T U M

I N S T I T U T F Ü R I N F O R M A T I K

Shape from Silhouette in Dynamic Scenes

Ali Bigdelou, Alexander Ladikos, Nassir Navab

TUM-I0826
August 08

T E C H N I S C H E U N I V E R S I T Ä T M Ü N C H E N

Shape from Silhouette in Dynamic Scenes

Technical Report
Technical University of Munich, Germany

Ali Bigdelou
ali.bigdelou@khayam.ut.ac.ir

Alexander Ladikos
ladikos@in.tum.de

Nassir Navab
navab@cs.tum.edu

Abstract

The volumetric reconstruction of a scene, from multiple views is one of the interesting problems in computer vision. Currently there is a lack of proper methods for reconstructing visual hull in time varying scenes and most of the available techniques are only concentrated on the static scenes. In this work we have developed new methods for reconstructing the visual hull of a dynamic scene incrementally. For each camera, our method processes the image differences in two consequent time steps and uses the ray casting for updating changes in the last reconstructed visual hull.

Introduction

The visual hull reconstruction is the process of generating, three dimensional scene geometry, from silhouette images captured from different views in the scene which also is known as Shape-from-Silhouette. There are many usages for visual hull reconstruction such as motion tracking and capturing, industrial assembly and inspection, robot obstacle detection and collision avoidance, path planning, autonomous vehicle navigation of unfamiliar environments and many other machine vision problems. With the recent advances of digital cameras and imaging technology, even more considerable attention has been focused on these methods.

In this work we have concentrated on the incremental voxel based visual hull reconstruction of dynamic scenes. Currently there is a lack of proper researches on the reconstruction of time varying scenes. We have developed a new method for incrementally reconstructing the visual hull of dynamic scenes using ray casting. We aim to reduce the runtime and calculation complexity in these environments. We have done this, using the fact that in dynamic scenes, in most cases – except for the fast motions – only small portion of the reconstructed volume changes, between two consequent time steps. Additionally, the use of modern digital cameras, which can capture the scene with high frame rate, will guarantee small changes between two successive time steps.

Although we can use the same manner, which is used for reconstructing static scenes, for consequent time steps separately, it is not clever to lose data gathered in the previous time step. In our algorithm we have tried to keep as much as possible data, from previous time and only change the time varying sections. Using this approach we have shown that runtime and search space of the problem reduce noticeably.

In this report, we first introduce some basic definition that will be used throughout this report. Moreover a literature review of existing research on the image based visual hull reconstruction problems is presented. Then we propose our new approaches for reconstruction in dynamic scenes followed by the mathematical proof of our approach. And finally a comparison of different visual hull reconstruction algorithms including our incremental approach and a formulated comparison with existing methods are presented in the analysis section.

Volume reconstruction

Reconstructing a three dimensional model is one of the basic challenges in computer vision and there are many different usage for such a reconstructed model. In the last decades, for solving this problem in different situation, many methods have been presented.

Volume reconstruction methods traditionally have been based on image matching techniques, using either intensity-based (direct) or feature-based approaches. All multi-view stereo techniques are member of this class. In multi-view stereo reconstruction, first the correspondence across stereo images is computed and then 3D structure of the volume is reconstructed by triangulation and surface fitting. These approaches have some disadvantages as categorized below:

- Captured views often must have small baseline which means they should be close together for improving effectiveness of correspondence techniques.
- Correspondence data must be calculated and stored from many different views for spanning large changes in the viewpoint.
- During reconstruction process, many partial models should be computed and then these patches must be fused into a single model.
- Using sparse feature matching which reduces runtime, will force us to use a parameterized surface model, fitted to extracted points, for recovering final dense surface.
- There is no explicit handling of occlusion differences between views which may cause incorrect output result.

An alternative approach for volume reconstruction is based on computation in 3D space for constructing the geometries which are consistent with input images. Methods based on this approach are called volume reconstruction. Using this technique, the search space changes to three dimensional spaces instead of image base search and this avoids the above problems.

Volumetric reconstruction methods assume the existence of a bounding region for the scene that includes the entire model which should be reconstructed. Normally this region is a box, surrounding the scene. This bounding region then will be segmented into discrete volume elements which also often are in cube shape, called voxel.

There exist other presentations for voxels such as octrees [3, 4, 5]. Using octrees the searching process is more efficient. In this technique, for getting higher resolution, each voxel is divided into eight inner sub voxel if it is occupied, starting with low resolution voxelization. Disparity space representation [6, 7, 8], is other technique which define the segments based on one camera's viewpoint. The voxels are referenced with their projection coordinate in the image and the distance from camera eye position as *(x, y, d)*. Furthermore they have not same size and shape in this method and become smaller, moving toward camera position. For modeling the visibility with respect to a particular camera, generalized disparity space [9] is used *(x, y, d, k)*, *k* is the index of the target camera. If we have fundamental matrices between all pairs of camera instead of the projection matrix, a projective space [6, 10] can be defined using two of the cameras as the basis views. Recent review of these alternate methods can be found in [11]. A survey on volumetric reconstruction approaches [12, 18, 19, 20] has been written too.

Visual hull reconstruction

The state of each voxel can be expressed as occupied or empty for visual hull reconstructions, and for colored reconstruction each voxel can have either one of three, transparent, opaque or unseen values plus its color. Voxel reconstruction can be computed using one of the shape from silhouette or shape from color consistency methods. In this work we have developed a new shape from silhouette approach for time varying models.

