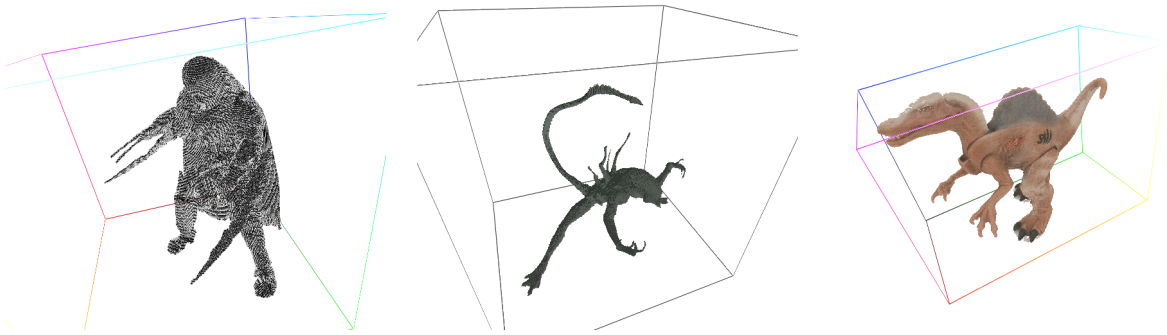
Report of Mini-Project #2

Peihong Guo

Department of Computer Science and Engineering

October 28, 2011



(a) (b) (c)

Figure 1: Sample results: (a)Volumetric visual hull with regular grid; (b)Volumetric visual hull with octree; (c)Voxel coloring.

In this mini-project, I implemented two image-based volumetric model reconstruction algorithms: the visual hull algorithm[1] and the voxel coloring algorithm[2, 3]. The initial results I obtained are quite promising, and lots of possible improvements exist.

The image data and camera parameters used are provided by Jean Ponce at University of Illinois at Urbana-Champaign[1]. Figure. 1(a) is a visual hull of the predator model, reconstructed with regular grid. Figure. 1(b) is a visual hull of the alien model, reconstructed with octree. Figure. 1(c) is a recontruction of the dinosaur model with voxel coloring.

1 Introduction

The basic idea of reconstructing volumetric model from multiple view image sequence is to divide the model into massive voxels, each of which represents a perticular

[1] http://www-cvr.ai.uiuc.edu/ponce_grp/data/visual_hull/

position in the scene, and reconstruct the model through assigning color information to each voxel. The key to successful reconstruction with the volumetric approach is differentiating model voxels from those redudant voxels correctly. Given the projection matrix for each image, the model voxels can be identifide by comparing their projected footprints in each images with the model's contour. Voxels with footprints inside the model's contour are considered to be model voxels, otherwise are regarded as redundant voxels and are thus eliminated. For voxel coloring algorithm, correspondances between voxels and its projection in multiple images are crucial in assigning correct color to voxels since the color of voxels are determined by their footprints.

One of the biggest disadvantage of the volumetric model reconstruction approach is its high time complexity in achieving high quality model. Octree representations are thus introduced to [4] as a method to reduce the cost for reconstruction. One straight forward explaination to the advantage of octree over regular grid is that the octree representation allows early and collective pruning of redudant voxels, while for regular grid large amount of extra effort are needed for those voxels. The advantage of using octree representation can be also viewed as its capability of representing model with the equal preicision to that by regular grid with less memory cost due to its adaptive nature. Figure. 2 is an illustration of the difference between regular grid and octree-based representation.

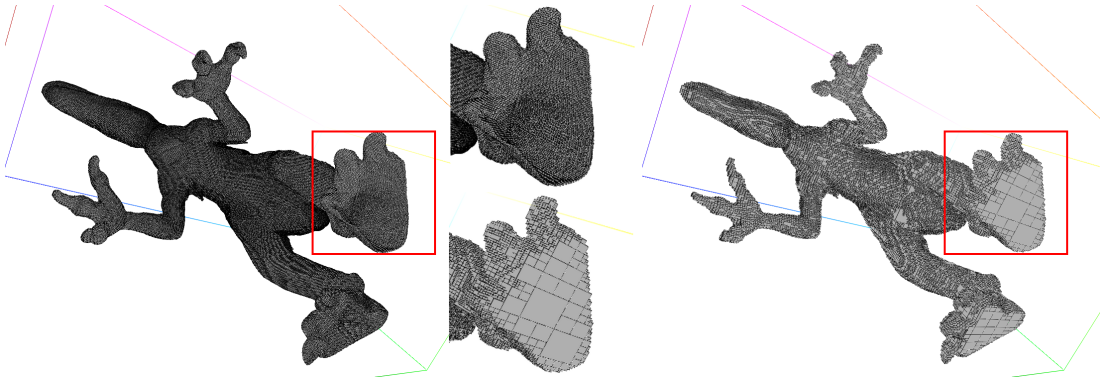


Figure 2: An illustration of the difference between regular grid and octree-based representation.

