
BBC

# Research & Development White Paper

## WHP 214

*November 2011*

# Fast Volumetric Visual Hull Computation

**Oliver Grau**

*BRITISH BROADCASTING CORPORATION*

White Paper WHP 214

# Fast Volumetric Visual Hull Computation

Oliver Grau

**Abstract**

Silhouette-based reconstruction techniques using multiple wide-baseline camera set-ups are popular building blocks to generate models of temporal human action. This contribution reviews visual hull computation algorithms. In particular we focus on the class of volumetric-based algorithms. We identify different processing strategies based on hierarchical, octree-based processing and suggest a new dynamic strategy that can significantly minimise the computational effort. The techniques discussed are evaluated with synthetic and real data sets. The savings demonstrated in computations on the real test sequences are in the order of a factor six on a 512x512x512 volumetric resolution..

This document was originally published in Proceedings of 2011 IEEE International Conference on Computer Vision Workshops, IEEE International Workshop on Dynamic Shape Capture and Analysis (4DMOD), November 13, 2011, Barcelona, Spain.

**Additional key words:** Computer Vision, Computer Graphics, Real-time signal processing.

White Papers are distributed freely on request.

Authorisation of the Chief Scientist or General Manager is required for publication.

# Fast Volumetric Visual Hull Computation

Oliver Grau
BBC Research & Development
Centre House, 56 Wood Lane, London, UK
Oliver.Grau@bbc.co.uk

## Abstract

*Silhouette-based reconstruction techniques using multiple wide-baseline camera set-ups are popular building blocks to generate models of temporal human action. This contribution reviews visual hull computation algorithms. In particular we focus on the class of volumetric-based algorithms. We identify different processing strategies based on hierarchical, octree-based processing and suggest a new dynamic strategy that can significantly minimise the computational effort. The techniques discussed are evaluated with synthetic and real data sets. The savings demonstrated in computations on the real test sequences are in the order of a factor six on a 512x512x512 volumetric resolution.*

## 1. Introduction

Silhouette-based reconstruction techniques using multiple wide-baseline camera set-ups are popular building blocks to generate models of temporal human action, for example see [11, 5, 13, 15]. The approaches rely on known camera parameters and segmentation of the foreground objects against background. The object silhouettes and camera parameters are used to compute an approximation of the scene. Laurentini coined the term *visual hull* for this class of reconstruction, as it only represents a convex approximation of the visible object shape [7]. The visual hull (VH) can be computed very robustly and many 4D reconstruction approaches use it solely [1, 4], or as an initial solution followed by a refinement, e.g. using stereo matching [6].

In this contribution we discuss aspects of fast volumetric visual hull computation. The computation of the visual hull from 2D silhouettes is very robust, fast and well-suited for modelling of human action. The computation of the visual hull, also known as shape-from-silhouette, is equivalent to an intersection of the back-projected 2D silhouettes (visual cones) in 3D. A number of approaches to compute this intersection have been suggested in the literature that differ in the underlying data structure and the processing.

### 1.1. Previous work

Algorithms for VH computation can be grouped according to the underlining data representation and/or processing principle:

*Volumetric VH computation*

A basic implementation of this algorithm runs over all elements (voxels[1]) of a 3D array and projects each voxel into the 2D silhouette images and tests whether its footprint intersects with foreground or background. If the voxel footprint is on background it can be set to 'false' or 'not-object'. From this volumetric data a surface description can be generated with an iso-surface extraction, for example using the marching cubes algorithm [8]. Using a 3D array in this simple algorithm requires a high number of projections and footprint tests. To reduce this effort the use of hierarchical processing using octrees as a representation has been proposed [12, 14].

Matsuyama et al. describe an algorithm that computes a VH by projecting the silhouette images into the volumetric space [9]. This is achieved with a 2D transformation into planes of a volumetric 3D array. The resulting back-projected volumes are then intersected in place.

*Edge-based polyhedral reconstruction*

This approach computes a surface description directly without going into an intermediate volumetric representation [10, 3]. These algorithms re-project the boundaries or edges of the 2D silhouettes using the inverse camera projection matrix and find the intersections in 3D to build up a polyhedral surface description.

*View-dependant for direct rendering*

Matusik et al. describe an algorithm that synthesises new views of an object directly without generating a surface model. Their algorithm samples the object based on lines along the viewing frustum of the new view and computes a VH for this parametrisation [11].

