Université de Montréal

Multi-View 3D Reconstruction Using Virtual Cameras

by

Marc Racicot

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en informatique

Juillet, 2015

To my children Justine and Victor.

RÉSUMÉ

Cette thèse s'intéresse à la reconstruction d'un modèle 3D à partir de plusieurs images prises autour d'un objet. Le modèle 3D est élaboré avec une représentation hiérarchique de voxels sous la forme d'un octree (représentation par une arborescence à huit descendants). Pour ce faire, un cube englobant le modèle 3D est calculé à partir de la position des caméras. Ce cube contient tous les voxels et il est aligné selon le système de référence des caméras. Ce cube définit aussi la position de caméras virtuelles qui seront alignées sur chacune des faces du cube.

Par la suite, le modèle 3D est initialisé par la détection d'une enveloppe convexe qui se base sur la couleur uniforme du fond des images. Cette enveloppe permet de creuser la périphérie du modèle 3D. Pour les voxels étant toujours actifs, un coût est calculé pour évaluer la qualité de chaque voxel à faire partie de la surface de l'objet. Ce coût tient compte de la similarité des pixels provenant de chaque image associée à la caméra virtuelle. Le coût est aussi pondéré en fonction de l'angle de la caméra ayant généré chaque image par rapport à l'angle de la caméra virtuelle.

Finalement et pour chacune des caméras virtuelles, une surface est calculée basée sur le coût en utilisant la méthode de SGM (semi-global matching). La méthode SGM tient compte du voisinage lors du calcul de profondeur et cette thèse présente une variation de la méthode pour tenir compte des voxels précédemment exclus du modèle par l'étape d'initialisation ou de creusage par une autre surface. Par la suite, les surfaces calculées sont utilisées pour creuser et finaliser le modèle 3D.

Cette thèse présente une combinaison innovante d'étapes permettant de créer un modèle 3D basé sur un ensemble d'images existant ou encore sur une suite d'images capturées en série pouvant mener à la création d'un modèle 3D en temps réel.

ABSTRACT

This thesis concentrates on the reconstruction of a 3D model from multiple images taken around an object. The 3D model is built with a hierarchical representation of voxels using an octree (tree representation with eight children per node). In order to achieve this, a cube surrounding the object is calculated from the camera's positions. This cube contains all the voxels and it is aligned with the camera's reference system. The cube also defines the position of the virtual cameras which are aligned to the faces of the cube. After that, the 3d model is initialised with a detection of the visual hull that is based on the uniform color of the images' background. This visual hull information is used to pre-carve the 3D model. Then, for the voxels still active, a cost is calculated to evaluate the quality of each voxel as being on the surface of the object. This cost takes into account the similarity of the pixels from each images associated to a virtual camera. The cost is also adjusted to take the angle of the camera, with regards to the virtual camera, into account. Finally a surface is calculated for each virtual camera based on the voxel cost and by using the SGM method (semi-global matching). The SGM method takes the surrounding voxels into account when calculating the depth and this thesis presents a variation to this method where we take the previously excluded voxels into account. The excluded voxels coming from the initialisation step or from the carving done by another virtual camera. The resulting surface is used to carve the voxel representation. This thesis presents an innovative combination of steps leading to the creation of a 3D model from a set of existing images or from a series of images capture one after another leading to a real-time creation of a 3D model.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to express my gratitude to my supervisor Sébastien Roy for the useful comments, remarks and engagement through the learning process of this master thesis. I would also like to thank Lisa Day, my son Victor, and my mother Micheline for helping me correcting my thesis. Finally, I would like to thank my loved ones, who have supported me throughout the entire process.

NOMENCLATURE

η Quality factor of an image.

$\bar{\mathbf{x}}$ Homogeneous coordinate$(x, y, z, 1)^\top$

$\mathbf{A}_{level,\theta,s}$ Matrix converting a viewpoint's position into a world coordinate

$\mathbf{B}_{level,\theta}$ Matrix converting a viewpoint's position into a position in the reference viewpoint.

$\mathbf{n}$ maximum level reached by the octree, $2^{\mathbf{n}}$ voxels per side for a total of $2^{3\mathbf{n}}$ voxels.

$\mathbf{P}_{\mathbf{f},\mathbf{c},\mathbf{r},\mathbf{t}}$ Matrix converting a world position into a camera screen position.

$\overleftarrow{\mathbf{b}}$ Bounding box vector.

$\overleftarrow{\mathbf{c}}$ Camera vector.

$\overleftarrow{\mathbf{v}}$ Virtual camera vector.

ψ_θ List of cameras associated to a viewpoint.

θ Angle of the view-point

Chapter 1

INTRODUCTION

"The goal of multi-view stereo is to reconstruct a complete 3D object model from a collection of images taken from known camera viewpoints" Seitz et al.[2].

In this master thesis, our goal is to tackle the problem of reconstructing a 3D object from multiple images. Because of the inherent ambiguity of this problem, it is not a simple task. Multiple methods exist to process the images into a 3D object. Some of them will create a 3D object from a cloud of points retrieved from correspondences between images [3]. Others will stitch multiple surfaces together to create a volume [4] but this requires alignment and integration into a finished 3D model. The method we propose here uses a volume of voxels to create a model of the 3D object; the advantages are the simplification of the aggregation of surfaces, the easy parallelisation of the processing of surfaces, and the practical hierarchical nature of the octree data structure [5]. Consequently, the main contribution of our approach are the following:

1. The use of a cost function that takes the position of the camera (with regards to the virtual camera position) into account.

2. The modification of the SGM algorithm to take into account previously carved voxel, either from visual hull or a previous surface carving.

3. The simplification of the aggregation of surfaces using virtual cameras. The finish 3d model defines the reference (not the camera positions).

4. The modification of the SGM algorithm to fix the P2 parameter when multiple images are used with a virtual camera.

There are two ways a 3D model can be created. One way is when all the images are available from the start as in a pre-existing dataset. The other way is when the images are captured one after the other (on the fly), like for a real time scanner where the 3D model is refined as more images are captured. In this thesis, the former method is the one used to get the results. There will be some thoughts given to the second method as well.

The steps to create the 3D model are:

1. Calibration and lens aberration correction - chapter 2.

2. Obtain the bounding box of the work area and perform background segmentation (appendix A) which provides a preliminary carving - chapter 3.

3. Calculation of the cameras association and cost function - chapters 4 and 5.

4. Calculation of cost smoothing and surface creation which leads to the final carving - chapter 6.

1.1 related work

A method by Li et al.[6] is using depth map merging for mutli-view stereo reconstruction (MVS). In a first step, the method calculates depth maps from chosen image pairs. Then, highly correlated points are matched between depth maps using a DAISY descriptor. A track is then created that keeps a relation between these matching points. These track are then processed using a bundle optimisation step that will keep reliable tracks, this creates an initial point cloud. A refine steps is then used to increase point's accuracy and calculates normal which will be used to create a mesh of the model. This method is sensible to the accuracy of the depth maps.

Another method by Soremann at al.[7] uses a min-cut/max-flow solution on a graph representation of a volumetric aggregation of dense maps to obtain a watertight 3D model. The dense depth maps is calculated using a plane sweep method and the resulting depth maps are transformed into a graph representation with edge weights coming from the calculated depth maps. The results is the solution of a min-cut/max-flow solution on the weighted graph.

A third method by Schroers et al.[8] tries to find a surface by minimizing an energy function based on the total variation of signed distances from depth maps to the surface position. Depth maps are calculated using a stereo based method. The sign distance is smoothed while optimizing the surface using a gradient descent. This method uses significant processing.

Other methods are provided in the Middleburry reference list [9].

1.2 current work

The main idea of this thesis is to reduce the complexity of surface merging, like we saw in the methods previously described in 1.1 this task can be quite complex. In order to simplify the surface merging, a global reference frame on which a 3D model will be reconstructed is created. The information from multiple images is used to carve the voxels by using the depth coming from the calculated surfaces. This carving is occurring on multiple view-points around the reconstructed 3D model. These view-points are, in fact, virtual cameras facing the sides and edges of a cuboid volume that sits on the world reference. The surfaces are created following a smoothing step performed by a semi-global method [1].

The use of voxels can lead to a complex problem if the resolution is high because it requires a lot of resources (cpu and memory). By using a hierarchical method and by only processing areas where the object is present (as detected by preliminary steps) it is possible to reduce the resource usage. It is then possible to achieve better

performance than $O(\beta^3)$ where β is the number of voxels per axes at the maximum voxel level.

Chapter 2

CAMERA

Since 3D reconstruction relies on image only, it is important to accurately model the camera that took these images. This chapter highlights the basic concepts of camera modeling.

2.1 Camera matrix

The camera matrix is the combination of the external and internal parameters of the camera. The external parameters are the physical displacements (translations and rotations) with regards to a global reference system that will bring the camera at the position and direction from which the image is taken. The internal parameters represent the projection of a point in the scene, expressed in the camera frame of reference, to a point on the camera sensor.

The following matrix (2.1) contains the translation $\mathbf{t}$, the rotation $\mathbf{r}$, the center of the camera sensor $\mathbf{c}$ and the focal distance $\mathbf{f}$. More information can be found in [10, p. 49].

$$\mathbf{P}_{\mathbf{f},\mathbf{c},\mathbf{r},\mathbf{t}} = \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

When projecting a world point onto the screen, the depth information (z) is lost in the conversion; only the x and y position on the screen remains.

$$(u, v, w, 1)^\top \;=\; \mathbf{P_{f,c,r,t}} \;\cdot\; (x_w, y_w, z_w, 1)^\top$$

Where $(u, v, w)^\mathsf{T}$ is a 2D projective representation of the projected point on the screen. The Euclidian sensor coordinate (x_s, y_s) is obtained from (u, v, w) as:

$$(x_s, y_s) \;=\; \left(\frac{u}{w}, \frac{v}{w}\right)$$

The camera center position (the camera origin) can be retrieved from the inverse of the matrix $\mathbf{P_{f,c,r,t}}$ by taking the last column $(O_x, O_y, O_z)^\top$ of $\mathbf{P^{-1}}$.

$$\mathbf{P^{-1}} = \begin{bmatrix} \dots & \dots & \dots & O_x \\ \dots & \dots & \dots & O_y \\ \dots & \dots & \dots & O_z \\ \dots & \dots & \dots & 1 \end{bmatrix}$$

2.2 Lens correction

Most of the time, if not all the time, the lens of the camera suffers from certain aberrations that need to be corrected before a pixel position can be used to calculate a cost value. These aberrations can be corrected by taking into account the distance of the pixel position from the center of the aberration and by calculating a Taylor approximation based on that distance. Barrel and pincushion aberrations [10, p. 52] can be corrected like this. More severe or nonlinear aberrations can be corrected with a look-up table that maps every pixel position to the correct position.