Silhouette is a binary image captured from a camera's viewpoint, describing that if the value at any given pixel is representation of a foreground object (silhouette) or background scene. By intersecting silhouette images, an approximated volumetric model of the 3D object, known as visual hull, can be reconstructed. Each voxel, which projection is mapped on a foreground pixel, in all silhouette images, should be marked as occupied and all other voxels should be marked as empty.

For developing an algorithm to compute visual hull, all voxels in the set should be traversed. Beside that for each voxel, the projection coordinate of that voxel should be computed per camera. This is the simplest possible implementation for visual hull reconstruction. The time complexity of this method directly depends on the voxel resolution and number of used cameras. If we divide the volume of the scene to *N* voxel per axis and we use *M* camera for capturing the scene, the computational complexity of this algorithm is $O\left(N^3 \times M\right)$. This means that, it is not efficient to reconstruct a scene with large number of voxels and cameras using this technique.

Visual hull gives us a conservative estimate of the shape placed in the foreground and it can be used as a start point for other algorithms [14, 15]. In [16, 3] silhouette images are used for reconstruction with described intersection approach. An example of stereo images together used with silhouette for high quality reconstructing is presented in [17, 21, 22, 23].

Extension for dynamic scenes

By dynamic scene, we mean, a time varying model of the scene which data expand along defined time steps. There should be a meaningful relation between two consequent time steps which means the objects in the scene should not change their state so fast that their relation between two successive time steps can not be evaluated. In other word, there should be limited amount of changed parts in each time in respect to its previous time.

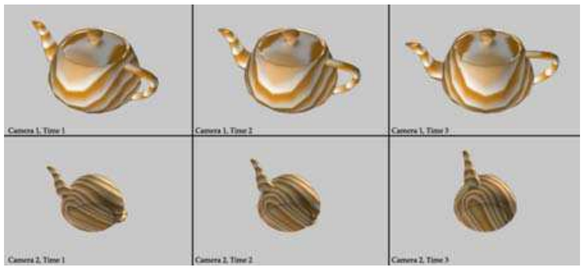

Figure 1: The rotating teapot model, which is rendered from two different cameras in three successive time steps.

With an acquisition frame rate of 25 fps, there is generally small inter frame displacement. Figure 1 shows a rotating teapot which is rendered in 3DSMAX in three consequent time steps from two different angles.

The visual hull reconstruction in a time varying scene can be done by repeatedly calling the method implemented as discussed in the previous section for each time step. But it is obvious that using this approach the occupancy state of all voxels in the scene should be recomputed in each time step. The most important inefficiency reason of this approach – used for dynamic scene reconstruction – is its non-optimized search space. Search space is the subset of voxels which should be examined in each time step during updating process. In the time varying scenes which is captured using new high frame rate digital cameras, there exist a small amount of movement and changes between two successive time steps. Moreover, especially the changes will be take place nearby the previously occupied voxels. So the search space size can be reduced using this similarity in time varying scenes.

Knowing about this fact, we can only process the voxels in the neighborhood areas of the previously reconstructed model. This approach will reduce the search space size and the number of processed voxels dramatically. However, this will limit the inter frame movement of the objects. The search space will be reduced to only adjacent voxels to the previously reconstructed model and this directly affects runtime.

For implementing this method we need to define a key frame concept which will happen whenever the algorithm does not have the reconstructed visual hull from previous time step or its data is not correct. First frame of a sequence is such a case, because we don't have reconstruction form previous step. Other situation that key frame technique, can be used, is whenever the reconstruction has not been computed successfully for previous time. This can happen when objects move more than we expect or new objects which are not in the fattened border of the previous time step come to the scene.

Main idea

As described before, each voxel in the reconstructed visual hull can have two distinct states, empty and occupied. Visual hull of a model is represented by a 3D array of voxels. In dynamic scenes, because of their nature, only limited amount of these voxels change their state in each time step. So theoretically the optimum search space contains those voxels with changed occupancy state and no other voxels. We call this set as **Changed Set**.

Obviously, in period of times when scene has small changes, small number of voxels are member of the **Changed Set** and when there isn't any movement in the scene this set will be empty. The dependency between size of the **Changed Set** and the movement in the scene has been shown in analysis section.

Having the **Changed Set**, we can easily convert the state of all including voxels, in the previous time's visual hull, for obtaining the current visual hull in the later step. But in reality, normally we can not compute the **Changed Set** directly. So we should predict it with a superset, which we call it as **Search Space**.

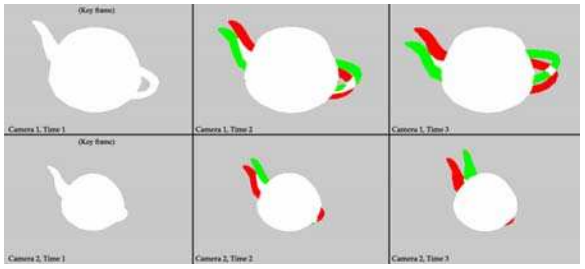

Figure 2: Silhouette images of rotating teapot – rendered images are shown in figure 1 – with highlighted changed areas. Red pixels are removed since previous time step and green pixels are newly added in current time step.

In this work we have found an optimal **Search Space** which size is really close to the **Changed Set**. The main idea of this work is to find changed pixels in successive images and trace them into the voxel space with ray casting [2]. Using this technique most of the search process should be done in the two dimensional image space and this will reduce the required amount of operations, noticeably. Figure 2 shows areas which have changed state – changed pixels – since previous time step.

Actually, for updating scene model in the current time, we look for the changed areas in each image since last time step and then we use ray casting technique to find all voxels in the 3D spaces which two dimensional projection are mapped to these changed areas. We have set up our search space by searching in two dimensional difference images for reducing search space dimensionally. This fact is shown in figure 3 in two dimensional image space.