Efficient storage of the reconstructed model is another concern in this project. In pratical consideration, the surface of the model is sufficient for the purpose of representing an opaque model, since both the interior and exterior region of the model can be derived from the surface information. Therefore only the surface region of the model is of our interest in representing a model, which implies a efficient scheme for encoding the model should only include surface voxels and ignore other voxels.

2 Implementation Details

2.1 Basic Settings

A few settings are introduced to facilitate the reconstruction process, including a auxiliary description file as reconstruction guide and a voxel array model for storing the reconstructed model.

The machine used for this experiment is equipped with an AMD A3850 quad core CPU running at 2.9GHz and a total memory of 8GB.

Description file The description file is used to storing relevant information of a reconstruction process. The structure of the description file is as follows:

- NUMBER *number of images*
- NAME *model name*
- PATH *path to image sequence*
- IMAGE_FILE_NAME_LENGTH *the length of image file*
- IMAGE_FILE_NAME_SUFFIX *the suffix of image file*
- CAMERA *path to camera parameters file*
- CONTOUR *path to contour file*
- XRANGE *the min and max x coordinates of volume*
- YRANGE *the min and max y coordinates of volume*
- ZRANGE *the min and max z coordinates of volume*

The reconstructor uses the description file to search for image sequence and other relevant files, as well as setting the range of volume for reconstruction. Note that the min coordinate of the volume can be larger than max coordinate, which means a mirror operation is needed for that direction.

Volume format The reconstructed volume is stored as a list of opaque cubes, each of which represents an occupied region of the scene volume. The scene volume is assume to be a unit cube for convenience, so the coordinates value of vertices for each cube is within 0 to 1. The scale in each direction is therefore needed to correctly represent the shape of the model. Color information is attached to each cube, denoting the correspoding color for that cube. At the head of the voxel array file are the number of voxels and the scale of each axis.

A sample voxel list file (.vxl) is as follows:

- NUMBER *number of voxels*
- XSCALE *scale of x axis*
- YSCALE *scale of y axis*
- ZSCALE *scale of z axis*
- $x1_{min}\ x_{1,max}\ y_{1,min}\ y_{1,max}\ z_{1,min}\ z_{1,max}\ r_1\ g_1\ b_1\ a_1$
- $\vdots$
- $x_{n,min}\ x_{n,max}\ y_{n,min}\ y_{n,max}\ z_{n,min}\ z_{n,max}\ r_n\ g_n\ b_n\ a_n$

Contour Images A preprocessing step is needed to convert the contour polygon described in sets of coordinates into two-value contour images, where black color represents region within the contour and white color for outside region. This is done by rendering the concave polygon into images with the same sizes of corresponding input images. These two-value contour images are then used for foorprint tests performed in both visual hull and voxel coloring algorithms. Determining whether a pixel lies within contour or not is then equivalent to testing if it is a black or white pixel, thus avoiding the time consuming polygon-point relation judgement. An example contour image is shown in Figure. 3.

Figure 3: An illustration of contour images.

2.2 Visual Hull

The algorithm of volumetric visual hull reconstruction in regular grid is as follows:

The $Check - coverage$ subrutine is defined in Algorithm. 4:

Algorithm 1 Volumetric Visual Hull($voxels$)

```
for all v ∈ voxels do
    footprint ← Project-to-images(v)
    isVald ← Check-coverage(footprint)
    if isValid then
        model.add(v)
```

Algorithm 2 Check-coverage($footprint$)

```
insideCount ← 0
for all pixel ∈ footprint do
    if isInContour(pixel) then
        insideCount++
coverage ← insideCount/footprint.size
return coverage > threshold
```

The threshold is specified by user input. The reason for checking coverage ratio to determine if a voxel belongs to the target model is that for each voxel there are 3 possibilities: its footprint resides inside the contour, outside the contour, or on the contour. For the inside cases, the coverage is 1, meaning that all footprint pixels are within the contour. For the outside cases, the coverage is 0, since no pixel lies within the contour. For boundary cases, some footprint pixels are inside the contour, while others are not, resulting in a coverage between 0 and 1. It is intuitive that the higher coverage is, the more likely that a voxel belongs to the model. Therefore only voxels with coverage above threshold are marked as belongs to the model.

Note that the extreme cases can be utilized to simplifie the model. Voxels with coverage value of 1 is completely inside the model's surface, therefore does not contribute to refining the model. These voxels are therefore not included in the final model. Voxels with zero coverage for any image is completely outside the model's surface, therefore should be carved as early as possible. In my implementation the coverage test is performed for each footprint-image pair, and voxels with 0 coverage are carved as soon as it is found.

Typically, threshold value below 0.5 produces best results. However, very low threshold values tend to produce model with redundant voxels included, which is expected. Threshold value higher than 0.5 tends to result in over-carved models. The threshold value is also sensitive to the size of voxel used. For larger voxel sizes, lower thresholds tend to produce better models; while for smaller voxel sizes higher thresholds are better.