Although the algorithms discussed above aim to compute the visual hull of an object, they all have different characteristics. The (classical) volumetric algorithms with iso-

[1] Volumetric elements

surface generation are very popular, because of their low complexity. On modern machines they are also fast to compute. Furthermore, they are very robust against segmentation errors. As shown later the computational expensive part of the computation is the number of required 3D projections. The next section reviews the various implementations of volumetric VH computation algorithms and discusses means to reduce the computational effort.

In section 3 the run-time behaviour of volumetric VH algorithms is analysed and compared. Based on this analysis we propose a new dynamic processing strategy that can significantly reduce the computational effort. The paper finishes with a discussion of this new strategy.

## 2. Visual hull computation

The VH (visual hull) gives an approximation of a scene $S$ at time $t$ from silhouettes of the scene taken from a number of viewpoints. The silhouettes are usually binary label images indicating whether a point in the label image belongs to the scene object (foreground) or to the scene background. The label images can be created using any keying or segmentation technique, like chroma- or difference keying. Also, camera parameters for all label images are needed that allow employment of a camera model to project 3D points into the label images. That means the external orientation and internal camera parameters have to be known.

### 2.1. Basic volumetric visual hull computation

A basic algorithm to compute a VH from a list of cameras $D = \{\{L_0, C_0\}, .., \{L_N, C_N\}\}$ , with camera projection parameters $C_i$ and label images $L_i$ is shown in algorithm 1.

**Algorithm 1** Basic volumetric visual hull computation (camera loop).

**for all** $voxel$ in $V_{def}$ **do**
  $voxel := true$
**end for**
**for all** $c$ in $listofcameras$ **do**
  **for all** $voxel = true$ **do**
    $fp = FootPrintTest(c, voxel)$
    **if** $fp = AllZero$ **then**
      $voxel := false$
    **end if**
  **end for**
**end for**

The first loop sets all voxels to 'true' (object). Then all voxels are tested in all cameras with a footprint test. Algorithm 2 shows an alternative implementation that swaps the order in which the loops are nested and tests each voxel in all cameras. Since the only voxels that have to be tested are those that are still 'true', both implementations have different run-time behaviour, as will be discussed in section 3.

Algorithm 1 is referred to as 'camera loop' since the 'dominant' loop runs over the camera list; conversely algorithm 2 has the voxel loop as the dominant one.

**Algorithm 2** Basic volumetric visual hull computation (voxel loop).

**for all** $voxel$ in $V_{def}$ **do**
  $voxel := true$
**end for**
**for all** $voxel = true$ **do**
  **for all** $c$ in $listofcameras$ **do**
    $fp = FootPrintTest(c, voxel)$
    **if** $fp = AllZero$ **then**
      $voxel := false$
    **end if**
  **end for**
**end for**

### 2.2. Octree visual hull computation

Octrees were introduced as a data representation for memory-efficient storage. Because of their tree structure they inherently support hierarchical processing. Octree nodes are further sub-divided into 8 equally spaced child-nodes as depicted in figure 1. Algorithm 3 shows the modified visual hull computation using octrees.

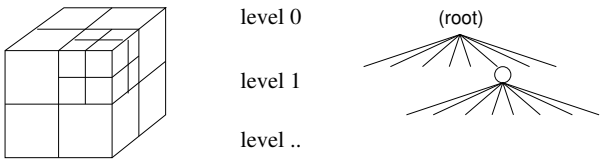

Figure 1. One octree node is sub-divided into 8 child nodes.

The main difference from the basic algorithms (1 + 2) is the fact that an octree node will not be further sub-divided if the footprint test reports 'AllZero'. This results in a potential saving of operations, as discussed in section 3.

Octree nodes can be regarded as voxels with varying size. That means the same footprint test function as for volumetric data can be used.

By analogy to the basic algorithms, algorithm 3 can be modified into processing voxels first (voxel loop).

### 2.3. Two-pass octree visual hull computation

The octree-based algorithm 3 can be further refined. For this purpose we propose using a two pass strategy:

In the first pass the maximal descend level (octree level, compare fig. 1) is limited to $p1_{maxlevel} = n$. That means

**Algorithm 3** Octree hierarchical visual hull algorithm (camera loop).

```
for all voxel in V_def do
  voxel := true
end for
for all c in listofcameras do
  for all octreenodes = true do
    fp = FootPrintTest(c, octreenode)
    if fp = AllZero then
      voxel := false
    else if fp = Some then
      Subdivide octree node
      childnodes = true
      descend
    end if
  end for
end for
```

the octree nodes are not further sub-divided if this level is reached.