We have implemented different algorithms, all working based on this idea. The key point of this idea is the fact, that if any changes which affect the geometry of visual hull take place in 3D environment, there should be a representation of this change at least on one of the 2D image planes used for reconstruction. We have proven this fact in theorem 2 which is described in the following section.

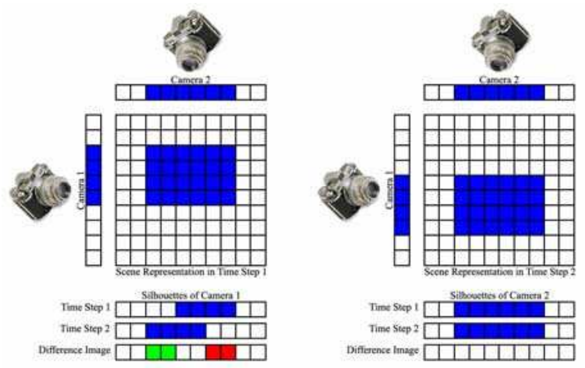

Figure 3: Two consequent time steps from a dynamic 2D scene viewed by two cameras and calculated difference images for each camera.

All different algorithms developed based on this idea, share some common required operations including starting up (key framing), computing difference images and updating model which itself contains two other main parts, deleting newly emptied and adding newly occupied voxels. So the following pseudo code can be thought as a skeletal structure for upcoming algorithms.

```
if this is a key frame
        reconstruct complete visual hull
else
{
        compute the difference images
        delete empty voxels
        add occupied voxels
        store current silhouettes as previous images
}
```

The key framing and its consequence reconstruction model have been discussed in the key framing section. First step for updating the reconstruction is to compute the difference images which are used later. This should compute the state of each pixel. Unchanged, newly occupied and recently emptied are the three states that a pixel can be marked to.

After computing state of the pixels, this data should be used for finding and removing the recently emptied voxels, since previous time step. Then difference images should be used for finding newly occupied voxels and creating those voxels. And finally silhouette images of current time step should be stored for using in the next time step.

The implementation for deletion and addition phases depends on the used algorithm. We have developed two different models of reconstruction, which have been discussed in the following sections, and for each we have presented appropriate implementations.

After finding changed pixels, all voxels for which the projection is mapped on changed pixels should be processed appropriately. We have used ray casting in voxel space [2] as an approach for traversing voxels using the rays created from camera's optical center and passing through corresponding changed pixels. As it is shown in figure 4, having a ray, this method gives us the ability to find, next adjacent voxel which has intersection with the ray. The following pseudo code traverses voxel space to find the next occupied voxel.

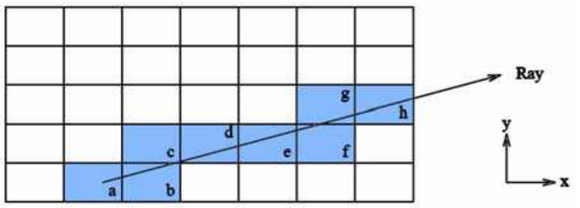

Figure 4: The ray which passes through a, b, c, d, e, f, g, h [2].

```
While (Step to next voxel is successful)
   if (current voxel is occupied)
      return true

return false
```

For achieving faster result, some voxels can be ignored in each step. Finding next empty voxel can be written in the same manner. Stepping to the next voxel is as described in reference work [2] which also gives complete definition and implementation on ray casting.

Key framing

Constructing using the described approach requires the reconstructed visual hull of previous step. But in some circumstances, we have not visual hull of previous time such as starting time steps or incorrect reconstruction in previous time. This may occur whenever the update phase of algorithm could not complete its work appropriately. This failure depends on update algorithms. For example, in fattened visual hull reconstruction the movement more than we've expected or the entrance of new objects into the scene can cause a reconstruction which is not valid.

In situations where we have not a valid reconstructed visual hull of the previous time, we have to do a full reconstruction. We call these time steps as key frames and the process of retrieving correct visual hull as key framing. For reconstruction in key frames we can use any of the methods which are used for reconstruction in static scenes. Key frames in are methods are the starting frames plus frames in which the update subroutine returns false in the previous time step. Moreover, we can set a new key frame with defined time ratio – for example every one second – for removing probable system faults.

Projection of changed areas

As expressed before, 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 at least in one of the silhouette images used for reconstruction. In this section we have shown the correctness of this assumption using mathematical notations.

As described before, in the volumetric reconstruction approaches, the scene volume is divided into discrete units, named voxels. We define the set V such that it contains all voxels in the scene. We will refer to each voxel with v.

For reconstructing the visual hull of a scene, we use n defined and calibrated cameras viewing the scene. The set C contains all such cameras and we will refer to each individual camera in this set using $c_1, c_2, ..., c_n$. Each of these cameras also has its own projection function named as P_{c_i} and $P_{c_i}(U)$ is the 2D coordinate of any given 3D point U, projected in to the image plane of camera c_i.

Silhouette images for each of these cameras are also referred to using I_{c_i}. Moreover the values in these silhouette images can be accessed using a 2D coordinate such u as $I_{c_i}(u)$. As described in the previous sections, in the silhouette images pixels can have two distinct values, foreground and background. We will refer to foreground with value one and to background with value zero.

For example, consider a case that we need to check the value in I_{c_1} at the projection of voxel v, to see if it is foreground. We can use $I_{c_1}(P_{c_1}(v)) = 1$ for getting required result.

Extending these notations for dynamic scene requires presence of a time step index. We will add this time related index on top of each previous notation. For example for referring to the camera image c_1 in time step t we use $I_{c_i}^t$ and so on.