The reconstruction can be accelerated by using a octree model. The algorithm

for octree-based volumetric visual hull is very similar to that for regular grid, differing only in the voxels used for construction. The octree-based process tests large voxels first to see if it satisfies the coverage condition. Large voxels are accepted if they pass the coverage test, otherwise they are splitted into smaller ones, which are then tested. The iterative process proceeds until voxels pass the coverage test, or a minimum voxel size threhold is reached.

Corresponding to the coverage threshold, in the octree algorithm a splitting coverage threshold is used to determine whether a large voxel should be splitted into smaller ones or not. In most cases, large voxels are more likely to be further divided than smaller ones, so the threshold is actually voxel-size-dependent. To ensure the reconstruction quality, a lower bound for the voxel size is introduced. In my implementation, the threshold value decreases from 1 to the user-specified lower bound from top level voxels to lowest level voxels.

2.3 Voxel Coloring

The algorithm for voxel coloring is also very similar to that for regular grid visual hull, except that in voxel coloring $Check - consistency$ is used instead of $Check - coverage$ and extra image masks are used.

Algorithm 3 Voxel Coloring($voxels$)

```
1: for all v ∈ voxels do
2:     footprint ← Project-to-images(v)
3:     isVald ← Check-consistency(footprint)
4:     if isValid then
5:         Mark-footprint-in-masks(footprint)
6:         model.add(v)
```

The $Check - consistency$ subrutine is defined as follows:

Note that extreme coverage properties can also incorporated in voxel coloring algorithm by changing the $Check - consistency$ subrutine a little. The background color test in Algorithm. 4 is a general one, which eliminates voxels with similar color to background. In my implementation, the elimination scheme is even simpler, which is disgarding all voxels above an intensity threshold, since the background color is always nearly white.

The computation for voxel coloring is much more time consuming than visual hull, which is caused by expensive calculation of the average and standard deviation of footprint pixels color. This is especially the case when the input images are large,

Algorithm 4 Check-consistency($footprint$)

$avgcolor \leftarrow 0$
for all $pixel \in footprint$ **do**
    if isInContour($pixel$) **then**
        $avgcolor+ = pixel$.color
$avgcolor/ = footprint$.size
$sigma \leftarrow 0$
for all $pixel \in footprint$ **do**
    if isInContour($pixel$) **then**
        $sigma+ = (avgcolor - pixel.\text{color})^2$
    else
        $sigma+ = (255^2, 255^2, 255^2)$
$sigma/ = footprint$.size
$isConsistent \leftarrow sigma < consistency\_threshold$
$notBG \leftarrow ||avgcolor - background\_color|| < background\_threshold$
return $notBG \&\& isConsistent$

for example 1600x900 or even larger, since the footprints can contain from tens to hundreds of pixels. To accelerate the computation, further approximation is introduced by restricting the footprints to much smaller window. The window size used in my implementation is 7x7, which produces nice results in reasonable time. However, this is an emperical value and can be optimized through further analysis.

3 Results and Discussion

In the following sections, some results obtained by my implementation is presented. The data set I used includes three different models:

- Alien: default volume size is 288x256x288;
- Dinosaur: default volume size is 312x600x436;
- Predator: default volume size is 640x640x640;

3.1 Visual Hull

Two approaches to estimate the visual hull are experimented, i.e. regular grid based and octree-based methods. The results show that both methods have their advantages and disadvantages. The regular grid based method is usually faster for

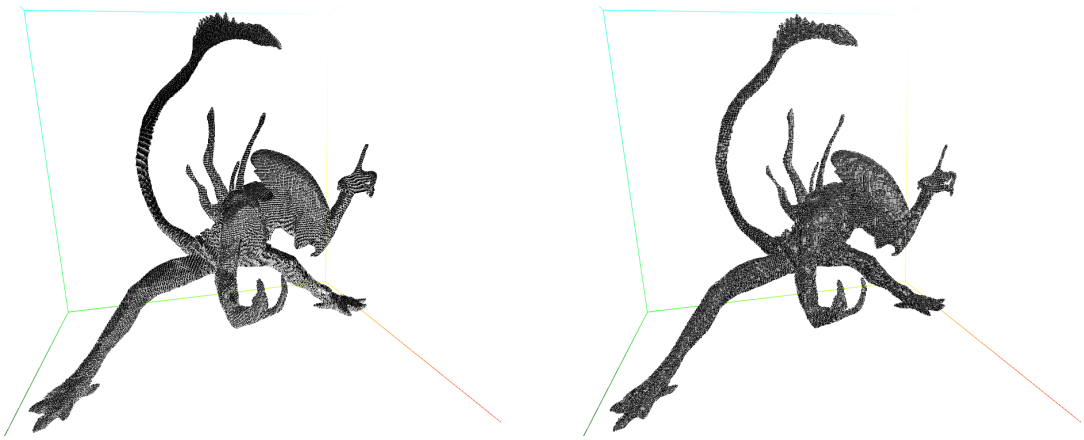
small grids, and always produces reasonable but not very precise model; as a contrast, the octree-based method is less stable on the model quality, but its running time grows much slower than the regular grid method. As shown in Table. 1 and 2, the octree method performs much better in both memory and time comsumption in cases high precision model is needed(Figure. 4).