In a second pass all octree nodes $\neq$ 'false' are revisited and further sub-divided. The potential saving of this strategy comes from the effect of different run-time behaviours of the loop dominance, as we will demonstrate in section 3 below.

For the maximal level of the first pass we determined experimentally a value of $p1_{maxlevel} = p_{maxlevel}/2$, ie. half of the final maximum octree depth.

### 2.4. Footprint test

A footprint is the projection of a (sub-)volume into a camera $C_i$. The footprint test determins whether this area intersects with pixels marked as 'foreground' in the segmented label image $L_i$ associated with that camera, as depicted in fig. 2. A correct test would project eight vertices defining the considered volume and scan all pixels of the polygonal outline of that projection in the label image.

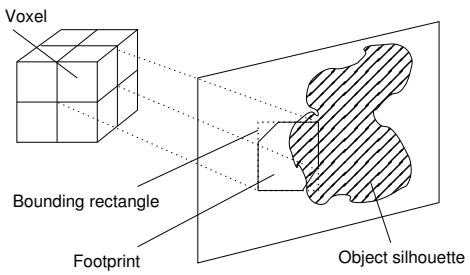


Figure 2. Footprint test of a voxel.

In practice usually the 2D bounding box $a_{bounding}$ of the footprint is used in the footprint test. This will cause errors in the resulting reconstruction, but in the case of a hierarchical octree-based processing a potentially wrongly-classified 'foreground' node is delegated to the next subdivision level for a refined test. The residual approximation error will be minimised that way.

The evaluation of the footprint test of an area $a := x \in (x_1, y_1, x_2, y_2)$ returns three states:

$$l(a) = \begin{cases} AllZero, & \text{if all pixels in a=0} \\ AllNonZero, & \text{if all pixels in a!=0} \\ SomeZero & \text{else} \end{cases} \quad (1)$$

A naïve implementation would evaluate all pixels in the areas $a_{bounding}$ of a footprint. However this would require a lot of memory read operations. A number of techniques are in use that reduce this bottleneck:

Instead of evaluating all pixels in a bounding box only the centre pixel might be evaluated. This technique might lead to severe errors as some voxels might be assigned to 'False' (i.e. 'non-object'), leading to an eroded volumetric reconstruction. An alternative is using a *pyramid* approach.

Szeliski [14] uses a pre-computed checkerboard distance function to reduce the computational effort of the evaluation. This method is based on squares and needs at least one memory look-up and more to approximate non-squared footprints.

More recently the technique of *summed area table* (also known as an integral image) [2] is used to perform a footprint test. This technique needs four memory reads to determine the exact number of foreground pixels in a rectangular area. A variation of this technique is to evaluate a 'per line' summed area table (*line sum table*). Both techniques will be compared in the next section.

### 2.5. Super-sampling

As described in our previous work [5] super-sampling can be successfully applied to further decrease the approximation error introduced by the simplified footprint test.

## 3. Analysis of the run time performance

In this section we analyse the run time performance of different implementations of volumetric visual hull computations. We use synthetic generated data and two real data sets, as described below to evaluated the run time performance of these algorithms.

### 3.1. Test data set 'sphere'

The 'sphere' data set (see fig. 3) is a synthetic data set, of one or a number of spheres rendered in HD-resolution (1920 x 1080). Various parameters were varied for the experiments:

1. size of the sphere(s), to achieve different fill rates in the images

2. addition of noise, to simulate erroneous segmentation

The data set is rendered for 18 camera views. All cameras are positioned on a circle, slightly elevated. One example is depicted in 3.

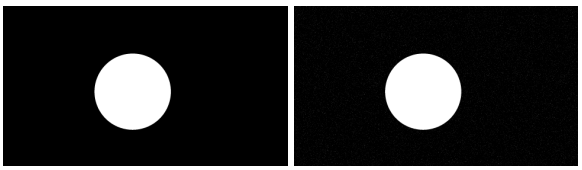

Figure 3. One view of the 'Sphere' data set (radius 0.6), noise 0% (left), noise 1% (right).