Using the defined mathematical notation, now we can show the visual hull reconstruction process. The reconstructed visual hull of a scene can be shown using set D of voxels defined as $D = \{v \in V \mid \forall_{c \in C}(I_c(P_c(v)) = 1)\}$, which is a subset of V including only occupied scene voxels. So D should contain all voxels like v which projections are mapped to foreground pixels in all images.

State of any given voxel in any two consequent time steps, like t and $t+1$, can be categorized in two different groups as unchanged or changed voxels (**Changed Set**). The voxels in the former group either are not occupied in both times or occupied in both times and the voxels in the later group either are occupied in first time and not in second or vice versa.

The following theorem will give us an approach for updating the reconstructed visual hull in time t, to match the visual hull of the scene in time $t+1$.

Theorem 1: For creating the voxel representation of the scene in time $t+1$, it is enough to only invert the state of the voxels which are in the **Changed Set**.

Proof: By the definition of **Changed Set**, it contains all the voxels which have different occupied state in time $t+1$ with respect to time t and by changing their state the previously visible voxels turn to invisible and previously invisible voxels become to visible, making voxel representation of the scene in time $t+1$. □

Having previous theorem, now we can concentrate our works on finding the subset of voxels which contains the changed set. Such a subset can be defined as expressed in the following lemma.

Lemma 1: Subset of voxels D, defined as

$$D = \{v \mid (v \in V^t) \wedge \exists_{c \in C}(I_c^{t+1}(P_c(v)) = 0)\} \cup$$
$$\{v \mid (v \notin V^t) \wedge \forall_{c \in C}(I_c^{t+1}(P_c(v)) = 1)\}$$

contains all voxels in the changed set.

Proof: The equation defining the set D is the union of two sets. The left part is describing all voxels which are visible in time t, $(v \in V^t)$ but turned to invisible in time $t+1$, $\exists_{c \in C}(I_c^{t+1}(P_c(v)) = 0)$, which means, there should be at least one camera in which the projection of v on its image in time $t+1$ has value equal to zero, otherwise v should be visible in time $t+1$ too. Additionally the right part adds the other section of the changed voxels group to D which are invisible voxels in time t, $(v \notin V^t)$ that are visible in time $t+1$, $\forall_{c \in C}(I_c^{t+1}(P_c(v)) = 1)$, so their projection on the images of all cameras in time $t+1$ should have value equal to one.□

Lemma 1, gives us a clear definition for the optimum subset of voxels which only contains, changed voxels (changed set) in time $t+1$ in respect to voxels of time t. Having this subset and inverting the state of those voxels as described in theorem 1, voxels representation of time $t+1$ can be calculated.

However, computing subset D itself is not a trivial task and actually we still need to search all voxels in the scene. For making things faster, we should decrease our search space which currently contains all voxels.

Definition: For images of any given camera, in two consequent times like t and $t+1$, define two new binary images I_c^{RV}, I_c^{AV} and compute corresponding values at coordinate u as described in the formulas below:

$$I_c^{RV}(u) = I_c^t(u) \wedge \neg I_c^{t+1}(u),$$

$$I_c^{AV}(u) = \neg I_c^t(u) \wedge I_c^{t+1}(u)$$

The I_c^{RV} represents the projection of the removed voxels whose disappearance cause difference between I_c^t and I_c^{t+1}, and I_c^{AV} represents the projection of the added voxels whose appearance cause difference between I_c^t and I_c^{t+1}. Note that not all changed voxels have representation on all I_c^{RV} and I_c^{AV} images of all cameras. In the figure 2, the red and green colored areas are showing I_c^{RV} and I_c^{AV} images respectively. Using the next theorem, we have shown that our proposed search space is a superset for minimal search space.

Theorem 2: The subset E which is defined as described below contains all voxels whose appearance state are changed in time $t+1$, in respect of time t:

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

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

$$E = \left\{ v \mid \exists_{c \in C} \left(I_c^{RV}(P_c(v)) = 1 \right) \right\} \cup \left\{ v \mid \exists_{c \in C} \left(I_c^{AV}(P_c(v)) = 1 \right) \right\},$$

In which by replacing I_c^{RV}, I_c^{AV} from the above definition,

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

$$\left\{ v \mid \forall_{c \in C} \left(I_c^t(P_c(v)) = 1 \right) \wedge \exists_{c \in C} \left(I_c^{t+1}(P_c(v)) = 0 \right) \right\} \cup \left\{ v \mid \exists_{c \in C} \left(I_c^t(P_c(v)) = 0 \right) \wedge \forall_{c \in C} \left(I_c^{t+1}(P_c(v)) = 1 \right) \right\} =$$

$$\left\{ v \mid \left(v \in V^t \right) \wedge \exists_{c \in C} \left(I_c^{t+1}(P_c(v)) = 0 \right) \right\} \cup \left\{ v \mid \left(v \notin V^t \right) \wedge \forall_{c \in C} \left(I_c^{t+1}(P_c(v)) = 1 \right) \right\} = D$$

For reaching to the third line, we have used the fact that if a voxel can be seen from all cameras in time t, $\forall_{c \in C}\left(I_c^t(P_c(v)) = 1\right)$ actually it is part of the reconstructed volume in time t which gives $v \in V^t$ and also if there exist at least one camera from which the voxel can not be seen $\exists_{c \in C}\left(I_c^t(P_c(v)) = 0\right)$ in time t, in fact that voxel is not occupied in the scene in time $t, v \notin V^t$. Using the result from lemma 3.1 and the face that $D \subseteq E$, correctness of this theorem can be stated.□

Conclusion 1: For any newly removed voxel from the scene in time $t+1$, which alter the visual hull, at least one of the I^{RV} images exists, which reflect the change.