Model	Voxel Size	16	8	4	2	1	0.5
Alien	# Voxels	194	864	3856	13752	43760	210645
	Running Time (s)	5.67	6.71	9.41	20.99	90.51	633.75
Dinosaur	# Voxels	1603	6318	24587	95781	387882	-
	Running Time (s)	8.62	10.69	24.68	105.12	617.13	-
Predator	# Voxels	4139	16248	63097	245151	991687	-
	Running Time (s)	14.05	19.07	54.72	269.511	1732.31	-

Table 1: Performance of regular grid visual hull.

Model	Cutoff Voxel Size	16	8	4	2	1	0.5
	Min. Split Threshold (s)	0.1	0.2	0.3	0.4	0.5	0.6
Alien	# Voxels	102	207	719	2741	11936	58034
	Running Time (s)	8.87	9.63	10.55	12.06	16.16	33.70
Dinosaur	# Voxels	336	911	4182	21593	118995	735338
	Running Time (s)	8.86	9.88	12.63	19.86	53.79	261.38
Predator	# Voxels	1233	3769	16432	88472	558951	-
	Running Time (s)	12.96	15.62	22.41	46.65	201.63	-

Table 2: Performance of octree-based visual hull.



(a) (b)

Figure 4: Comparison of regular grid and octree-based representation: (a) regular grid with *voxel size* = 0.5, #voxels = 188101, running time = 633.75s; (b)octree-based with $cutoff\ voxel\ size = 0.5$, *min split threshold* = 0.8, #voxels = 69383, running time = 43.36s.

The effect of changing voxel size is shown in Figure. 5. We can see that by decreasing the voxel size, the model becomes more and more precise, however the number of voxels required to represent the model increases dramtically.