2.3 Limitations

Some constraints and limitations that are not addressed by this thesis are listed here:

2.3.1 Lack of texture (ill-posed problem)

When a region of the image does not contain any texture, it is difficult to match it to other image and to retrieve a valuable depth. The smoothing performed by SGM (chapter 6) on the cost will propagate the depth from the boundaries of the texture less region but there could be some improvements.

2.3.2 Non-Lambertian surfaces

The surface of the object used in this thesis are assumed to be Lambertian (no reflection). This simplifies the evaluation of a cost function because a point is guaranteed to have the same color when viewed from different angles. In order to cope with non-Lambertian surface, a method of cost calculation like Census[11] would have to be used. The Census matching cost is described has having the highest radiometric robustness in the cost methods comparison done in [12].

2.3.3 Translucency

Translucency surface is adding a lot of complexity to the reconstruction of a 3D model. This could be the subject of a Ph.D. and it is not handled in this thesis.

2.3.4 Image noise

Noise is always present in the captured image. A direct impact of noise is the variations it causes on the cost values and the erroneous depth retrieved from that. Fortunately, the smoothing step of SGM and the combination of multiple images reduce the impact of the noise.

2.3.5 Imperfect calibration

When the calibration is not precise, unrelated pixels are matched together and will lead to imprecise results.

2.3.6 Differences in exposures, white balance

Differences in camera parameters from one pose to another is not handled in this thesis. Some preliminary steps like histogram equalization and color corrections would be needed to be able to calculate a cost value from multiple images.

2.4 Expectations

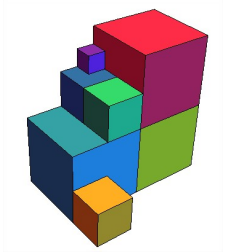
Even with all these restrictions, the method developed in this thesis gives good results and is usable in many context.

Chapter 3

MODEL INITIALIZATION

In this chapter we will see how to calculate a bounding box based on the cameras position. We will also see the octree construct that is used to describe the 3D model. Finally, we will see how to use a visual hull to initialize the octree with a crude representation of the 3D model.

3.1 Octree



An octree is a hierarchical structure used to divide a volume in multiple subdivisions. Each subdivision can be recursively splitted into eight subdivisions of equal size. All subdivisions are voxels. The first division starts at level 0 and the last one ends at level n. There is only one voxel at level 0 and it is the size of the bounding box. All the other subdivisions are completely filling the voxel of the lower level[1] that they are associated to[2]. The process of dividing the volume goes on with the level increasing until a sufficient voxel resolution is reached. To specify a voxel, the position, level and viewpoint angle (x,y,z,level,θ) are needed.

Each entry in the octree defines whether the voxel is inside the volume (white), outside the volume (black) or partially inside/outside (gray) in which case another level of subdivision is needed and/or present. Octree was demonstrated in [13] to reconstruct an object in a process called shape from silhouette.

Octree representation is relatively simple since it only has 3 states (empty, full,

[1] The lower the level is, the bigger the voxel size. Voxels at higher level are smaller in size than voxels at lower level

[2] See Appendix B for algorithm details

partial) and each level of the octree requires $O\left(2^{level}\right)$ of memory. For example, an octree using 1 byte per voxel with 9 levels will use 147 Megabytes (equation 3.1). This amount can be reduced to 37 Mega-Bytes if we use 2 bits per entry.

$$OctreeSize(\mathbf{n}) = \sum_{k=0}^{\mathbf{n}} 2^{3(\mathbf{n}-k)} \tag{3.1}$$

Octree representation can help reduce cpu and memory usage when a background thresholding can be made on the images. If the object can be separated from the background, then it is possible to create a convex hull by marking off some voxels of the object. Voxels identified as not being part of the object will not be used in the cost calculation and the SGM smoothing. Furthermore, if many voxels are marked off at a lower level (after percolation) it means a significant volume of the bounding box does not need to be processed. Percolation is the process by which all lower levels are updated from the higher level. To gain insight on octree related algorithms see Appendix B.

An advantage of using an octree construct is coming from its hierarchical nature. Some processing needs to be done at a high voxel resolution (higher level), for instance the cost calculation. Other processing can be done using only a lower resolution, for instance collision avoidance. Figures 3.1a and 3.1b show two representations of the same 3D model at level 6 and 9. We see that even if the resolution is lower, the object is still recognizable and useable.

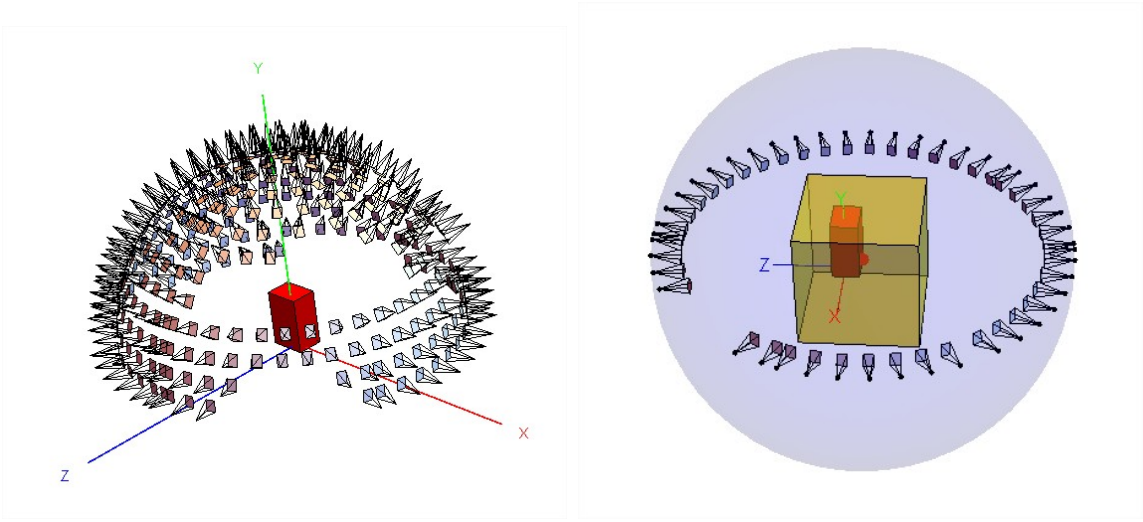
(a) 3D model at voxel level 6

(b) 3D model at voxel level 9

Figure 3.1: 3D model at different voxel levels

3.2 Bounding box

In order to infer a bounding box containing all the voxels, the position of the cameras are used to calculate a single point $\overleftarrow{\mathbf{c}}$. This point $\overleftarrow{\mathbf{c}}$ will be the center of a sphere of radius μ and the bounding box will be calculated from this sphere[3] since they have the same center point. Figure 3.2a illustrates a bounding box surrounded by the poses of each camera.



(a) Camera poses

(b) Inferred bounding box(yellow) englobing the voxels

Figure 3.2: Bounding box

[3] This assertion is only valid if the camera poses are made around the objects and are pointing toward the object.

In order to determine this bounding box center point $\overleftarrow{\mathbf{c}}$, we try to find $\overleftarrow{\mathbf{c}}$ that minimized:

$$(\rho)\ \mu\ +\ (1-\rho)\ \epsilon \tag{3.2}$$

where

$$\rho = 0.2$$

and

$$\mu = \frac{1}{n}\sum_{i=1}^{n} \left\| \overleftarrow{\mathbf{o_i}} - \overleftarrow{\mathbf{c}} \right\|$$

and

$$\epsilon = \sum_{i=1}^{n} \left\| \left(\overleftarrow{\mathbf{c}} + \mu \left(\frac{\overleftarrow{\mathbf{o_i}} - \overleftarrow{\mathbf{c}}}{\left\| \overleftarrow{\mathbf{o_i}} - \overleftarrow{\mathbf{c}} \right\|} \right) \right) - \overleftarrow{\mathbf{o_i}} \right\|$$

with $\overleftarrow{\mathbf{c}}$ initialised to:

$$\overleftarrow{\mathbf{c}}\ =\ \frac{1}{n}\sum_{i=1}^{n} \overleftarrow{\mathbf{o_i}}$$

Minimizing equation 3.2 will minimized the error ϵ and the radius length μ. This makes the bounding box closer to the cameras. The error for a given center point $\overleftarrow{\mathbf{c}}$ is calculated by equation 3.2. Multiplying the radius μ by a unit vector leaving the bounding box's center, in direction of the camera's center, gives an error vector. The vector leaving the bounding box's center and reaching the camera position $\overleftarrow{\mathbf{o_i}}$ gives a camera vector. The error is the sum of the norm of the difference between the error vector and the camera vector. The factor ρ is used to reduce the weight of the radius in the minimization. A value of $\rho = 0.2$ was used with good results. But other values will also give good results because this parameter is not sensible, it is

only used to keep the radius length μ short which in turn will keep $\overleftarrow{\mathbf{c}}$ close to the cameras position $\overleftarrow{\mathbf{o_i}}$.

From the calculated radius μ we can evaluate the side of the bounding box with the following equation:

$$s = \alpha \sqrt{\frac{4\mu^2}{3}} \tag{3.3}$$

This equation gives the biggest bounding box that fits inside a sphere of radius μ. A ratio α is used to reduce the size of the bounding box because it is not necessary to have the bounding box touching the camera; it is more important to have a smaller bounding box closer to the object than to have a big bounding box containing the object because the number of voxels increases with the bounding box size. The use of this reduction factor with a value of 0.5 has demonstrated good results on the test datasets (Middlebury). The parameter α could be ignore with a value of 1.0 and the reconstruction will give the same results except for the voxel resolution that could increase. This parameter is a way to reduce the resources usage (memory and cpu) when working on pre-defined dataset.

Figure 3.2b shows the calculated bounding box[4] in yellow. The red box represents the dataset suggested bounding box. We see that the red bounding box is included inside the yellow bounding box. The calculated bounding box is quite large compared to the suggested one and it will require further processing in order to reduce its size to be closer to the real object. The octree along the convex hull will perform this duty by reducing the number of voxels to be processed by carving off the volume.

3.3 Serial processing of camera poses

In the case where camera poses are received one after the other, like when a camera is free handed around an object, each pose is processed as they come. The calculated radius and bounding box center will change along the way. The more camera positions

[4] Middlebury's temple ring datasets, 47 poses

there are, the more precise the bounding box. It is better to wait for at least 3 camera poses before starting to calculate a bounding box and processing the voxels. Once a first bounding box has been defined we get the center **c** and the size s of the bounding box. Two scenarios can happen when new camera poses are taken into account.

In the following 2 scenarios, the minimum voxel size is used to align the bounding box position and size changes. The minimum voxel size is defined by the maximum level reached by the octree and by the size of the bounding box's side s. The minimum voxel size is discussed in section 3.7. It is important to note that when applying changes to the center position or size of the bounding box, the octree construct must be percolated.