### 3.2. Test sequence 'Peter'

The sequence 'Peter' a short sequence of one person captured in a studio with chroma-keying facility, as depicted in fig. 4. The scene was captured with 10 HD cameras (Sony HDC-X300) at HD resolution (1920 x 1080), at 25 fps. In addition one SD-resolution camera (720 x 576) was taking pictures from the studio ceiling. The camera set-up is depicted in fig. 5.

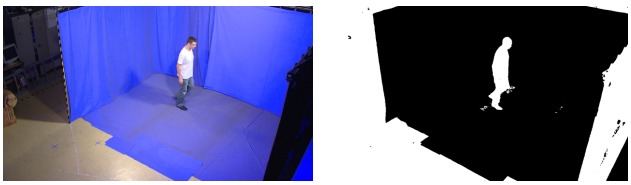

Figure 4. Sequence 'Peter',left: frame 100, right: key-image.

The images from the cameras are keyed using chroma-keying. The image fill rate (ratio of foreground to all image pixels) is between 7% and 40%. Although some cameras show a lot of 'inactive' areas, that offset this ratio (see fig. 4 right). The 'active' area is more around 7-10 %.

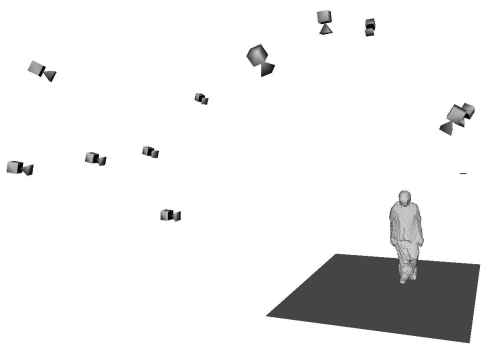

Figure 5. Camera set-up test sequence 'Peter'.

### 3.3. Test sequence 'Rugby'

The sequence 'Rugby' was captured during a rugby game, using five broadcast HD cameras (1920 x 1080 resolution). The segmentation is based on motion compensated difference keying (see [4] for details). The fill rate is between 5% and 15%, depending of the camera framing and consists of more objects than in the previous studio scene.

Figure 6. Test sequence 'Rugby', one frame of main camera (left) and key (right).

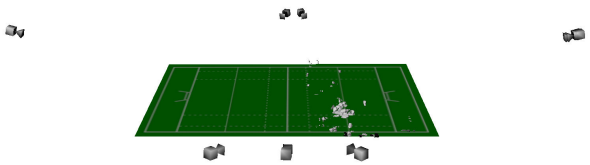

Figure 7. Camera set-up test sequence 'Rugby'.

### 3.4. Comparing different volumetric VH computation methods and strategies

We have carried out a number of experiments in order to evaluate the run-time performance of different VH computation implementations, as described in section 2. The aim of these tests was to analyse and optimise the run-time performance of the considered algorithms.

#### 3.4.1 Effect of footprint test method

Table 1 demonstrates the impact of different footprint evaluation methods on the computation time for a visual hull. The timings were measured on a Intel P9400@2.40GHz, executed single threaded. The sequence 'Peter' was used here with an octree depth of 10 (1024 x 1024 x 1024). The computation required $1.7 * 10^7$ projections. The methods tested were 'full search' over all pixels of the bounding rectangle of the footprint, 'summed area table' and 'summed row table', as described in section 2.4.

The time for 'Pre-compute' contains computations like the summed area table or row. The time for VH computation contains the time for building an octree reconstruction, including projections and footprint test.

| Method | Pre-compute | VH | Total |
|---|---|---|---|
| full search | 0 s | 8.65 s | 8.65 s |
| summed area table | 0.24 s | 6.76 s | 7.00 s |
| summed row table | 0.1 s | 7.45 s | 7.55 s |

Table 1. Run time for VH computation.

As expected, the ‘full search’ method performs worst. The ‘summed area table’ method performs best, but has a longer pre-computation time than the ’summed row table’ method. This is surprising, because both perform almost the same operation: summing the numbers of ‘foreground’ pixels in one pass and storing it into the table. The only difference is that the area table needs a longer datum (64bit) than the row method (32bit), to avoid numerical overflows. That means that the limiting factor is the memory write-throughput and the measured number will vary on different hardware.

On hardware with very high memory bandwidth the ‘summed area table’ method should perform even better than on the tested hardware. On the other hand on ‘moderate’ commodity hardware, as used in the tests and a lower octree resolution the ‘summed row table’ method might perform faster overall, since the gain in the pre-computation might be higher than the gain in the footprint test.