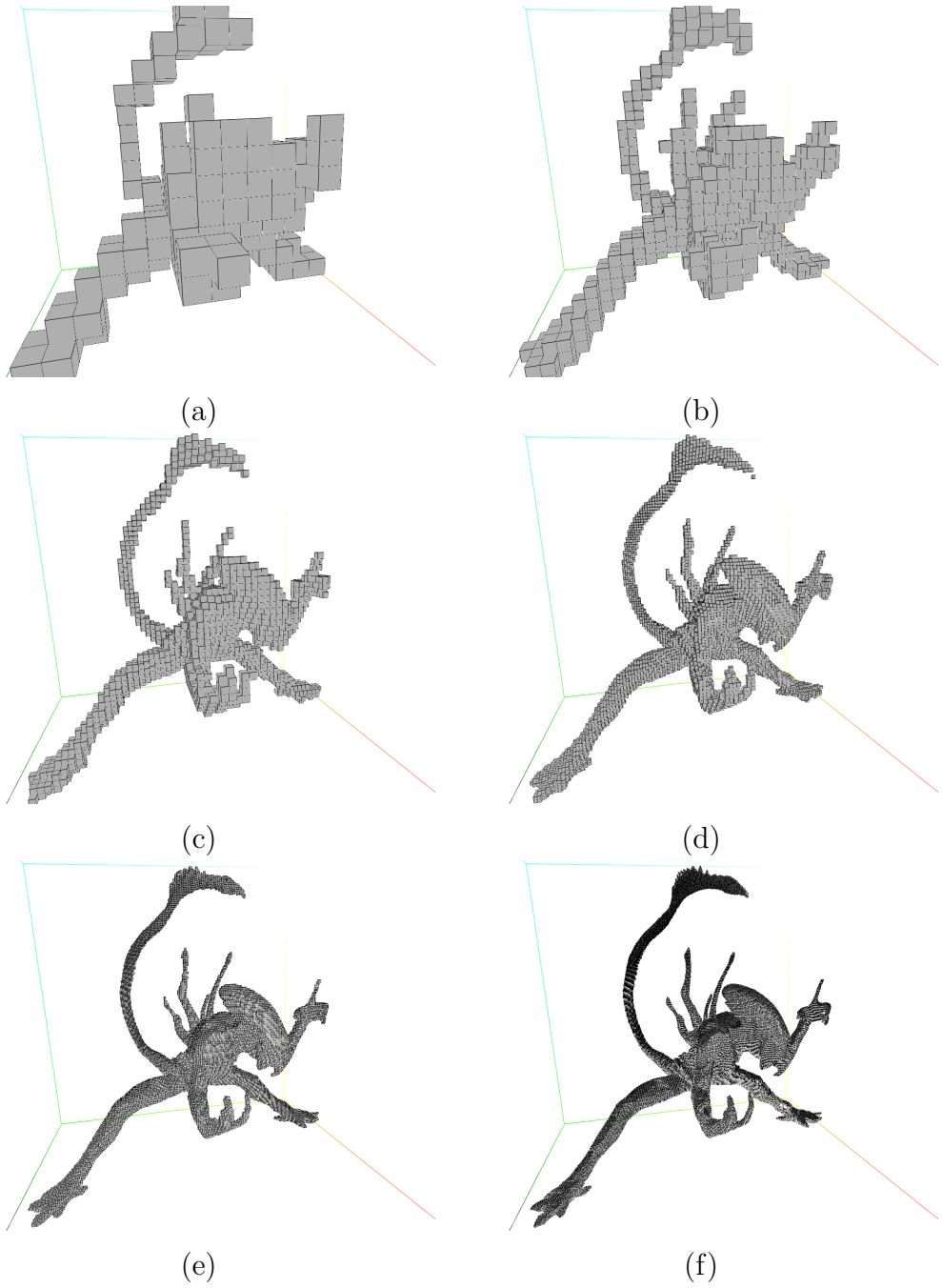


Figure 5: The alien model reconstructed with regular grid visual hull algorithm. From (a) to (f) the voxel size is 16, 8, 4, 2, 1, 0.5, respectively.

The effect of changing coverage threshold is shown in Figure. 6. By increasing the coverage threshold, the model becomes more and more precise, while large scale

components of the model remains being represented by larger cubes. As a result, the number of cubes used to represent the model is much smaller than that by regular grid volume.

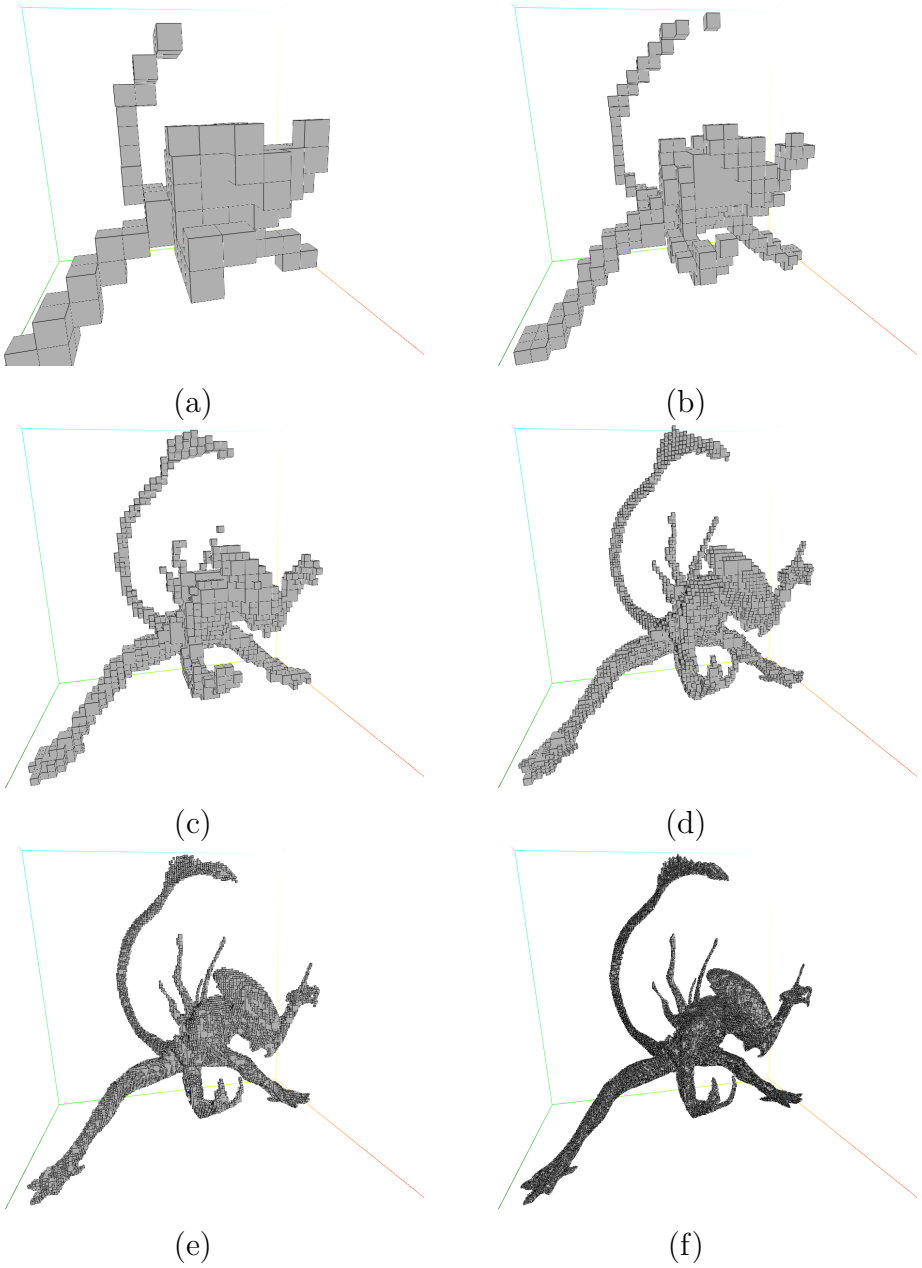


Figure 6: The alien model reconstructed with octree-based algorithm. From (a) to (f) the cutoff voxel size is 0.5, 1, 2, 4, 8, 16, respectively.