3.3.1 Bounding box position change

When the calculated position of the bounding box center changes, the new position must be aligned to the smallest voxel size. This is done by rounding the position along the axes x,y, and z to the smallest voxel size or $\frac{s}{2^n}$.

The size of the bounding box must also be adjusted. The old and new positions of the bounding box corners are used to calculate a new bounding box size. For each axis we choose the coordinate that is creating the biggest bounding box.

$$\mathbf{Sx} = Max\left(OldPosX, NewPosX\right) - Min\left(OldPosX, NewPosX\right) \tag{3.4}$$

Equation 3.4 calculates the size of the bounding box side for the X axis. The same thing is done for the Y and Z axes and the new size is calculated by equation 3.5.

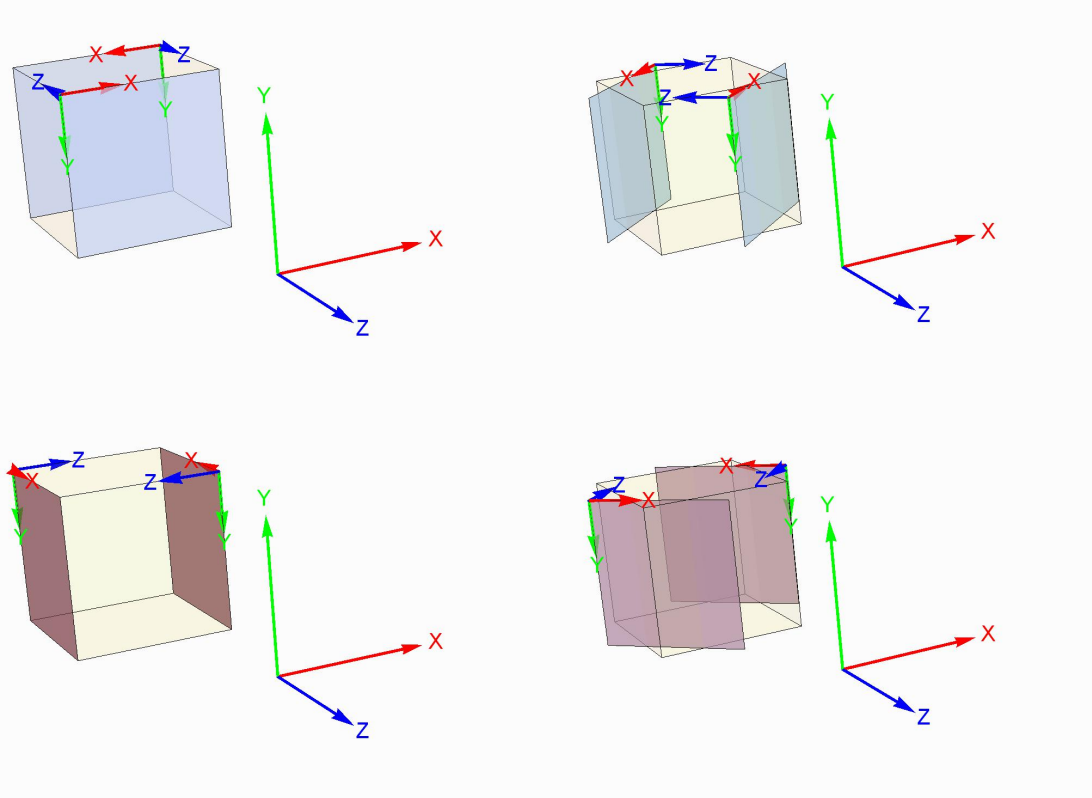
$$\mathbf{S} = Max\left(Sx, Sy, Sz\right) \tag{3.5}$$

3.3.2 Bounding box size increase/decrease

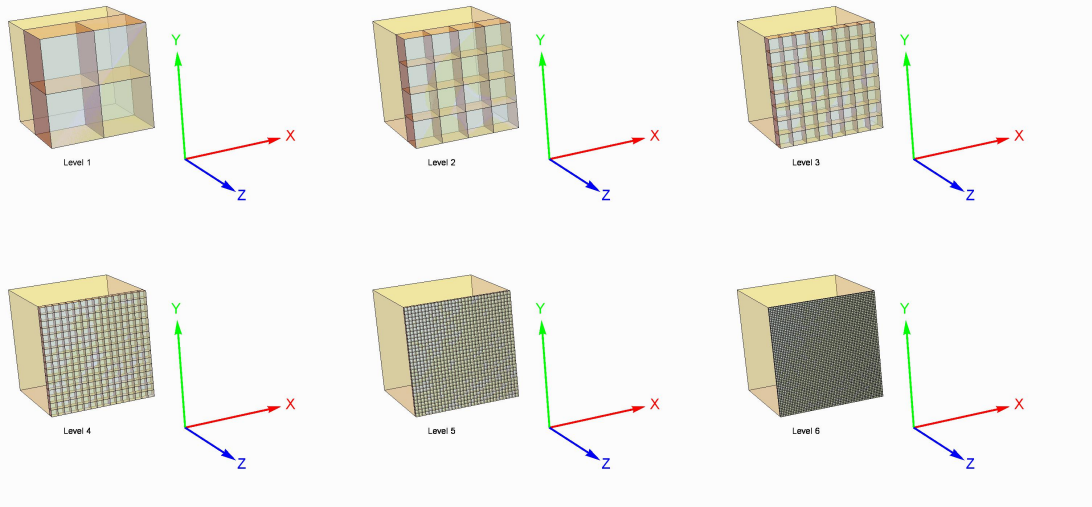
When the bounding box size changes, Equation 3.5 will give the new size.

3.4 Coordinates systems

When the camera poses (the camera matrix) are provided it is relatively straightforward to find a valid bounding box like we saw in section 3.2. The camera positions provide direct information on the orientation of the bounding box because of the reference frame used by the external camera parameters. Since there are no guarantees that the object under reconstruction is at the center of the camera's reference frame, another reference frame, which will be called the bounding box reference frame, is built from the bounding box center calculated by Equation 3.2. The orientation is simply the natural axes obtained from the bounding box coordinates which are the same as the camera's coordinate system (camera external parameters). On this coordinate system, viewpoints are defined by an angle θ and they defines multiple planes on which the surfaces will be calculated. Figure 3.3a shows the world reference system and the coordinate system for each viewpoints. Figure 3.3b shows different octree levels and the voxel separation for the first depth plane. This is discussed in more detail in section 3.7.



(a) These figures shows the coordinate systems of each viewpoint and the world reference. 2 reference-frames are shown on each bounding box (to save space). The z axis (depth) is always pointing inward.



(b) Multiple octree levels

Figure 3.3: Octree axes and levels

3.5 Coordinates conversion

The method to reconstruct the 3D object is to create surfaces from different viewpoints. These viewpoints have independent coordinate system with the origin at the upper left corner and the depth going inward (toward the object). All coordinate systems respect the right hand rule. Figure 3.4 shows the viewpoints and their axes.

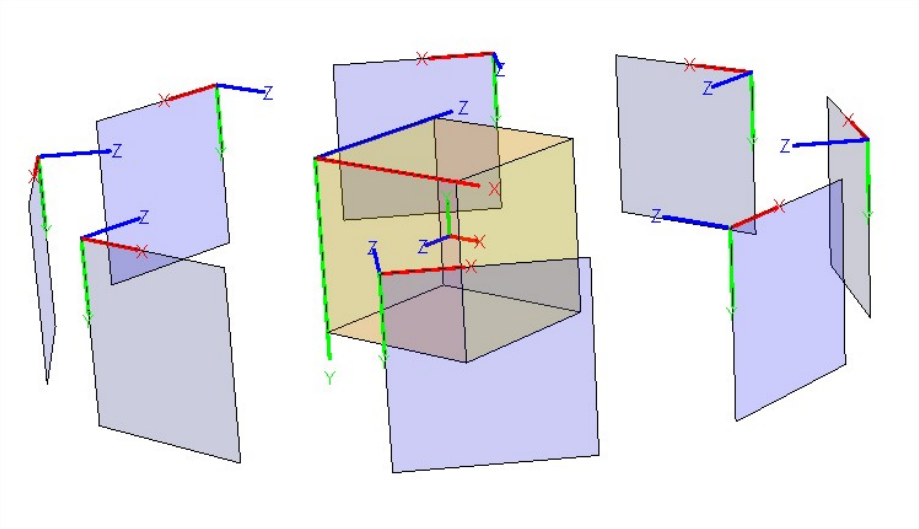


Figure 3.4: This figures shows an exploded view of the viewpoints and their axes.

To convert a point $\bar{\mathbf{x}}_{viewpoint}$ from a viewpoint coordinate system to a world coordinate system $\bar{\mathbf{x}}_{world}$, multiple transformations need to be applied to that point, represented as a homogeneous vector $(x, y, d, 1)^\top$. Equation 3.6 shows the operations needed to accomplish this transformation. The details of the inner operations are defined in equations 3.7 to 3.11.

$$\bar{\mathbf{x}}_{world} = [\ \mathbf{T}_c\ ] \cdot [\ \mathbf{R}_{vp}\ ] \cdot [\ \mathbf{T}_{bb}\ ] \cdot [\ \mathbf{R}_z\ ] \cdot [\ \mathbf{S}_v\ ]\ \bar{\mathbf{x}}_{viewpoint} \tag{3.6}$$

$$\mathbf{T}_c = \begin{bmatrix} 1 & 0 & 0 & c_x \\ 0 & 1 & 0 & c_y \\ 0 & 0 & 1 & c_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.7}$$

$$\mathbf{R}_{vp} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.8}$$

$$\mathbf{T}_{bb} = \begin{bmatrix} 1 & 0 & 0 & \left(-\frac{s}{2} + \frac{s}{2^{level+1}}\right) \\ 0 & 1 & 0 & \left(\frac{s}{2} - \frac{s}{2^{level+1}}\right) \\ 0 & 0 & 1 & \left(\frac{s}{2} - \frac{s}{2^{level+1}}\right) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.9}$$

$$\mathbf{R}_z = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\pi & -\sin\pi & 0 \\ 0 & \sin\pi & \cos\pi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.10}$$

$$\mathbf{S}_v = \begin{bmatrix} \frac{s}{2^{level}} & 0 & 0 & 0 \\ 0 & \frac{s}{2^{level}} & 0 & 0 \\ 0 & 0 & \frac{s}{2^{level}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.11}$$

Equation 3.7 performs a translation to take the center position of the bounding box into account. Equation 3.8 rotates the view-point plane around the y axis. The angles for the rotation are: $0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}, \pi, \frac{5\pi}{4}, \frac{3\pi}{2}$, and $\frac{7\pi}{4}$ radian. Equation 3.9 translates the coordinate axes from the middle of the bounding box to the upper left voxel middle position, taking into account the level of the voxel. Two voxels at position (0,0) but

at different levels will not have the same world coordinate. Equation 3.10 rotates the coordinate axes to have the Z-axis pointing inward. This allows the z values to always be positive when the surface is calculated. Equation 3.11 scales the coordinate from 2^{level} voxels (voxel resolution) to s (bounding box size).