Conclusion 2: For any newly added voxel into the scene in time $t+1$, which alter the visual hull, at least one of the I^{AV} images exist, which reflect the change.

Theorem 2 helps us to define new search space E which is optimal and really close to the real changed set D as described in lemma 1. Moreover, because I^{AV} and I^{RV} images are made up from the projection of changed portion of the scene we have really limited area to search on each image. In other word, size of the search space is directly related to size of changed portion of the scene; for instance if the scene be static, search space will be empty because all I^{AV} and I^{RV} images are equal to zero.

Ray casting

Here, we explain our first algorithm for incremental visual hull reconstruction in dynamic scenes which uses the ray casting to update geometry of occupied voxels in each time. For each image pixel – in difference images – with changed state in two successive time steps, we create a ray from the optical center of camera, passing through the pixel, and traverse all voxels which the ray passes using a ray casting technique [2].

The size of search space in this implementation is equal to the volume of the geometry which is created by back projecting changed part of each difference image in voxel space. It is obvious that if there exist small amount of changes in the 3D space the search space also will be small accordingly.

Deleting emptied voxels

The areas presented with red color in difference images – see figure 2 – are the projection of removed voxels since previous time step, so occupancy value for all voxels on each ray passing through these region – starting from camera optical center – should be set to empty. This fact is shown in the figure 5. The following pseudo code shows the deleting process in this method.

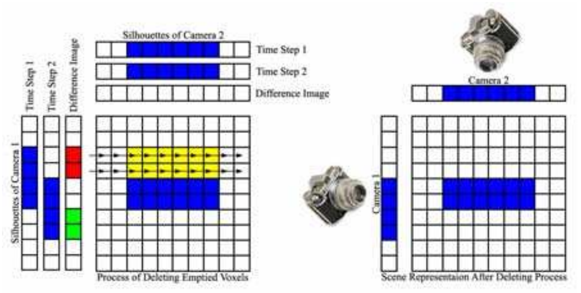

Figure 5: The traversing process using created rays from removed pixels in difference images. Yellow voxels are removed from previous visual hull.

```
for each image i in the difference images list
   for each removed pixel p in image i
   {
      create ray r from optical center passing from p
      set r to the first voxel in the reconstruction volume
      do
         set occupancy state of the voxel at r to empty
      while (step r to the next occupied voxel is successful)
   }
```

This algorithm searches all pixels in all difference images. Notice that, as expected, the search space size of this algorithm depends on size of areas with changed state. The implementation of ray casting methods is based on [2] as described briefly in the previous sections.

Adding newly occupied voxels

To complete the updating process, after removing deleted voxels, newly occupied voxels should be added. Figure 6 shows this routine. The following pseudo code creates newly occupied voxels in the reconstruction result of previous phase.

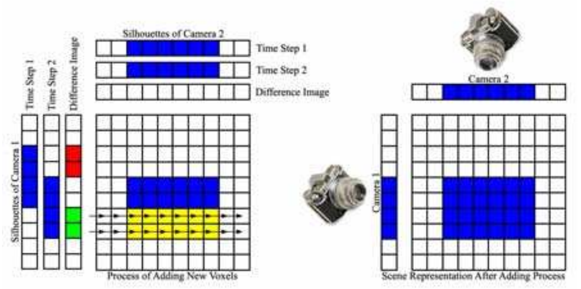

Figure 6: The traversing process using created rays from new pixels in difference images. Yellow voxels are added to previous visual hull.

```
for each image i in the difference images list
   for each added pixel p in image i
   {
      create ray r from the optical center passing from p
      set r to the first voxel in the reconstruction volume
      do
         if (voxel v at r is occupied)
            mark v as occupied
         else
            mark v as empty
      while (step r to next empty voxel is successful)
   }
```

The adding phase of this method is similar to its deleting phase. However before creating new voxels, their occupancy should be checked. Each voxel is occupied if its projection on all silhouette images, maps to a filled pixel. For checking occupancy silhouette images of current time step should be used.
The first frame is the only key frame in this method. All other changes can be reconstructed by updating phase of this approach. Even in the first frame, whole visual hull can be computed using blank silhouettes as images from previous time step. But the computation using this approach is redundant and the occupancy of some voxels might be checked more than once.

Ray buffers

More data we store from previous time step, more efficient methods can be developed. In most of the photo consistency approaches an image buffer is used for storing per pixel data. We have extended this idea with storing per pixel ray information, as described in [1, 13] and as is shown in figure 7, in each time step and keep these information for using in successive time step. We call this stored information as ray buffers.

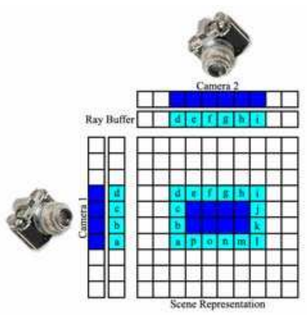

Figure 7: Ray buffer, storing information about surface voxels.
For each pixel the following information, required for ray casting [2] are stored:

- Starting point and moving direction.
- Distance toward next voxel border per axis.
- Current time and current voxel Id.
- Distance of current voxel from starting point.

This information helps us to restrict the search space even more. We should take care of this data in all parts of algorithm and update them whenever any change is happened on the surface voxels. The most important disadvantages of ray casting method, occurs whenever projection of a voxel extends in more than one pixel. In this situation which is common with high resolution cameras, some voxel will be processed several times. The simplest possible solution for this problem is to ignore some pixels with a predefined ratio. However this may cause some voxels not to be processed appropriately.
Using ray buffers, we can decide to only store one ray for each surface voxel and this will reduce the probability of missing a voxel during traversal process. Furthermore, using one ray per surface voxel will decrease the size of our computation dramatically.