Figure. 7 shows more examples of both regular grid visual hull and octree-based visual hull.

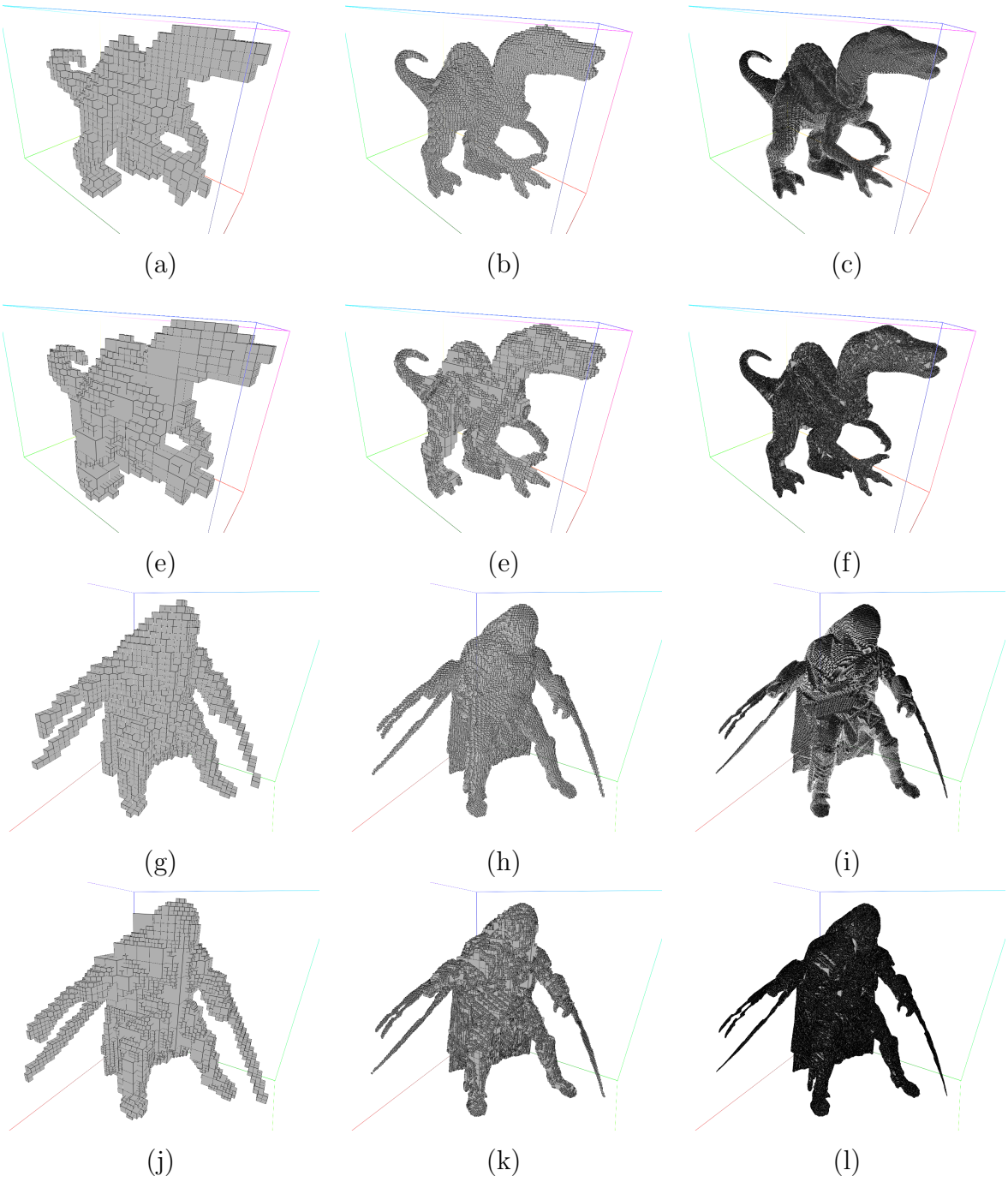


Figure 7: Visual hulls of the dinosaur model and predator model. (a)-(c) and (g)-(i) are reconstructed with regular grid. (d)-(f) and (j)-(l) are reconstructed with octree. The voxel sizes are 16, 4, 1 from left to right.

3.2 Voxel Coloring

The results of voxel coloring algorithm is shown in Figure. 8 and 9. As expected, decreasing voxel size results in more precise models.