Equations 3.12 shows all operations together. It is important to note that these operations are only valid for viewpoint as defined in figure 3.3a (on the right), which are on a horizontal plane. Having viewpoints on top or at angles of the horizontal plane would require another rotation in the equations.

$$m = -\frac{s}{2} + \frac{s}{2^{(level+1)}}$$

$$p = \frac{s}{2} - \frac{s}{2^{(level+1)}}$$

$$\mathbf{A}_{level,\theta,s} = \begin{bmatrix} \frac{s \cdot \cos\theta}{2^{level}} & 0 & -\frac{s \cdot \sin\theta}{2^{level}} & c_x + m \cdot \cos\theta + p \cdot \sin\theta \\ 0 & -\frac{s}{2^{level}} & 0 & c_y + p \\ -\frac{s \cdot \sin\theta}{2^{level}} & 0 & -\frac{s \cdot \cos\theta}{2^{level}} & c_z + p \cdot \cos\theta + m \cdot \sin\theta \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.12}$$

$$\bar{\mathbf{x}}_{world} \;=\; \mathbf{A}_{level,\theta,s} \;\cdot\; \bar{\mathbf{x}}_{vp} \tag{3.13}$$

Equation 3.13 converts a coordinate from the viewpoint's coordinate system to the world coordinate system. The matrix is defined by the rotation angle θ, the number of voxels 2^{level} and the size of the bounding box s.

3.6 From one viewpoint to reference viewpoint

There is only one octree construct that is maintained and it is associated with the viewpoint at angle $\theta = 0$ which is the reference viewpoint. To assess the state of a voxel from another viewpoint, a transformation of coordinate needs to be done.

We saw that a voxel can be translated to a world coordinate with equation 3.13. If we invert that equation, we can get the viewpoint from the world coordinate. The equation is repeated here but the position on the viewpoint is annotated with the viewpoint angle θ and θ_2.

$$\bar{\mathbf{x}}_{world} \;=\; \mathbf{A}_{level,\theta,s} \;\cdot\; \bar{\mathbf{x}}_{vp_\theta}$$

$$\bar{\mathbf{x}}_{vp\theta_2} \;=\; \mathbf{A}^{-1}{}_{level,\theta_2,s} \;\cdot\; \bar{\mathbf{x}}_{world}$$

$$\bar{\mathbf{x}}_{vp\theta_2} \;=\; \mathbf{A}^{-1}{}_{level,\theta_2,s} \;\cdot\; \mathbf{A}_{level,\theta,s} \;\cdot\; \bar{\mathbf{x}}_{vp_\theta}$$

$$\bar{\mathbf{x}}_{vp\theta_2} \;=\; \mathbf{B}_{level,\theta} \;\cdot\; \bar{\mathbf{x}}_{vp_\theta}$$

When the angle θ_2 is the reference viewpoint, the angle is 0 and the equations simplify to a new matrix: the B matrix.

$$\mathbf{B}_{level,\theta} = \begin{bmatrix} \cos\theta & 0 & -\sin\theta & -\frac{1}{2}\left(2^{level}-1\right)\left(\cos\theta-\sin\theta-1\right) \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & -\frac{1}{2}\left(2^{level}-1\right)\left(\cos\theta+\sin\theta-1\right) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.14}$$

As a test, if we take θ to be 0, which means we are converting a voxel coordinate

from viewpoint 0 to the same viewpoint (so we do not expect to have any changes in the position) we will get from Equation 3.14 a B matrix equal to the identity matrix. It is interesting to note that the B matrix does not depend on s, it only depends on the angle and octree level.

3.7 Octree level and bounding box

The relation between the pixel and the voxel is made through the camera matrix P from Equation 2.1 on page 5 and the world position matrix $\mathbf{A}_{level,\theta,s}$. This can be seen in Equation 3.15 and figure 3.5 shows this graphically.

$$\bar{\mathbf{x}}_{screen} \;=\; \mathbf{P}_{\mathbf{f,c,r,t}} \;\cdot\; \mathbf{A}_{level,\theta,s} \;\cdot\; \bar{\mathbf{x}}_{vp} \tag{3.15}$$

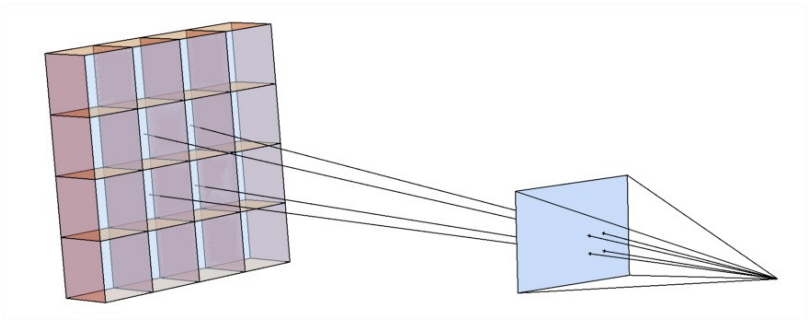


Figure 3.5: This figures demonstrates the projection of 4 voxels onto one camera.

By reversing the previous equation, we obtain a relation that will convert a screen position into a voxel position, as can be seen in equation 3.16.

$$\bar{\mathbf{x}}_{vp} \;=\; \mathbf{C} \;\cdot\; \bar{\mathbf{x}}_{screen} \tag{3.16}$$

With:

$$\mathbf{C} = \mathbf{A}^{-1}{}_{\mathbf{level},\theta,\mathbf{s}} \;\cdot\; \mathbf{P}^{-1}{}_{\mathbf{f,c,r,t}}$$

If we wanted to calculate the maximum level at which the distance in the image

is 1 pixel for a delta position of 1 voxel, we could use a function that will calculate the level at which the voxel distance is lower or equal to 1, resolving equation 3.17 would provide this information.

$$\left\| \mathbf{C} \cdot (x, y, z, 1)^{\top} - \mathbf{C} \cdot (x+1, y, z, 1)^{\top} \right\| <= 1 \tag{3.17}$$

With the unknown variable being the level, this previous function, while complex, would provide the maximum level to use. Going higher than the maximum level would use sub-pixel information from the images. Unfortunately, this function will be quite big and will depend upon many variables $(\theta, s, CameraMatrix)$ and the x and y position on the screen so it is not easy to represent a simple function resolving this. Nevertheless it is practical to calculate this for a couple of positions, cameras and viewpoints to get a good idea of the maximum level to use. Table 3.1 shows the pixel positions for 2 voxels side-by-side. We see that the greater the level, the closer the pixel position gets and it eventually becomes within 1 pixel of distance at level 10.

Table 3.1 tells us that for a specific camera, viewpoint, position, and bounding box we need to have $2^{10} = 1024$ voxels to achieve sufficient resolution in order to use the pixel value directly in a cost function (discussed in Chapter 5).

3.8 Visual hull

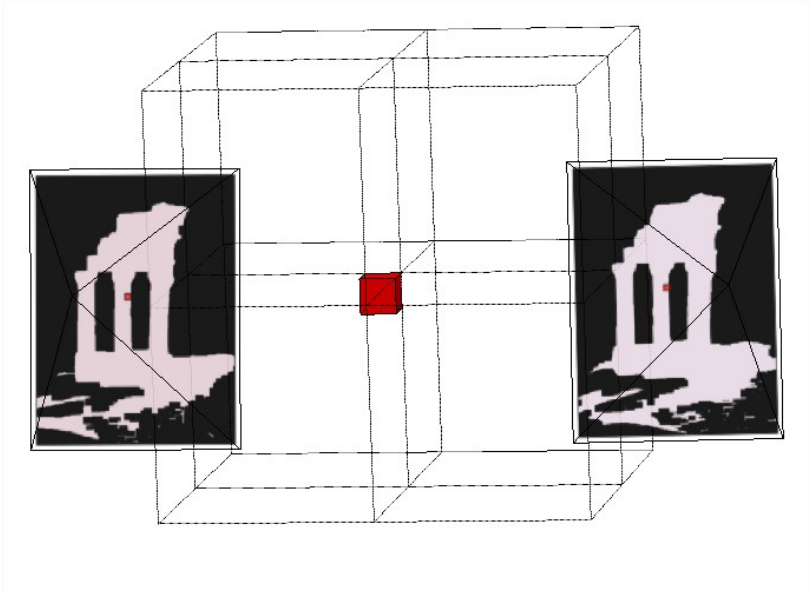
The bounding box defines the external limits for the reconstruction. Finding the bounding box size and position was the first step of the 3D model initialization and was explained in section 3.2. As a second step, we can create a mask by background segmenting the images and then applying this mask to the voxel to create a visual hull[5]. Appendix A explains a method to perform background segmentation on images with uniform background.

[5] some images are well suited for background subtraction, like Middlebury's Temple and Dino

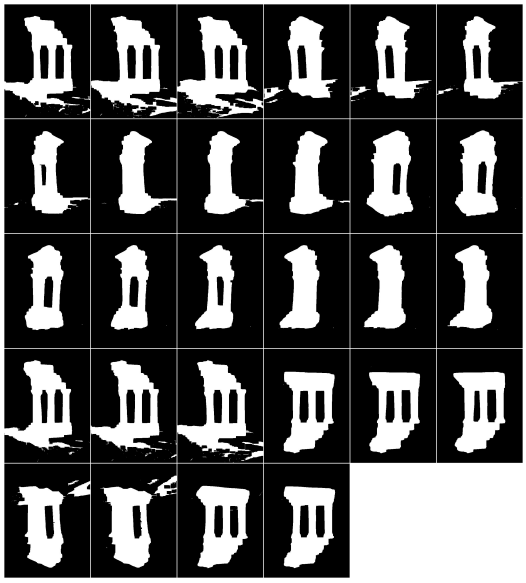
Level	voxel at {x,y}	voxel at {x+1,y}	distance
1	{579.917,-30.1624}	{71.0033,-7.23606}	509.43
2	{436.715,102.385}	{161.256,110.644}	275.58
3	{357.269,175.92}	{213.587,178.994}	143.71
4	{315.33,214.739}	{241.898,215.972}	73.44
5	{293.77,234.695}	{256.643,235.23}	37.13
6	{282.838,244.814}	{264.169,245.06}	18.67
7	{277.333,249.909}	{267.972,250.027}	9.36
8	{274.57,252.466}	{269.883,252.524}	4.69
9	{273.187,253.747}	{270.842,253.775}	2.35
10	{272.494,254.388}	{271.321,254.402}	1.17
11	{272.148,254.708}	{271.561,254.715}	0.59
12	{271.975,254.869}	{271.681,254.872}	0.29

Table 3.1: Pixel positions calculated from 2 voxels at positions (x, y) and $(x + 1, y)$ for multiple levels.