Deleting emptied voxels

Information stored in the ray buffers can help us to directly jump to the first voxel that should be removed on each ray. In large scenes this may saves us a large amount of computation which is required for finding first visible surface voxel.

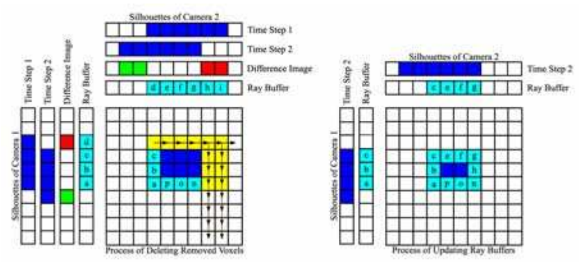

Figure 8: The visualization of deleting phase.

As can be seen in the figure 8, updating ray buffer, consists two different tasks. First is to remove previous surface voxels from all ray buffers and the second is to add voxels which currently become to a surface voxel. Following pseudo code shows the process of deleting emptied voxels in the presence of the ray buffers.

```
for each ray buffer rb and its paired difference image di
   for each removed pixel p in di with ray r in rb
      if (r is a border ray)
         do
            if (voxel v at r is surface voxel)
               for each ray buffer
                  erase v if it is not flagged
            set occupancy state of v to empty
            for each not surface voxel u adjacent to v
               if (u is consistent)
               {
                  set u as surface voxel
                  add u to all ray buffers
               }
         while (step r to next occupied voxel is successful)
      else
         do
            if (voxel v at r is a surface voxel)
               for each ray buffer
                  erase v if it is not flagged
            set occupancy state of v to empty
         while (step r to next occupied voxel is successful)
```

The only voxels which can change to border voxels are those, which are next to the border of emptied volume. So we divide this algorithm in two parts based on the rays. If the ray is in border of removed area so we call it a border ray. For all adjacent voxels to those, which are traversed using border ray, we check to see if they can become a surface voxel or not. Then, for new surface voxel, we update ray buffers as well.
We can find out that, if a voxel is on the surface of the reconstructed geometry, by checking all its adjacent voxels for an empty one. More than that, we can store this data as a flag in each voxel. Besides removing voxels, as described before, we should update ray buffers during this phase.
The deleting process is as same as previous method except the change which is added for updating ray buffers. We can make the deleting phase more accurate by traversing both front and back rays of geometry. Because rays associated with voxels on the back side of the geometry are denser and this will reduces the possibility of missing voxels.

Adding newly occupied voxels

For adding new voxels we can not go directly to the first voxel, because we have not required data. So we start to search like previous method but as soon as we find a new voxel we fill all the adjacent newly added area – like flood fill – with that voxel.

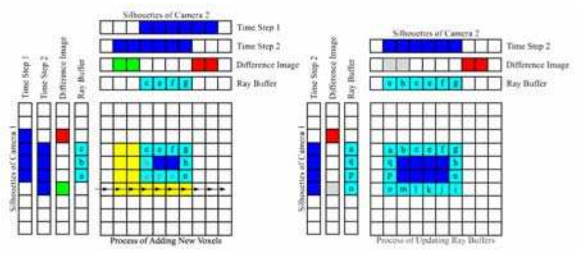

Figure 9: Visualization of adding new voxels phase.

```
for each image di in the difference images list
   for each added pixel p in image di
   {
      create ray r from the optical center passing through p
      set r to the first voxel in the reconstruction volume
      clear NVSVL list
      while (step r to next empty voxel is successful)
         fill occupied area from voxel at r

      for each voxel v in NVSVL
      {
         project v on all difference images as null
         add v to all ray buffers
      }
   }
```

Using this technique, we can project newly added surface voxels – which we store in NVSVL – on the difference images, erasing all green areas on their mapped pixels. The filling algorithm also erases all surface voxels which are not anymore on the border of reconstructed visual hull after filling process from all ray buffers. Figure 9 and the above code show this technique.
In this manner, we save lots of computation but it is possible to lose new parts which are hiding behind first created volume. But our scene is viewed by more than one camera so this reduces the risk. Also we can decrease this risk by only projecting starting voxel in the new volume however this needs more computation.

Adding with border search

Having the ray buffers in our hand, we can use their data to find the nearest adjacent voxels to the border of new volume. We use the same logic as previous border rays but this time the rays should pass inside of the previous visual hull. This idea is shown in the figure 10 and the following pseudo code.
For finding newly added voxels, the algorithm traverse through all the surface voxels which are on the inside border rays. These voxels can be adjacent to the recently filled areas so it checks the occupancy of all the neighboring voxels. In this method, the search process directly starts from the first adjacent voxel, which reduces the search space.

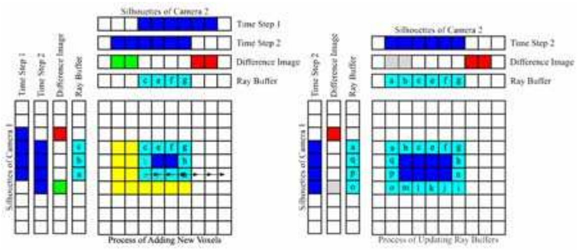

Figure 10: Adding process which searches adjacent voxels.

```
for each ray buffer rb and its paired difference image di
   for each added pixel p in di with border ray r in rb
   {
      clear the NVSVL list
      do
      {
         set v to voxel at r
         for each adjacent voxel of v like u
            fill occupied area from u
      }
      while (step r to next occupied voxel is successful)
      for each voxel v in the NVSVL
         add v to all ray buffers
}
```

Most of the other parts of this method are same as previous one. Updating ray buffer is done using NVSVL and also filling subroutine. Filling subroutine should be somehow implemented that rejects calls on currently occupied voxels.