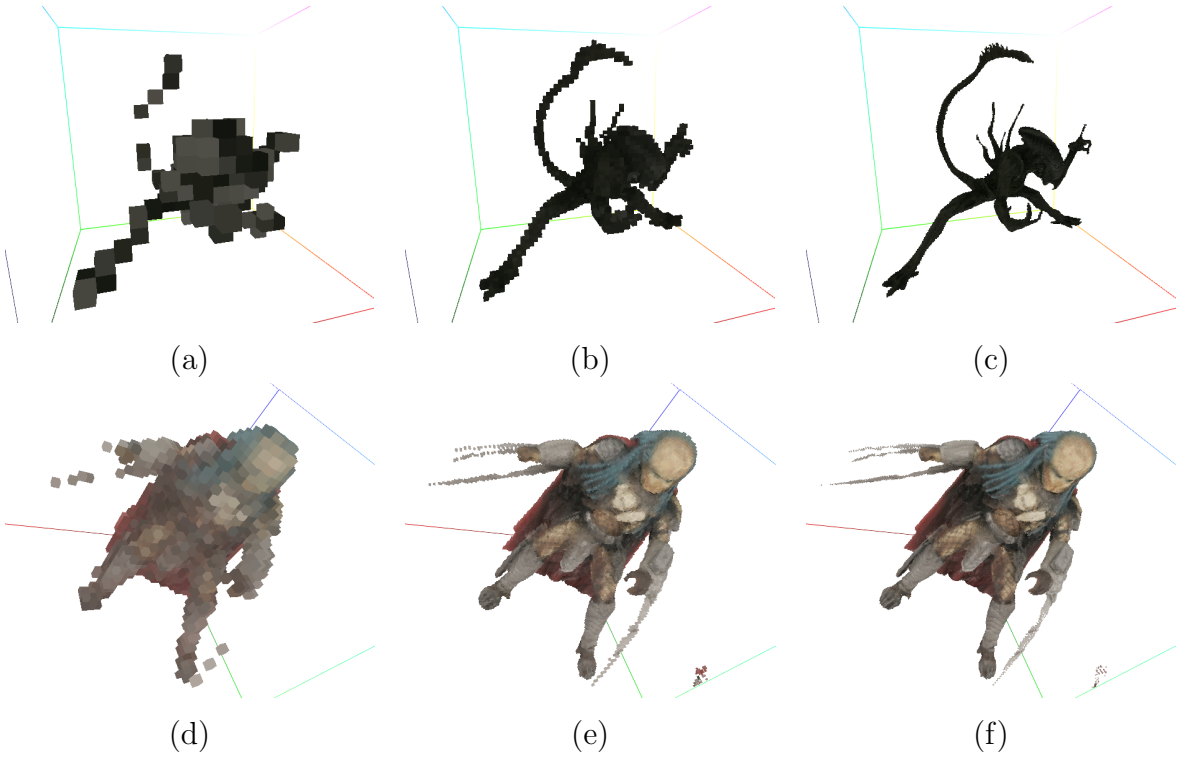
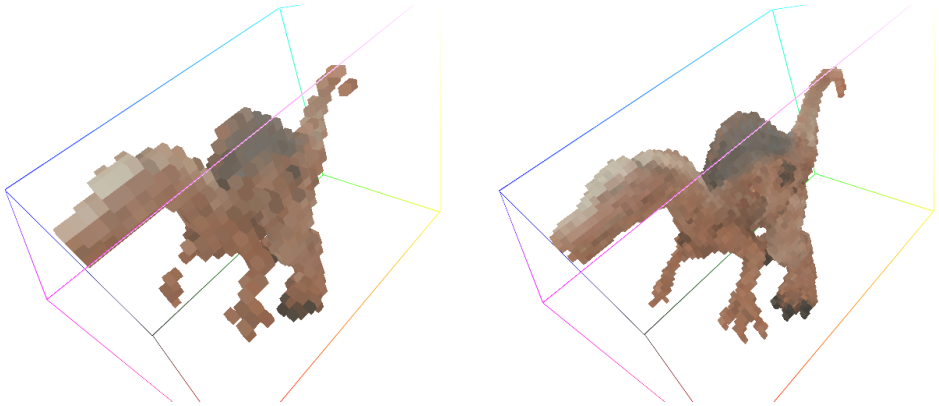


Figure 8: Reconstructed model by voxel coloring. (a)-(c) are reconstructed with voxel size 16, 4, 1; (d)-(f) are with voxel size 16, 4, 2.

Below is the performance data of the voxel coloring algorithm. Comparing to the regular grid visual hull algorithm with the same voxel sizes, the voxel coloring algorithm runs significantly slower due to extra computation. However, the growth pattern of running time remains similar since they have the same time complexity.

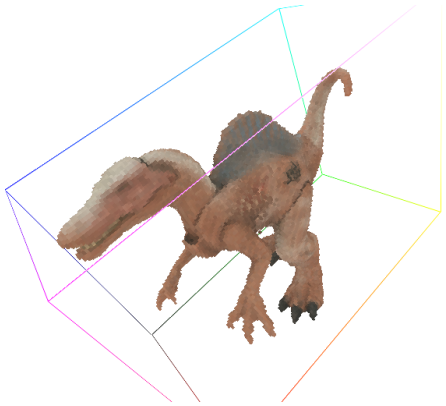
Model	Voxel Size	16	8	4	2	1
Alien	# Voxels	124	678	3738	19056	58134
	Running Time (s)	5.80	12.44	61.08	403.95	3198.84
Dinosaur	# Voxels	1424	11052	68917	131405	415610
	Running Time (s)	10.46	39.60	247.05	1952.64	12725.6
Predator	# Voxels	5772	28618	159736	276535	-
	Running Time (s)	20.171	102.04	710.49	5198.42	-

Table 3: Performance of regular grid voxel coloring.