The silhouette provides direct information on whether a voxel is part of the object, part of the background or both (when the bi-linear interpolation on the mask is on a boundary). Projecting each voxel at level n on all the images associated with the view-point and verifying that at least one of the projection is on the background allows to mask off voxels in the octree. The Octree will be populated like this at the maximum voxel resolution and percolation will be used to populate the lower level of the hierarchy. Figure 3.6a shows a bounding box with one big voxel in red projected to 2 cameras. We see that the voxel is not part of the background so the mask will not be modified. Figure 3.6b presents some of the masks that were used to create the visual hull of Figure 3.6c.



(a) The voxel in red is projected to masks. When a voxel falls on the background, the voxel in the Octree is marked as empty.



(b) Some of the binary masks used to create the visual hull of figure 3.6c

(c) visual hull result on Middlebury temple

Figure 3.6: Visual hull processing

Chapter 4

CAMERA ASSOCIATION

Typical mutli-view models provide arbitrary camera viewpoints in order to compute the 3D model. The present method uses pre-defined viewpoints in the reference frame of the reconstructed volume. These viewpoints are the point of view of virtual cameras. The virtual cameras are cameras that do not exists (they did not provide 2D images) but are used to calculate a surface from their view point as if they existed. In order to do that we need to associate real cameras to virtual cameras. Figure 4.1a demonstrates the various viewpoints [1] used to create the surfaces. Each surface created defines the carving depth.

In order to do this association, three vectors are defined:

Virtual camera vector is a vector normal to the sphere containing the bounding box and pointing toward the center of the bounding box as can be seen in figure 4.1a. This vector is called $\overleftarrow{\mathbf{v}}$.

Camera vector Vector formed by the origin of the camera and the camera center point. This vector is normal to the sensor plane as can be seen in figure 4.1c. This vector is called $\overleftarrow{\mathbf{c}}$.

Bounding box vector Vector from the origin of the camera to the center of the bounding box. This vector is used to make sure the camera is in front of the bounding box as can be seen in figure 4.1d. This vector is called $\overleftarrow{\mathbf{b}}$

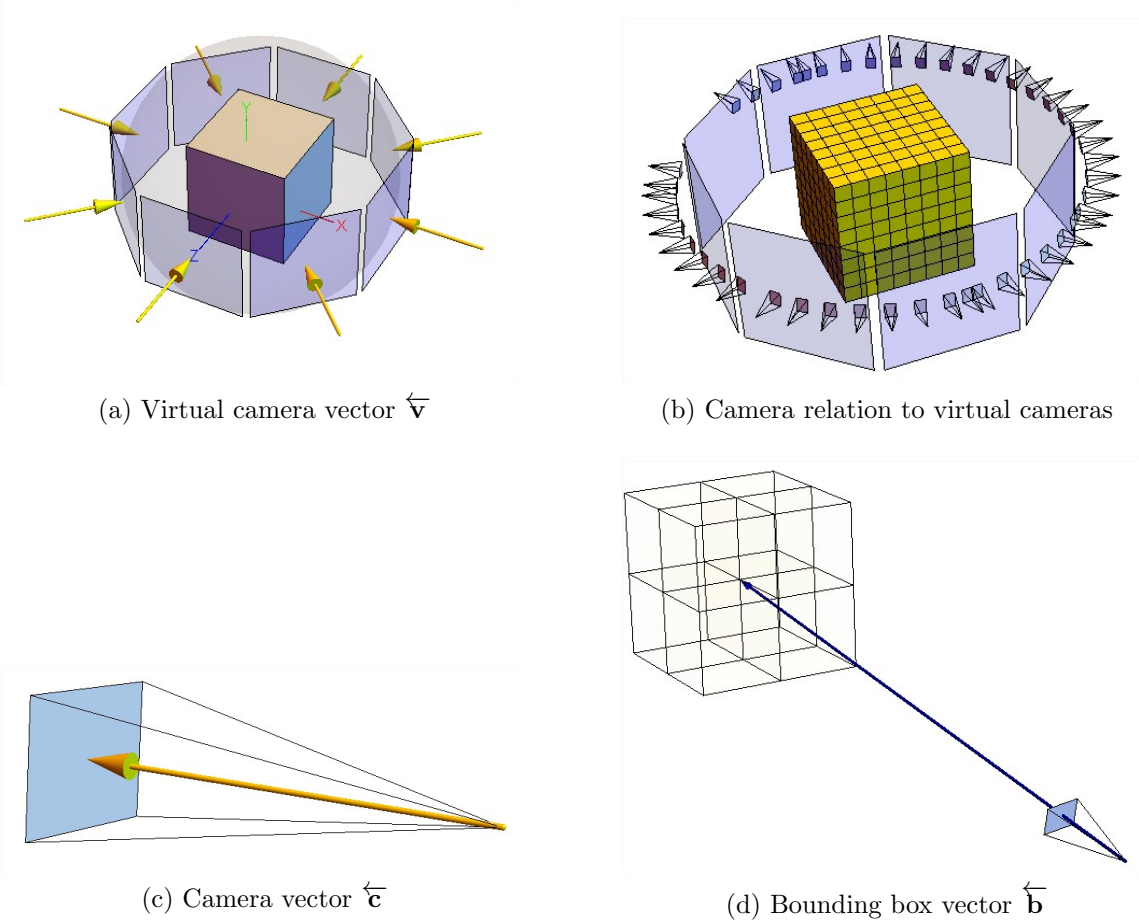
---

[1] Viewpoints are also called virtual cameras, both terms are used and specifies the same thing.

These vectors are used to assign cameras to viewpoints and they are normalised (vector length is 1).

$$\frac{\overleftarrow{\mathbf{v}}}{\|\overleftarrow{\mathbf{v}}\|} \cdot \frac{\overleftarrow{\mathbf{c}}}{\|\overleftarrow{\mathbf{c}}\|} = \cos\eta \tag{4.1}$$

Equation 4.1 calculates the angle η between the vectors $\overleftarrow{\mathbf{v}}$ and $\overleftarrow{\mathbf{c}}$. This angle will provide information on how parallel the camera plan is to the virtual camera plan. This value is also used as a quality factor in the cost function discussed in chapter 5.



(a) Virtual camera vector $\overleftarrow{\mathbf{v}}$

(b) Camera relation to virtual cameras

(c) Camera vector $\overleftarrow{\mathbf{c}}$

(d) Bounding box vector $\overleftarrow{\mathbf{b}}$

Figure 4.1: Virtual cameras

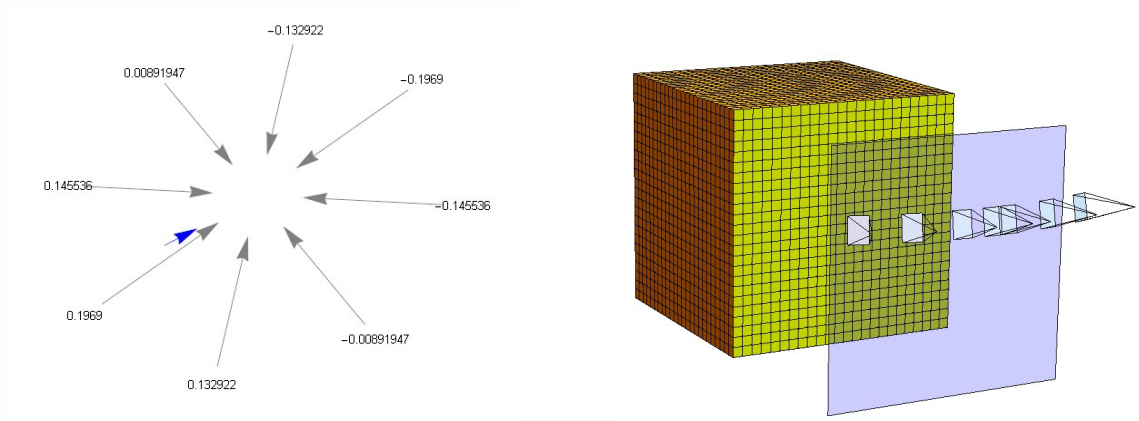
4.1 Camera selection

An image is associated to one or more viewpoints using the value obtained by the projection of the virtual camera vector $\overleftarrow{\mathbf{v}}$ over the camera vector $\overleftarrow{\mathbf{c}}$. The value must be positive to be valid because both vectors must be pointing in the same direction.

Also the projection of the camera vector $\overleftarrow{\mathbf{c}}$ over the bounding box vector $\overleftarrow{\mathbf{b}}$ must be positive otherwise the camera is on the other side of the bounding box, looking away.

$$\overleftarrow{\mathbf{c}} \cdot \overleftarrow{\mathbf{b}} > 0$$

When processing a viewpoint, the camera(s) assigned to it are used. But, it is also possible to use camera(s) assigned to another viewpoint when the projection value is positive. For instance, in figure 4.2a the camera represented with a blue arrow is associated with the normal vector having a projection value of 0.1969, which is the maximum value of all the normal. But other viewpoints can benefit from this camera's information like the viewpoints at projection values 0.132922 and 0.145536.



(a) Normal Vectors

(b) Cameras associated to a viewpoint

Figure 4.2: Cameras association to viewpoints

In the case when a camera only has part or none of the bounding box in its visual

field, the projection of the world position of a voxel onto the camera will fall outside the screen and the camera's information will not be used.

Because the chance of occlusions increases with the angle, the cost function must take this information into account when calculating the cost value.

Equation 4.2 is the list of cameras for a specific viewpoint. The list includes the camera matrix P as well as the angle η between the camera and the virtual camera.

$$\psi_\theta = \{\{\mathbf{P_{f,c,r,t}}, \eta\}_1, \{\mathbf{P_{f,c,r,t}}, \eta\}_2, ..., \{\mathbf{P_{f,c,r,t}}, \eta\}_p, \} \tag{4.2}$$

This cameras set will be used in the cost computation presented in the next chapter.

