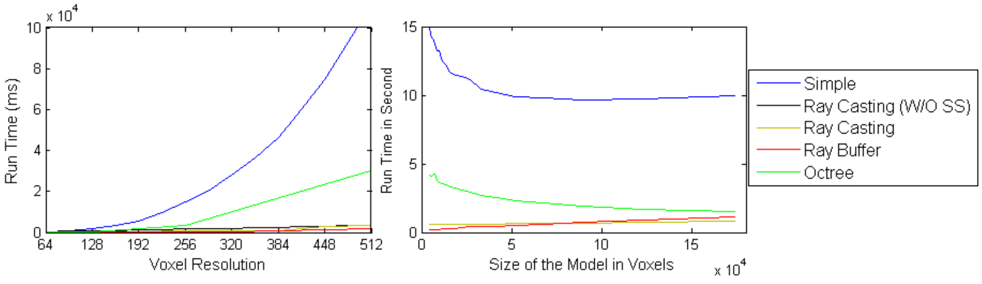


Figure 6: Dependency of the different algorithms on the voxel resolution (left) and the model size (right) using frame 27 of the Crane dataset.

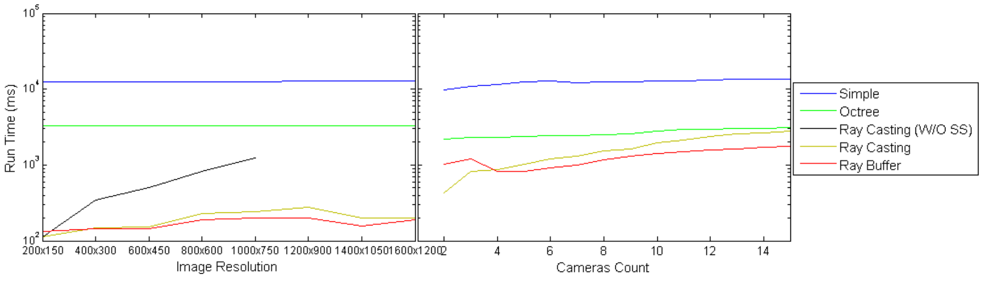


Figure 7: The left diagram shows the effect of using different image resolutions (frame 11 of the Bouncing dataset). The right diagram shows the performance with different numbers of cameras (frame 5 of the Teapot dataset). The volume resolution used was 256^3.

expected, both the number of occupancy checks and the runtime are related to these changes. For showing this, we computed a change ratio value for each model by dividing the number of changed voxels by the total number of voxels in the model. A rescaled version of these change ratios is shown in figure 5 with a black dashed line. Our ray casting and ray buffer method are consistently faster than the octree-based and the direct volumetric reconstruction approach, while also considering less voxels. This is expected, since we consider only the changed parts of the scene in our computations.

Unlike previous reconstruction methods, whose time and computational complexity are strongly related to the voxel resolution, the proposed dynamic approaches have less direct dependency on the voxel resolution. However, changing the voxel resolution affects the optimum sub sampling, which itself has influence on overall performance. Figure 6 compares runtimes with various voxel resolutions and model sizes. The model size is increased by decreasing the size of the reconstruction volume. Note that having larger models in the scene decreases the run time of the ray buffer approach due to the larger number of surface voxels, which have to be processed.

On the other hand, as shown in figure 7, image resolution and the number of cameras have little impact on the performance of the previous reconstruction methods, but it can affect our approaches noticeably, because of their 2D image based search space. Generally, without sub-sampling the performance of these algorithms is directly related to the number of cameras and the image resolution. However, this effect can be reduced by using sub sampling in the images.

8 Conclusion

We proposed a new incremental visual hull reconstruction approach for time varying scenes. We use the difference in the silhouette images between two frames to efficiently find potentially changed voxels in the scene. We suggested two ray-casting-based algorithms for this, one of which stores additional per-frame information for speeding up the computations. Furthermore, we showed the validity of our method using a mathematical analysis and tested them against several datasets. We demonstrated that our incremental method significantly improves the performance of the reconstruction process, outperforming existing reconstruction methods and making it suitable for real-time applications.

References

[1] E. Aganj, J. Pons, F. Ségonne, and R. Keriven. Spatio-temporal shape from silhouette using four-dimensional delaunay meshing. In *ICCV*, pages 1–8. IEEE, 2007.

[2] J. Amanatides and A. Woo. A fast voxel traversal algorithm for ray tracing. In *Eurographics '87, Amsterdam, North-Holland*, 1987.

[3] O. Batchelor and R. Mukundan. Ray traversal for incremental voxel coloring, 2006. M.Sc Thesis.

[4] O. Batchelor, R. Mukundan, and R. Green. Ray casting for incremental voxel colouring. In *Image and Vision Computing Conference - IVCNZ05 NewZealand*, 2005.

[5] E. Boyer and J. Franco. A hybrid approach for computing visual hulls of complex objects. In *IEEE CVPR*, 2003.

[6] K. Cheung, S. Baker, and T. Kanade. Shape-from-silhouette across time part i: Theory and algorithms. *IJCV*, 62(1):221 – 247, May 2005.

[7] K. Cheung, S. Baker, and T. Kanade. Shape-from-silhouette across time: Part ii: Applications to human modeling and markerless motion tracking. *IJCV*, 63(1):225 – 245, August 2005.

[8] J. Franco and E. Boyer. Exact polyhedral visual hulls. In *BMVC*, 2003.

[9] A. Ladikos, S. Benhimane, and N. Navab. Efficient visual hull computation for real-time 3d reconstruction using cuda. In *Proceedings of the 2008 Conference on Computer Vision and Pattern Recognition Workshops*, 2008.

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

[11] S. Lazebnik, Y. Furukawa, and J. Ponce. Projective visual hulls. *IJCV*, 74(2):137–165, 2007.

[12] C. Müller, M. Strengert, and T. Ertl. Optimized volume raycasting for graphics-hardware-based cluster systems. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV'06)*, 2006.

[13] M. Potmesil. Generating octree models of 3d objects from their silhouettes in a sequence of images. *Computer Vision, Graphics and Image Processing*, 40(1):1–20, 1987.

[14] S. Seitz, B. Curles, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *IEEE CVPR*, 2006.

[15] D. Snow, P. Viola, and R. Zabih. Exact voxel occupancy with graph-cuts. In *IEEE CVPR*, 2000.

[16] L. Soares, C. Ménier, R. Raffin, and J. Roch. Parallel adaptive octree carving for real-time 3d modeling. In *IEEE Virtual Reality*, 2007.

[17] J. Starck and A. Hilton. Correspondence labelling for wide-timeframe free-form surface matching. In *ICCV*, pages 1–8. IEEE, 2007.

[18] M. Strengert, T. Klein, R. Botchen, S. Stegmaier, M. Chen, , and T. Ertl. Volume surface octrees for the representation of 3d objects. *The Visual Computer*, 22(8):550–561, 2006.

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

[20] K. Varanasi, A. Zaharescu, E. Boyer, and R. Horaud. Temporal surface tracking using mesh evolution. In *Proceedings of the Tenth European Conference on Computer Vision*, volume Part II of *LNCS*, pages 30–43, Marseille, France, October 2008. Springer-Verlag.

[21] D. Vlasic, I. Baran, W. Matusik, and J. Popović. Articulated mesh animation from multi-view silhouettes. *ACM Transactions on Graphics*, 27(3):97:1–97:9, 2008.