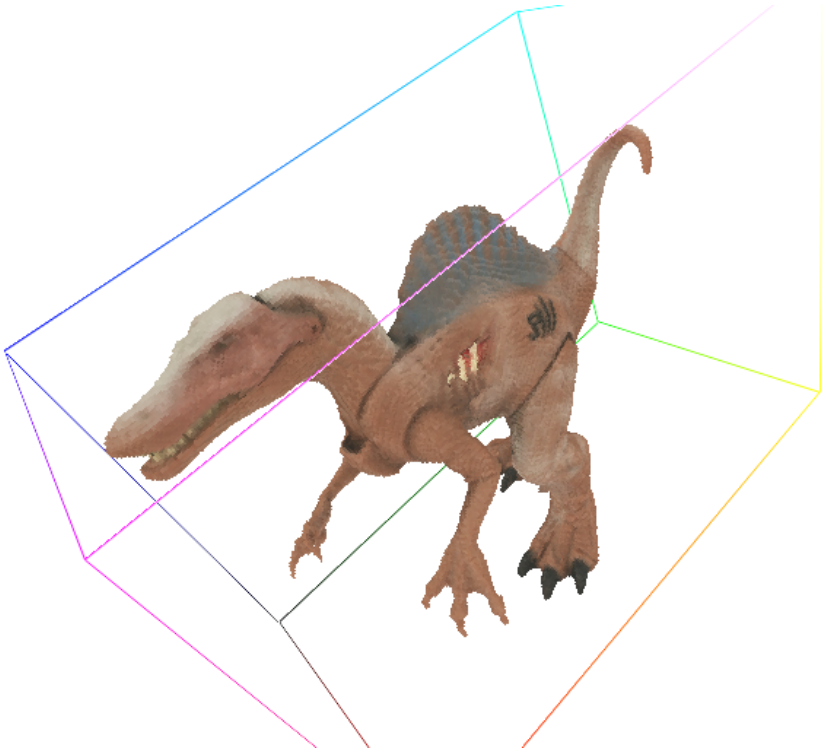
(a)

(b)



(c)

(d)



(e)

Figure 9: Dinosaur model reconstructed by voxel coloring algorithm. (a)-(e) are reconstructed with voxel size 16, 8, 4, 2, 1.

3.3 Discussion

Three algorithms for model reconstruction from mutiple view image sequence are implemented in this project, and expected results are obtained. I also compared the performance of the implemented methods, in the sense of both model quality and time complexity. The statisitcs show that these volumetric approaches have inherent high time complexity, which arises from their spatial division nature. Although theoetically high quality reconstruction is achievable with these methods, the unaffordable time consumption is a major difficulty in practice. Many improvements exist for these methods, among which are accelerating these algorithms through parallelism and combining geometrical approaches to reduce the search space. The reconstruction quality can also be improved by using more accurate computation in voxel coloring algorithm, as well as using more adaptive thresholding scheme. In conclusion, the initial results in this experiments are promising, while to achieve more pratical solutions many potential improvements over the existing system are needed.

References

[1] A. Laurentini. The visual hull concept for silhouette-based image understanding. *IEEE Trans. Pattern Anal. Mach. Intell.*, 16:150–162, February 1994.

[2] Steven M. Seitz and Charles R. Dyer. Photorealistic scene reconstruction by voxel coloring. In *Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*, CVPR '97, pages 1067–, Washington, DC, USA, 1997. IEEE Computer Society.

[3] Steven M. Seitz and Charles R. Dyer. Photorealistic scene reconstruction by voxel coloring. *Int. J. Comput. Vision*, 35:151–173, November 1999.

[4] Richard Szeliski. Rapid octree construction from image sequences. *CVGIP: Image Underst.*, 58:23–32, July 1993.

Manual

Qt library is required to build the system. To compile the source code, please make sure Qt 4 is correctly installed and configurated. The source code is tested ONLY in 64 bit Linux system.

Compile Procedures:

1. tar xvf *package*.tar.gz
2. cd *package*/
3. qmake (or qmake-qt4)
4. make

Usage: The program provides a GUI for convenience.

NOTICE A consistent set of description file and image sequence, together with camera parameters file and contour file, is needed for the program to execute correctly. A typical setting is as follows:

$$
\begin{aligned}
&./alien.des \\
&./data/alien/0000.jpg \\
&./data/alien/0001.jpg \\
&\vdots \\
&./data/alien/0023.jpg \\
&./data/alien/camera.txt \\
&./data/alien/contour.txt
\end{aligned}
$$

The description file for the above setting is included in the submission.

- To reconstruct a model, first select the reconstructor and specify all the parameters needed for that reconstructor, then load the description file (.des file) of the model from the "File" menu.
- To view a reconstructed model, simply load the model file (.vxl file) from the "File" menu.