Chapter 5

MATCHING COST

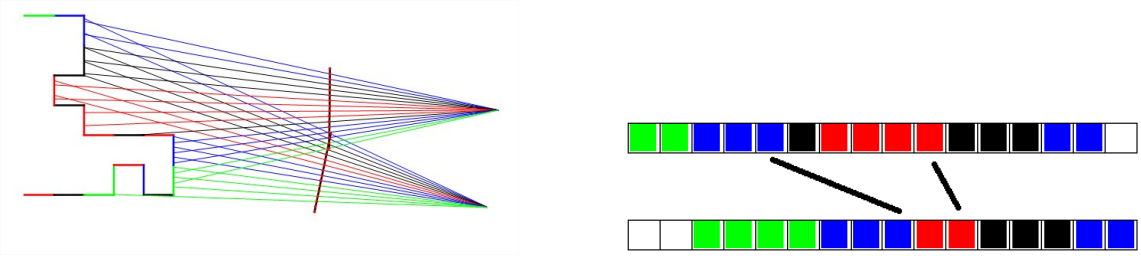
The cost is a value that provides information about the likelihood of associating a specific depth to a voxel from a particular viewpoint. The cost can also be seen as distance function and the algorithm will have to pay this cost when choosing a depth for a voxel position on a viewpoint. This cost, or distance, is minimal for voxel with high probability of being on the surface. There are multiple ways the cost can be calculated but the basic idea is to obtain a value that can be compared with other cost values in order to find the minimum cost. This minimum cost will be the depth where the surface of the 3D object has good chances to be located. But many factors could disturb the purity of the cost value. The surface property (Specular vs Diffuse), occlusions (a first camera does not see everything a second camera sees), lighting condition (shadows), camera calibration error (imprecise **P** matrix), lens aberrations, and mathematical rounding errors are all factors that influence the quality of the calculated cost. All the images available for a viewpoint could contribute to calculating the cost, as in plane sweep method [14]. The quality factor (the angle of the camera) is used to adapt the cost contribution of the camera.

5.1 1D camera and disparity

In order to understand the relation between cost and depth, the present section will illustrate the cost calculation for a 2-dimensional world captured by a 1-dimensional camera sensor (as opposed to the normal camera usage on a 3 dimensional world with 2 dimensional sensors).

5.1.1 1D image capture

Figure 5.1a shows two 1-dimensional cameras used to simultaneously[1] capture an object. The two cameras are located side by side and are pointing toward the middle of the object. The pixels seen by the camera's sensors are shown in figure 5.1b.



(a) Two 1-D cameras looking at an object (b) Pixel matches

Figure 5.1: 1-D Camera costs and depth

5.1.2 1D cost calculation

The cost is calculated for all possible depths and this is done for each pixel of the reference sensor (the bottom one in figure 5.1b). For example, if we take the rightmost red pixel from the bottom sensor and try to match this pixel with the other sensor (the one shown on top) we will get a cost value for each possible match. The first best match will be with the top rightmost red pixel with a distance of 1 pixel to the left as shown on figure 5.1b. This distance of 1 pixel is called the disparity. We can do the same thing with the bottom blue pixel and we will get a disparity of 4 pixels. The red pixel is further away (from the cameras) than the blue pixel. Since the depth is inversely proportional to the disparity it works with the disparity value that we retrieved.

[1] the simultaneous capture is only important if the scene is not static

5.2 Simple cost value

In the previous example, the cost was calculated for 2 RGB pixels. The following equations calculates the cost for any number of pixels where a pixel is composed of 3 channels (RGB). The ensemble ϕ contains all the pixels used to calculate the cost as seen in equation 5.1.

$$\mathbf{p} = (R, G, B)$$

$$\phi = \{\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_n\}$$

$$\mu = \frac{\sum_{k=1}^{n} \mathbf{p}_k}{n}$$

$$cost(\phi) = \sum_{k=1}^{n} \sum_{i=RGB} \|\mathbf{p}_k - \mu\|_i \tag{5.1}$$

The cost value would be minimal when comparing pixels of the same colour. This simple method of cost calculation is described in [15] and it has shown to give good results.

5.3 Bilinear interpolation

When using the A and P matrix to convert a voxel position into a world position we do not get an integer value. We use a bi-linear interpolation to get the correct RGB value. The figure 5.2 demonstrates the red pixel value being bi-linearly interpolated from the surrounding pixels. A first interpolation is made along a first axis followed by another interpolation to get the value at the red pixel position. A bilinear function receives an image and pixel position and returns the interpolated pixel.

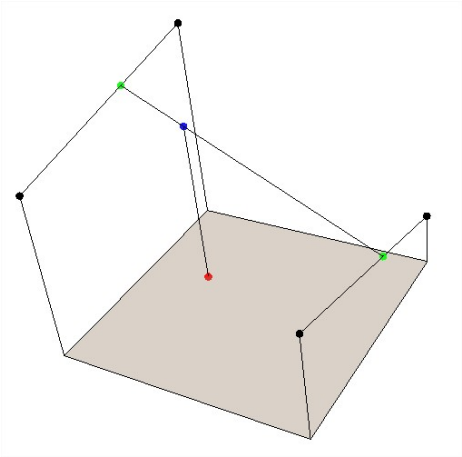


Figure 5.2: Red pixel value bi-linearly interpolated from 4 surrounding pixels.

5.4 Cost calculation on a viewpoint

The cost calculation for a voxel on a specific viewpoint needs to have the parameters of the viewpoint $(level, \theta, s)$, as seen with matrix A in equation 3.12 on page 20, and the voxel position. The cost function is like this: $cost_{level,\theta,s}(x, y, d)$ and it depends on the set of camera ψ_θ as seen in equation 4.2 on page 29. The list ϕ is the list of bilinearly interpolated pixel values calculated on the images I_k with the camera matrices $\mathbf{P}_k$ and the viewpoint coordinate to world coordinate conversion matrices $\mathbf{A}_{level,\theta,s}$.

$$\psi_\theta = \{\{\mathbf{P}_1, \eta_1\}, \{\mathbf{P}_2, \eta_2\}, ..., \{\mathbf{P}_p, \eta_p\} \tag{5.2a}$$

$$\phi = \left\{ Bilinear(I_k, \mathbf{P}_k \ \cdot \ \mathbf{A}_{level,\theta,s} \ \cdot \ (x, y, d, 1)^\top) \right\} \ \forall \ \mathbf{P}_k \ \in \ \psi_\theta \tag{5.2b}$$

$$cost(x, y, d) = cost(\phi) \tag{5.2c}$$

The steps to retrieve a pixel value are as follows:

- From the voxel position (x, y, d), the world position is calculated with Matrix

A (equation 3.12).

- From the world position, the camera sensor position is calculated with Matrix P (equation 2.1).
- From the sensor position, the pixel value is calculated with bi-linear interpolation.
- For all cameras associated with the viewpoint, the ensemble ϕ is constructed.
- The cost is calculated taking the angle η into account to reduce the importance of the cameras not facing the viewpoint.

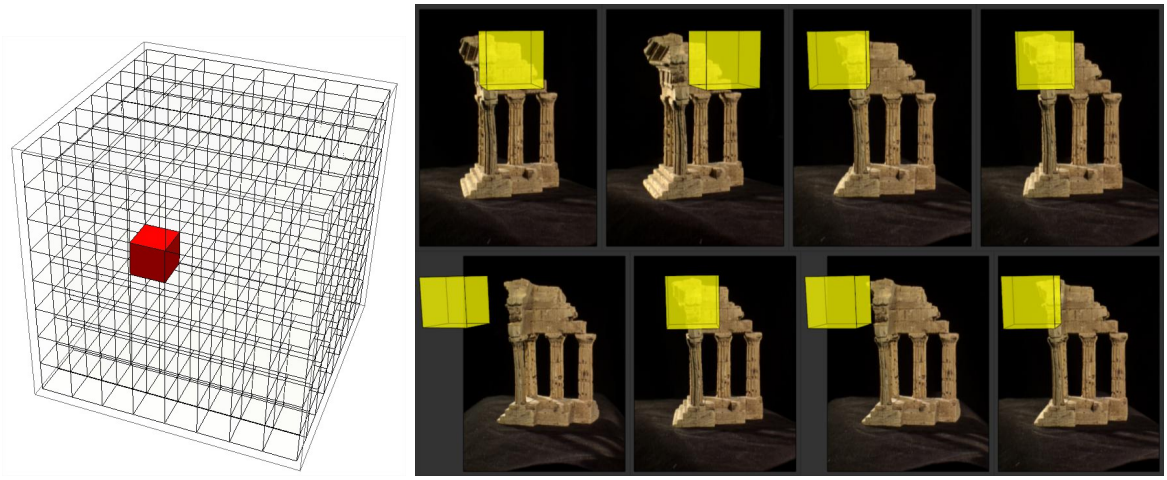
5.5 Cost for big voxels

It may be necessary to calculate the cost for a voxel that is not at the maximum level of the octree. This could happen if the image resolution is much bigger than the required 3d model voxel resolution. In this case, each voxel will span multiple pixels.

5.5.1 Patch of pixels

The voxel corners are projected to the images associated with the viewpoint. These projections define a region on the images that could span multiple pixels. This region is called a patch of pixels. This patch can be relatively big, for instance Figure 5.3a shows a voxel projected onto the images and we can see that this voxel is not a good match because it covers a heterogeneous area of the images.

To determine the cost of a voxel projection, we must measure the dissimilarity between the patches. One way of measuring this is to aggregate the information of each patch into a histogram [16]. Afterwards, the dissimilarity of each patch is calculated by measuring the correlation between the histograms as seen in equation 5.3. It is not efficient when we are comparing more than two histograms.



(a) Octree voxel representation (b) Voxel projections

Figure 5.3: Voxel projection

$$d_{correl}(H_1, H_2) = \frac{\sum_i H_1'(i) \cdot H_2'(i)}{\sqrt{\sum_i H_1'(i) \cdot H_2'(i)}} \tag{5.3}$$

$$H_k'(i) = H_k(i) - \frac{1}{N} \sum_j H_k(j) \tag{5.4}$$

5.5.2 Cost relation between levels

Another way to compare patches of pixels is by calculating the cost at the highest level n of the octree (the smallest voxel) and then performing the percolation. There is no gain in doing this when working at a lower level except to have a simple solution.

5.5.3 Scaling image to fit level

A third way of calculating the cost at a lower level is to use equation 3.17 on page 23 where a maximum level is found with a pixel distance equal to 1. We could use the information from the pixel distance along the level we want to use and scale down the image to get a distance of 1. For example, if we wanted to work at level 7, we

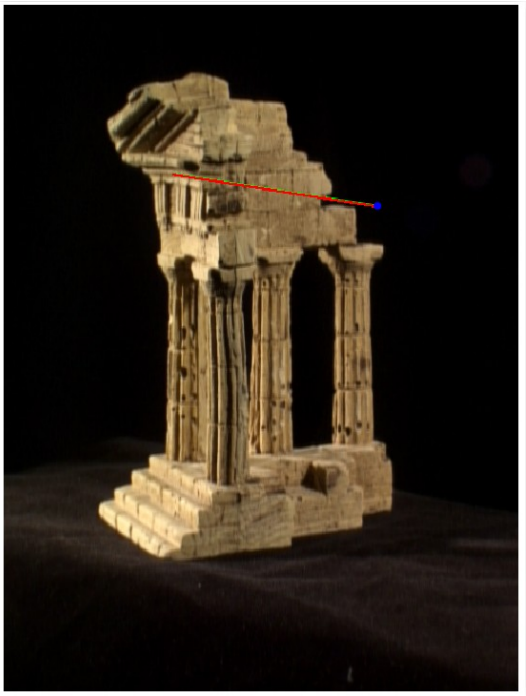
could find a scaling factor by looking at table 3.1 where we see that the pixel has a distance of 9.36 at that level. Scaling down the image from 640x480 to 68x51 will change the distance in the image to 1 pixel. The camera matrix would need to be adapted in that case.