Analysis

The discussed approaches for dynamic visual hull reconstruction change the search space from 3D voxel space to 2D image space. This method gives us a smaller search space and less occupancy checks. However 2D search space does not mean that for every scene configuration, using these algorithms leads to faster reconstruction. There are some different environmental aspects which directly affect the runtime. The output results of the teapot model are shown in figure 11 with removed and added voxels colored with red and green respectively.

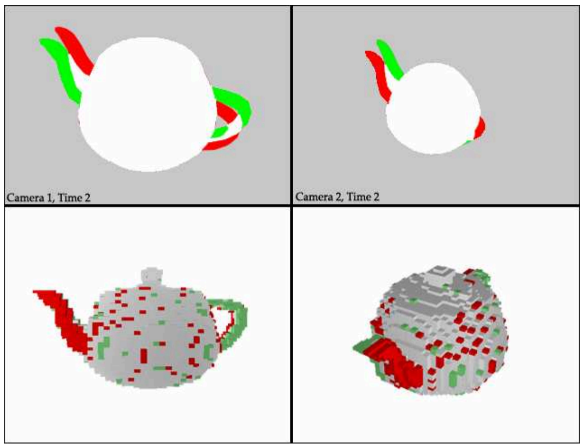

Figure 11: Top row, two of five difference images and bottom row, reconstructed model from two novel views.

Assume a scene consisting of N voxels per axis and the inner model, which should be reconstructed. Assume that the model occupies $v \times N^3$ voxels. Moreover most possible changed voxels between two consequent time steps is limited to $u \times v \times N^3$. Also assume that there exist M different cameras looking at the scene and there are at most r rays in the projection of each voxel on ray buffers. The value of r depends on image resolution and the size of voxel's projection on the image planes.

Number of surface voxels on the changed area can be computed by $\sqrt[3]{(u \times v)^2} \times N^2$, but notice that this changed area consist both added and removed voxels, so each part has $\sqrt[3]{(u \times v)^2 / 4} \times N^2$ surface voxels. The projection of each surface voxel has r rays per image, so putting all together the search space volume consists of $M \times r \times \sqrt[3]{(u \times v)^2 / 4} \times N^3$ voxels.

In table 1, we have compared the search space size and number of consistency checks between different visual hull reconstruction approaches in non key frames. Notice that, there is not any difference in the key frames between presented algorithms. In the teapot scene, we had $N = 64$, $v = 0.05$, $u = 0.04$, $r = 3$ and $M = 5$.

	Search Space	Occupancy Checks
Simple Reconstruction	N^3	N^3
Fattened Approach (*f* is Fattening Factor)	N^3	$f \times v \times N^3$
Ray Casting (Deleting)	$M \times r \times \sqrt[3]{(u \times v)^2 / 4} \times N^3$	—
Ray Casting (Adding)	$M \times r \times \sqrt[3]{(u \times v)^2 / 4} \times N^3$	$M \times r \times \sqrt[3]{(u \times v)^2 / 4} \times N^3$

Table 1: Comparison of different visual hull reconstruction approaches.

The process of checking occupancy is one of the most time consuming subroutines in all visual hull reconstruction approaches. For checking the voxel occupancy, the projection of given voxel should be computed on each camera image plane. Then the mapped color value on the projected area should be computed using different techniques such as interpolation and etc. Finally these extracted color values should be examined to find out if the voxel is occupied or not. Notice that in our approach, during the deleting phase, no occupancy checking is required.

From the accomplished statements it can be seen that the size of search space is directly related to the number of cameras and the number of rays per surface voxel. This shows the importance of good environmental configuration in this method. One more point to note is that, for static scenes or models with small movements, our approach does not spend long period of time for model reconstruction.

Our second reconstruction approach reduces the value of r down to one or even less. Using our approach is advised whenever $M \times r \times \sqrt[3]{(u \times v)^2 / 4} < 0.5$. In the teapot scene this value is 0.3 and 0.1 respectively using the first and the second algorithms.

Comparison

We have used a synthetic model, which is a rotating teapot rendered in 3DS MAX as our test data. One of the most important aspects about a reconstruction algorithm is its update time which is the time required by the algorithm to update the visual hull. In this section, we have computed and shown the update time in second unit. Furthermore methods with less occupancy checking are faster, so we have computed number of occupancy checking in each algorithm for more accurate comparison.

For all voxels in the scene the occupancy is checked using simple reconstruction, but in the fattened reconstruction approaches the count of checked voxels depends on the size of the fattened area we have used and size of the reconstructed visual hull in the previous time step. The number of occupancy checks, simply using ray casting, is close to the fattened methods as is shown in the chart 1. But with using of the ray buffers and optimizing algorithms this amount have become close to the real count of changed voxels.