The measured data in table 1 shows further that the influence of the footprint test method on the overall computational costs is as little as 13% on the tested data and hardware configuration, the main cost being the projections.

### 3.4.2 Strategies to reduce number of projections

As demonstrated by others (e.g. [14]) a hierarchical processing using octrees can reduce the number of projections and footprint tests effectively. Table 2 shows for the ’Peter’ sequence the number of projections needed for basic voxel-based and hierarchical octree-based computation of different volumetric resolutions $N$ [2]. The ’gain’ factor is the saving, i.e. ratio between both numbers. The advantage of a hierarchical octree-based approach over a full volumetric expansion (labeled ’voxel’) is very visible at higher resolutions. Similar results are obtained for the synthetic and the ’Rugby’ sequence, see appendix.

Fig. 8 shows the number of projections needed to compute a VH for the sequences ’Peter’ and ’Rugby’. The curve ’c-loop’ and ’v-loop’ correspond to the camera and/or voxel loop dominance, as discussed in algorithm 3 and 2, applied to hierarchical octree-based processing.

The *c-loop* algorithm loops over all cameras and computes the full depth of an octree for one camera at a time. The *v-loop* algorithm loops over all active octree node (i.e. those not set to ’false’ and tests for these nodes the footprints in all cameras (compare algorithm 2).

[2] actual volumetric resolution is $NxNxN$

| Resolution | voxel | octree | gain |
|---|---|---|---|
| 16 | 10358 | 7031 | 1.47 |
| 32 | 71263 | 18411 | 3.87 |
| 64 | 529275 | 55674 | 9.51 |
| 128 | 4.09661e+06 | 207917 | 19.7 |
| 256 | 3.22281e+07 | 929475 | 34.67 |
| 512 | 2.55365e+08 | 4.09051e+06 | 62.43 |

Table 2. Projections for sequence ’Peter’.

The *two-pass* algorithm uses the same strategy but stops at a certain level of the octree, as described in section 2.3. We evaluated both variants of the two-pass strategy, ’two-pass-c’ is based on the camera loop dominance and ’two-pass-v’ on the voxel dominance. However, the voxel loop dominant ’two-pass-v’ did not present any advantage in our experiments. Conversely the camera dominant ’two-pass-c’ showed interesting features: It performs best in the ’Peter’ sequence and is usually close to best performing strategy. In the following we only consider the camera dominant two-pass algorithm (abbreviated ’two-pass’).

As apparent in fig. 8 the algorithms perform quite differently on the two test sequences. We therefore further investigated the run-time behaviour of the algorithms on synthetic data.

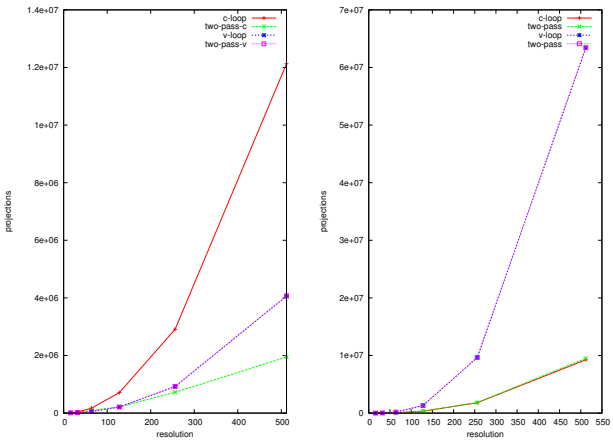


Figure 8. Number of projections for sequence ’Peter’ (left) and ’Rugby’ (right).

Fig. 9, 10 and 11 show the results of some experiments with synthetic data using projections of a ’sphere’ as described in section 3.1. The radius of the sphere is varied and depicted on the x-axis of the diagrams. A radius of ’0.1’ produces a fill rate of approximately 1.2 % and ’1.0’ approximately 25 %. Different amounts of white noise were added to the synthesised images (as shown in fig. 3). Fig.

9 with 0 % noise and 0.1% and 1.0% noise in fig. 10 and 11. The number of projections is depicted for the 'c-loop', 'v-loop, and 'two-pass-c' (shown as 'two-pass').