5.6 Cost line on image

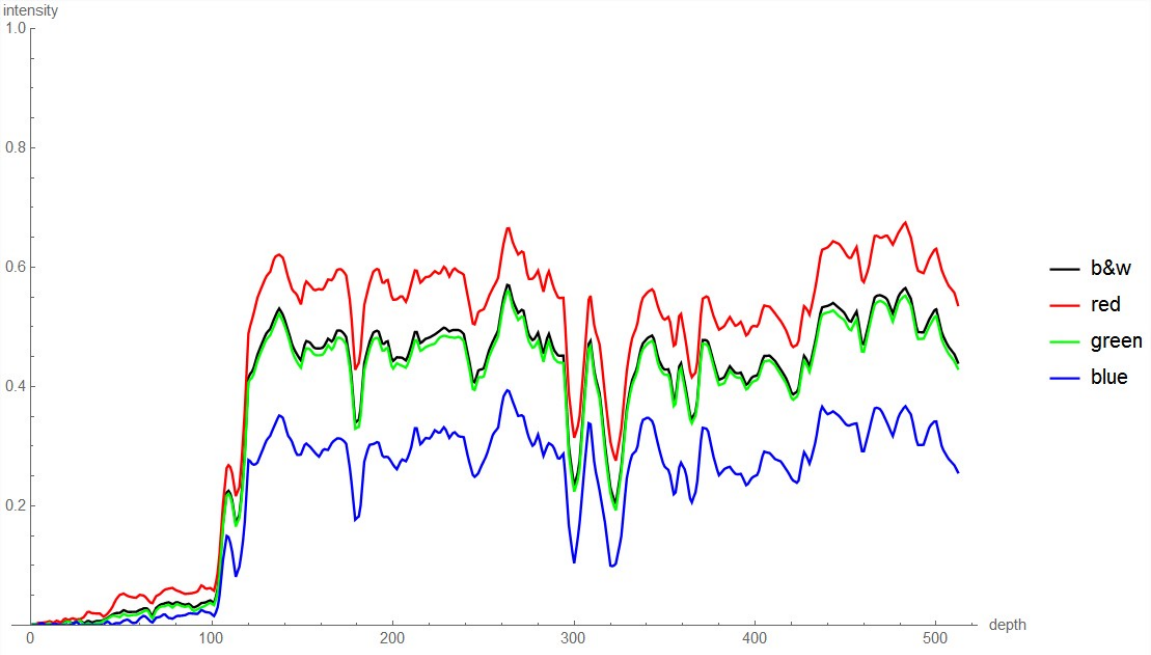
By taking voxel positions along the z axis (depth) of a virtual camera and by projecting these positions to an image associated with the virtual camera it will draw a series of dots on the image. If the voxel resolution is high, this series of dots will look like a continuous line on the image. This line is called a cost line. The figure 5.4 shows some images associated to viewpoint 0 with the cost line for a same voxel position. The x and y position of the voxel are the same and the green line represents the depth range. The red dots are depth position at level 3 (8 dots). Figure 5.5a zooms to image 9 of the set and we can see the depth 0 at the blue dot. Figure 5.5 on page 38 shows the red, green, blue, and black and white[2] intensity of the cost line of image 9. Figure 5.6 on page 39 shows the black and white intensity with color mapping to the depth line. The color is also filling the graph. This color mapping will be used in the following figures.

[2] Conversion to black and white is done like this: $bw = 0.21 * R + 0.72 * G + 0.07 * B$

Figure 5.4: Projection of the same cost line on multiple images of the viewpoint.



(a) Image 9 of previous image set (figure 5.4). The depth 0 starts at the blue point and increase toward the left.



(b) black and white, red, green, and blue intensity along depth line.

Figure 5.5: Cost along depth line for image 9

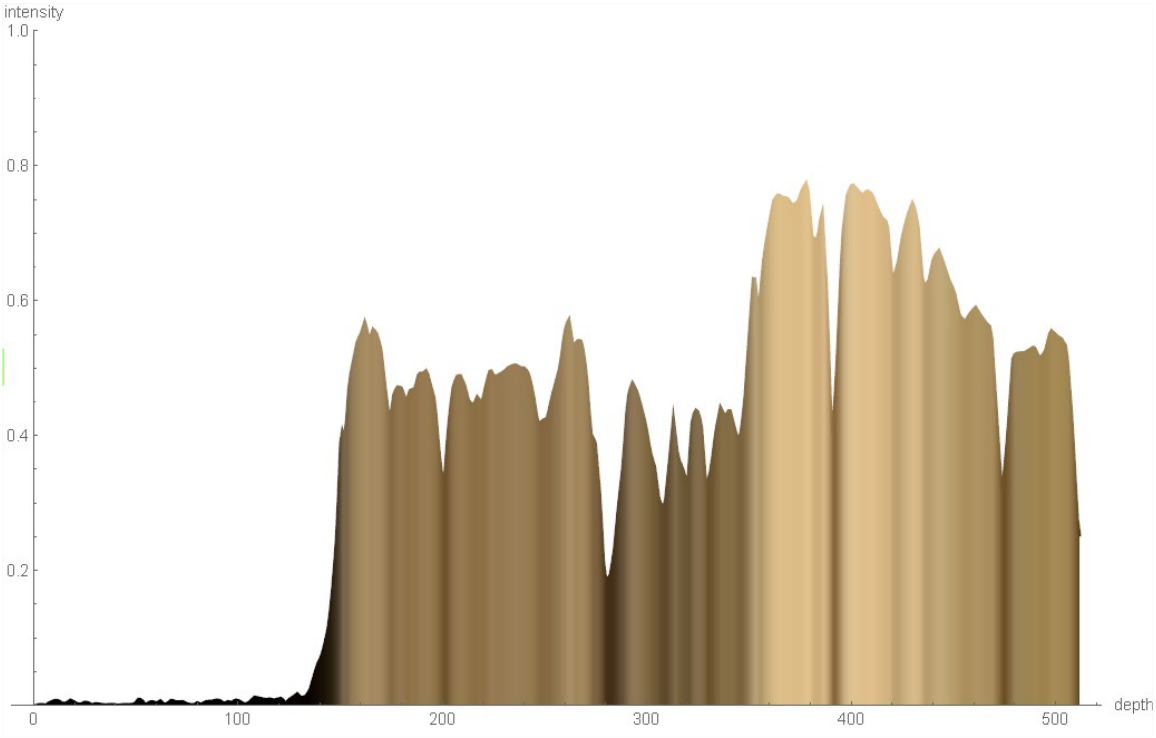
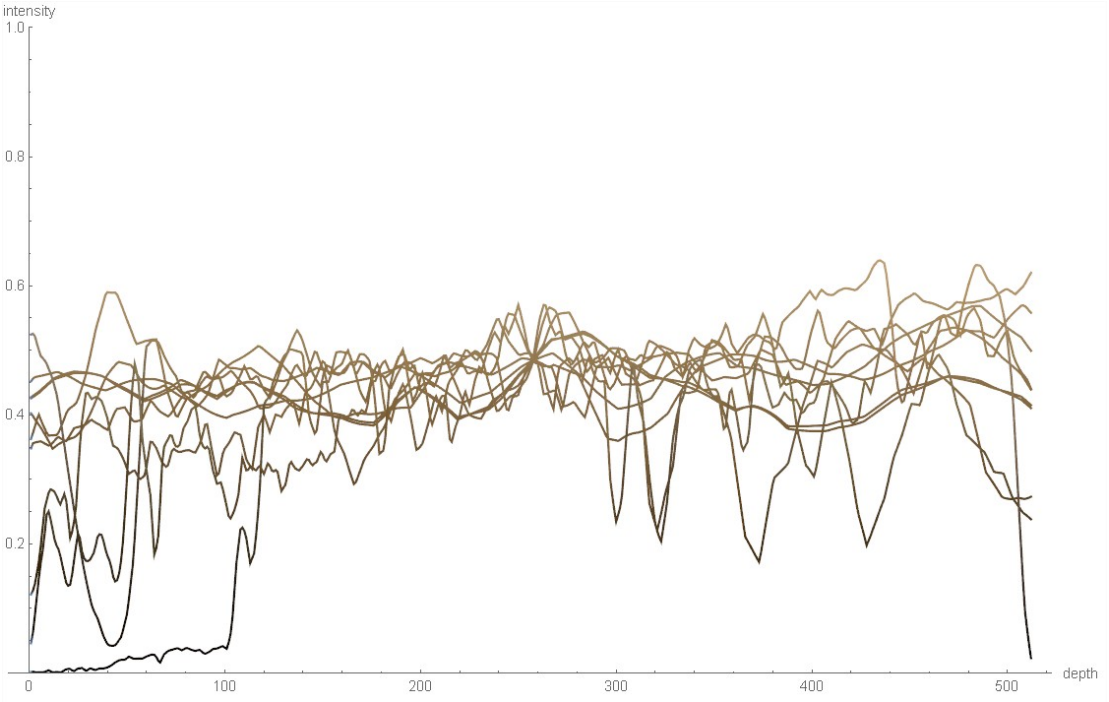
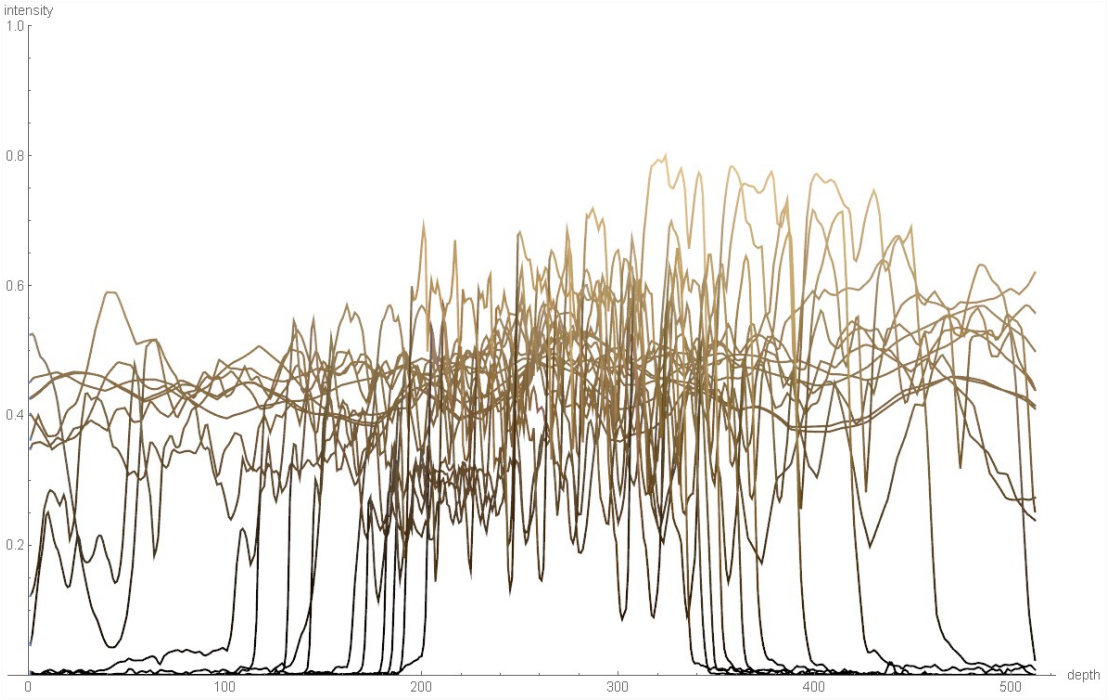


Figure 5.6: Cost line on image with color mapping.