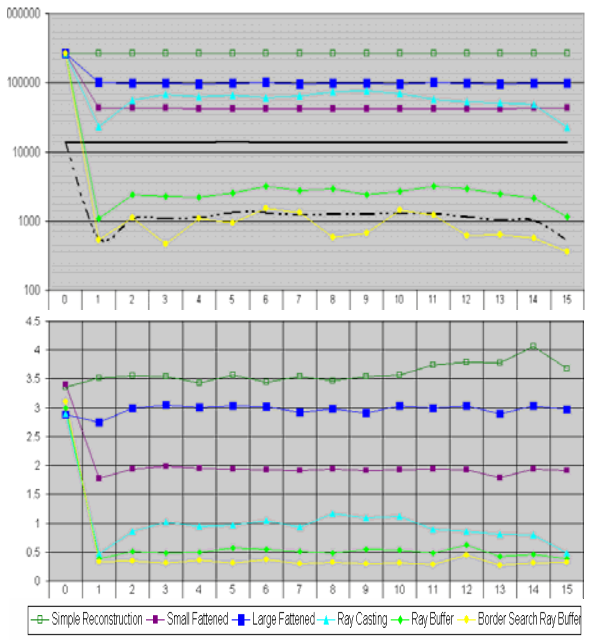

Chart 1: Comparison of the search space size and update times for the rotating teapot model, respectively in top and bottom charts.

The updating time of simple visual hull reconstruction method is simply depends on size of the reconstructed volume but the updating time of fattened approaches also depends on the previous time step reconstruction.

Conclusion

In this report, we have investigated the existing visual hull reconstruction approaches and their inefficiency in dynamic scenes reconstruction. Then we have proposed three new methods for reconstruction based on ray casting in the voxel space which increase the performance noticeably. Our method can be categorized as a passive, dense and local approach for extracting visual hull of a dynamic scene, incrementally. Furthermore, different aspects of a visual hull reconstruction algorithm have been introduced and then we have compared our implemented methods and previous algorithms against these aspects. This analysis proved the efficiency of our algorithms in the dynamic scenes.

References

1. O. Batchelor, R. Mukundan, R. Green, "Ray Casting for Incremental Voxel Colouring", Proc. of Image and Vision Computing Conference - IVCNZ05 NewZealand, (Nov 2005), pp 206-211.
2. J. Amanatides and A. Woo, "A fast voxel traversal algorithm for ray tracing," in Eurographics '87, pp. 3–10, Amsterdam, North-Holland: Elsevier Science Publishers, 1987.
3. R. Szeliski. Rapid octree construction from image sequences. Computer Vision, Graphics and Image Processing: Image Understanding, 58(1):23–32, 1993.
4. M. Potmesil. Generating octree models of 3D objects from their silhouettes in a sequence of images. Computer Vision, Graphics and Image Processing, 40:1–20, 1987.
5. C. H. Chien and J. K. Aggarwal. Volume surface octrees for the representation of 3D objects. Computer Vision, Graphics and Image Processing, 36:100–113, 1986.
6. M. Kimura, H. Saito, and T. Kanade. 3D voxel construction based on epipolar geometry. In Proc. Int. Conf. Image Processing, pages 135–139, 1999.
7. R. Szeliski and P. Golland. Stereo matching with transparency and matting. Int. J. of Computer Vision, 32(1):45–61, 1999.
8. Q. Chen and G. Medioni. A volumetric stereo matching method: Application to image-based modeling. In Proc. Computer Vision and Pattern Recognition Conf., volume 1, pages 29–34, 1999.
9. B. Curless and M. Levoy. A volumetric method for building complex models from range images. In Proc. SIGGRAPH 96, pages 303–312, 1996.
10. H. Saito and T. Kanade. Shape reconstruction in projective grid space from large number of images. In Proc. Computer Vision and Pattern Recognition Conf., volume 2, pages 49–54, 1999.
11. R. Szeliski. Stereo algorithms and representations for image-based rendering. In Proc. 10th British Machine Vision Conf., pages 314–328, 1999.
12. G. Slabaugh, B. Culbertson, T. Malzbender, and R. Schafer. A survey of methods for volumetric scene reconstruction from photographs. Technical Report 1, Center for Signal and Image Processing, Georgia Institute of Technology, 2001.
13. O. Batchelor, R. Mukundan, Ray Traversal for Incremental Voxel Coloring, M.Sc Thesis, 2006.
14. W. Matusik , C. Buehler , R. Raskar , S. J. Gortler , L. McMillan, Image-based visual hulls, Proceedings of the 27th annual conference on Computer graphics and interactive techniques, p.369-374, July 2000.
15. G. Zeng, S. Paris, L. Quan, F. Sillion, "Progressive Surface Reconstruction from Images Using a Local

Prior," iccv, pp. 1230-1237, Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 2, 2005.

16. A. Zhirkov. Binary volumetric octree representation for image based rendering. In Proc. of GRAPHICON'01, September 2001.
17. C. Andez and E. Schmitt. A snake approach for high quality image-based 3d object modeling. In 2nd IEEE Workshop on Variational, Geometric and Level Set Methods in Computer Vision, pages 241{248, Nice, France, 2003.
18. E. Trucco and A. Verri, Introductory Techniques for 3-D Computer Vision, Prentice Hall, 1998.
19. O. Faugeras, Three Dimensional Computer Vision: A Geometric Viewpoint, the MIT Press, 1993.
20. Tai Jing Moyung, Paul W. Fieguth: Incremental Shape Reconstruction Using Stereo Image Sequences. ICIP 2000.
21. Laurentini. The visual hull concept for silhouette-based image understanding. IEEE Trans. Pattern Analysis and Machine Intelligence, 16(2):150–162, 1994.
22. Ruigang Yang, Greg Welch, Gary Bishop (2003) Real-Time Consensus-Based Scene Reconstruction Using Commodity Graphics Hardware Computer Graphics Forum 22 (2) , 207–216.
23. Steven M. Seitz and Charles R. Dyer. Photorealistic scene reconstruction by voxel coloring. Int. J. Computer. Vision, 35(2):151-173, 1999.