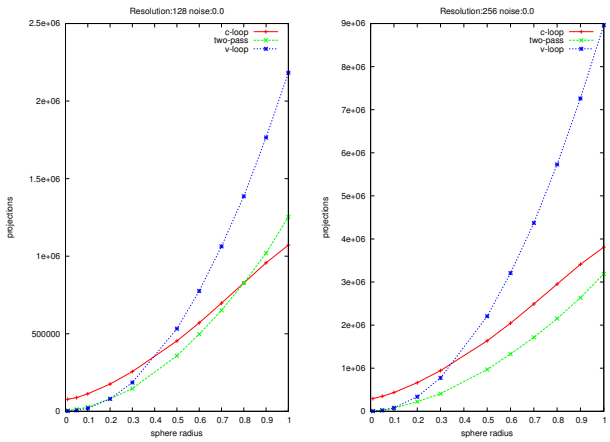


Figure 9. 'Sphere' in different object sizes, no noise.

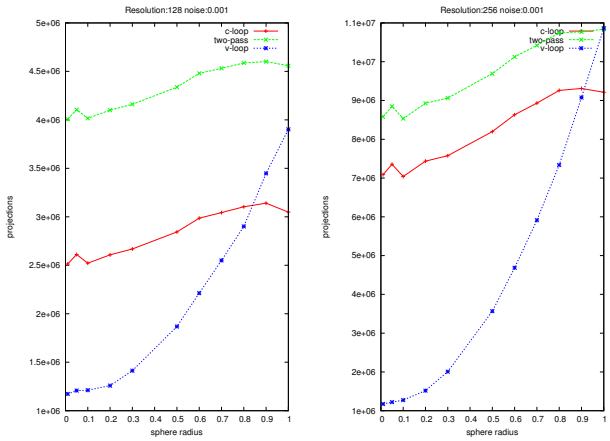


Figure 10. 'Sphere' in different object sizes and 0.1% noise.

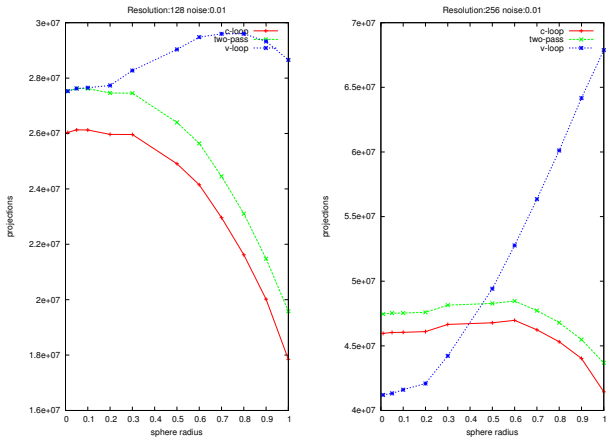


Figure 11. 'Sphere' in different object sizes and 1% noise.

The test shows that the number of projections is a function of different factors. The 'c-loop' algorithm usually performs best, with some exceptions. In particular when the object fill rate is very small and in presence of 'light noise' (fig. 10) the 'v-loop' algorithm can perform better here, but the 'c-loop' algorithm is usually better with high object fill rates.

Another observation is, that the 'two-pass-c' algorithm is usually relatively close to the optimal loop dominance. It fails on 'light noise' (fig. 10), but performs even better without added noise (fig. 9).

On the two examined real sequences 'Peter' and 'Rugby' the 'two-pass-c' algorithm performs consistently very close the the best loop dominance algorithm (fig. 8). Table 3 shows the numerical results for both sequences measured for 512x512x512 resolution. The factor in terms of reduction of projections compared to the worst loop dominance algorithm is shown in column 'gain'.

| Sequence | c-loop | v-loop | two-pass-c | gain |
|---|---|---|---|---|
| Peter | 1.213e+07 | 4.061e+06 | 1.948e+06 | 6.23 |
| Rugby | 9.258e+06 | 6.342e+07 | 9.506e+06 | 6.67 |

Table 3. Projections for sequence 'Peter' and 'Rugby' at 512x512x512.

## 4. Conclusions and future work

The analysis of the run-time performance of the octree-based VH algorithms with different loop dominance, as introduced in section 2 shows that it is possible to adapt volumetric VH algorithms in order to minimise the computational effort to particular scenes and/or camera setups. The factors ivestigated were object fill rates and erroneous segmentation. For the tested real sequences the gain was demonstrated to be in the order of a factor six by choosing the optimal loop order on a 512 x 512 x 512 volumetric resolution, which is a typical application scenario.

For a practical use of these findings we are suggesting to test on the particular scene being processed which loop dominance gives the best performance and use this configuration. Alternatively the two-pass approach with camera dominance can be used if the scenes and characteristics change over time, as it gave good performance on average in our tests.

We plan to further test the derived volumetric algorithms on more scenes and sequences in the future. Furthermore, we think it could be beneficial to design and test more dynamic adaptive strategies. This could take into account different characteristics of the used camera views, in particular if the scene objects are projected with a (very) different scale in different cameras.

## Appendix

| Resolution | voxel | octree | gain |
|---|---|---|---|
| 16 | 9455 | 8152 | 1.16 |
| 32 | 62346 | 30550 | 2.04 |
| 64 | 456755 | 127266 | 3.59 |
| 128 | 3.50211e+06 | 532447 | 6.58 |
| 256 | 2.74501e+07 | 2.20645e+06 | 12.44 |
| 512 | 2.17403e+08 | 9.08861e+06 | 23.92 |

Table 4. Projections for 'sphere' (r=0.5,noise=0)

| Resolution | voxel | octree | gain |
|---|---|---|---|
| 32 | 183348 | 21312 | 8.6 |
| 64 | 946598 | 168768 | 5.61 |
| 128 | 5.98238e+06 | 1.33436e+06 | 4.48 |
| 256 | 4.29311e+07 | 7.36298e+06 | 5.83 |
| 512 | 3.27237e+08 | 2.09772e+07 | 15.6 |
| 1024 | 2561215708 | 4.48391e+07 | 57.12 |

Table 5. Projections for sequence 'Rugby'.

## References


[1] P. Benjamin, L. Jean-Denis, et al. Multicamera real-time 3d modeling for telepresence and remote collaboration. *International Journal of Digital Multimedia Broadcasting*, 2010, 2009.

[2] F. Crow. Summed-area tables for texture mapping. *ACM SIGGRAPH Computer Graphics*, 18(3):207–212, 1984.

[3] J.-S. Franco and E. Boyer. Exact polyhedral visual hulls. In *In British Machine Vision Conference*, pages 329–338, 2003.

[4] O. Grau and et al. A robust free-viewpoint video system for sport scenes. In *Proc. of 3DTV-Conference*, 2007.

[5] O. Grau, T. Pullen, and G. A. Thomas. A combined studio production system for 3-d capturing of live action and immersive actor feedback. *IEEE Transactions on Circuits and Systems for Video Technology*, 14(3):370–380, March 2004.

[6] J. Guillemaut, J. Kilner, and A. Hilton. Robust graph-cut scene segmentation and reconstruction for free-viewpoint video of complex dynamic scenes. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 809–816. IEEE, 2010.

[7] A. Laurentini. The visual hull concept for silhouette-based image understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 150–162, 1994.

[8] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 163–169. ACM Press, 1987.

[9] T. Matsuyama, X. Wu, T. Takai, and T. Wada. Real-time dynamic 3-d object shape reconstruction and high-fidelity texture mapping for 3-d video. *IEEE Transactions on Circuits and Systems for Video Technology*, 14(3):357–369, March 2004.

[10] W. Matusik, C. Buehler, and L. McMillan. Polyhedral visual hulls for real-time rendering. In *Proc. of 12th Eurographics Workshop on Rendering*, pages pages 116–126, 2001.

[11] W. Matusik, C. Buehler, R. Raskar, S. J. Gortler, and L. McMillan. Image-based visual hulls. In K. Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 369–374. ACM Press, 2000.

[12] M. Potmesil. Generating octree models of 3D objects from their silhouettes in a sequence of images. *Computer Vision, Graphics and Image Processing*, 40:1–29, 1987.

[13] J. Starck and A. Hilton. Model-based multiple view reconstruction of people. In *Proc. of ICCV*, pages 915–922, 2003.

[14] R. Szeliski. Rapid octree construction from image sequences. *CVGIP: Image Understanding*, 58(1):23–32, July 1993.

[15] S. Würmlin, E. Lamboray, O. Staadt, and M. Gross. 3D Video Recorder: a System for Recording and Playing Free-Viewpoint Video. In *Computer Graphics Forum*, volume 22, pages 181–193. Wiley Online Library, 2003.