Figure 5.7a shows the first 10 cameras of the sets drawn on the same graphics. It is interesting to note that there seems to be a convergence around depth position 260. Figure 5.7b also shows the intensity but for all cameras of the sets. In that case, no convergence seems to happen and this is due to the noise induced by the occlusions.



(a) Costs of the first 10 cameras plotted together.



(b) Cost of all the cameras associated to viewpoint 0.

Figure 5.7: Cost for multiple cameras

Figure 5.8a demonstrates more clearly the convergence around position 260. Figure 5.8b shows the min max graph for all cameras.

Figure 5.9a and figure 5.9b clearly show the range of intensity.

Figure 5.10a shows the mean of the cost line in red as well as the normalized variance in blue for the first 10 cameras. Figure 5.10b shows the same thing but for all cameras of the viewpoint's set.

5.7 Quality factor in cost

Looking at image 5.10a we could find a minimum value at the depth position 258. But it is impossible to do that on figure 5.10b. The solution is to change the cost function and to use the quality factor seen in section 4.1. The idea is to reduce the costs using the angle between the camera vector and the virtual camera vector. The weight of the camera is further amplified (or reduced) by modifying the quality factor with a constant exponent.

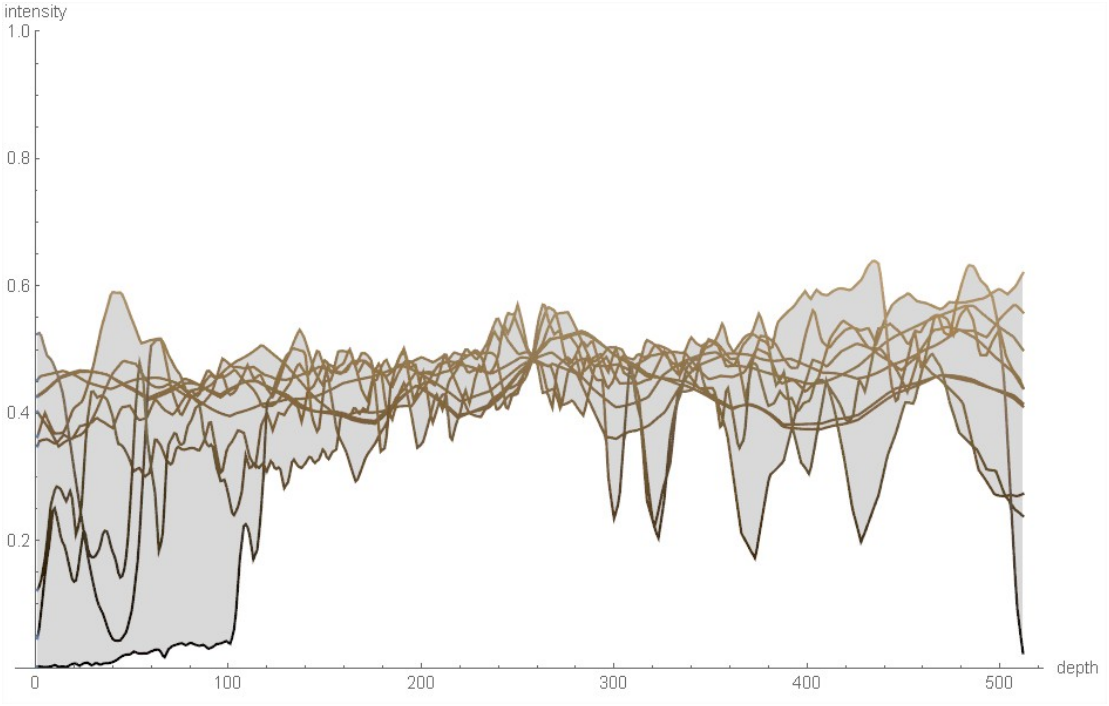
$$cost = \sum_{cam=2}^{nb} Variance(1 \ldots cam) * QualityFactor_{cam}^{3} \tag{5.5}$$

The quality driven cost function from equation 5.5 uses all the cameras of the viewpoint and calculates the variance for subsets of these cameras. These subsets are modified by the quality factor (the camera angle) and summed together. The quality factor is modified with an exponent of 3. A value of 3 is chosen because it makes the quality factor non-linear and it will increase the cost when the angle is high. This new cost evaluation provides better results when using all the cameras as can be seen in figures 5.11, 5.12, and 5.13.

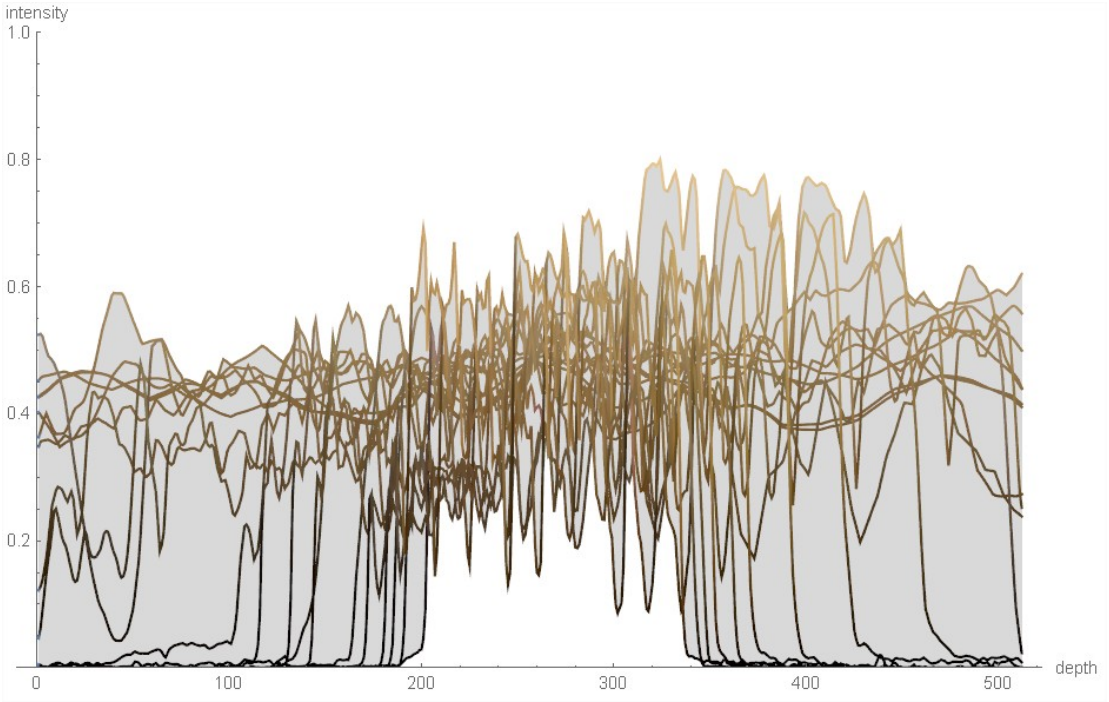
The depth retrieved by looking at the minimum value is relatively stable as long as the number of cameras is greater than 2 as can be seen in table 5.1.

Nb Cameras	depth
2	45
3	257
4..12	258
13	259
14..20	258

Table 5.1: Depth found per number of cameras in subset



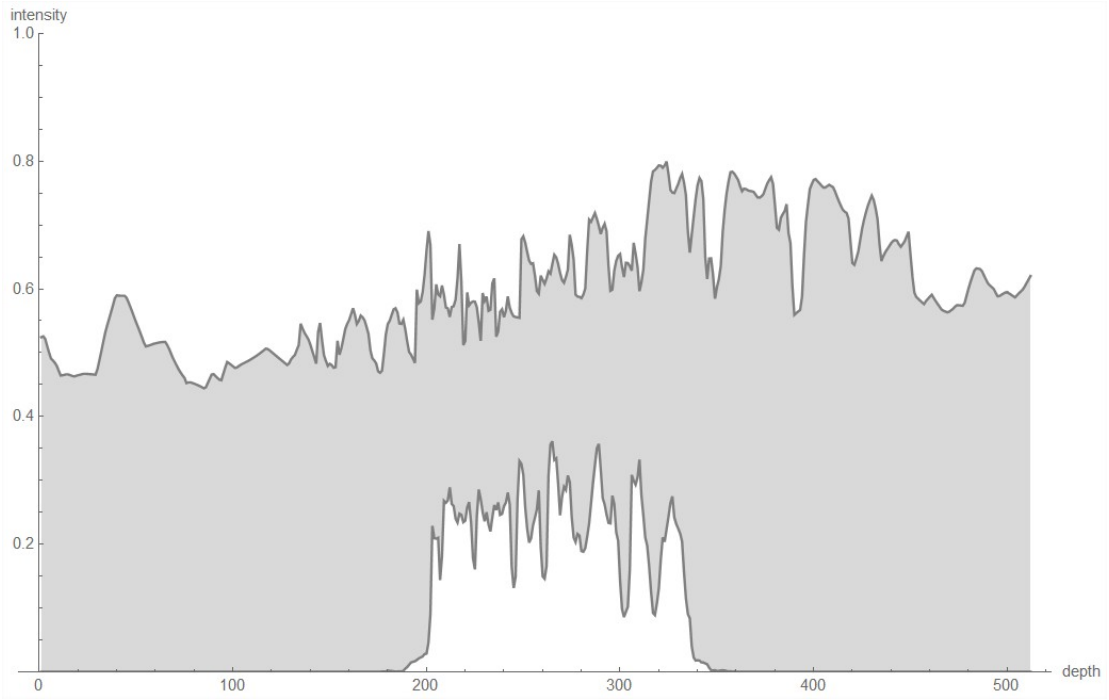
(a) Min max range of costs for the first 10 cameras.



(b) Min max range of costs for the first 10 cameras.

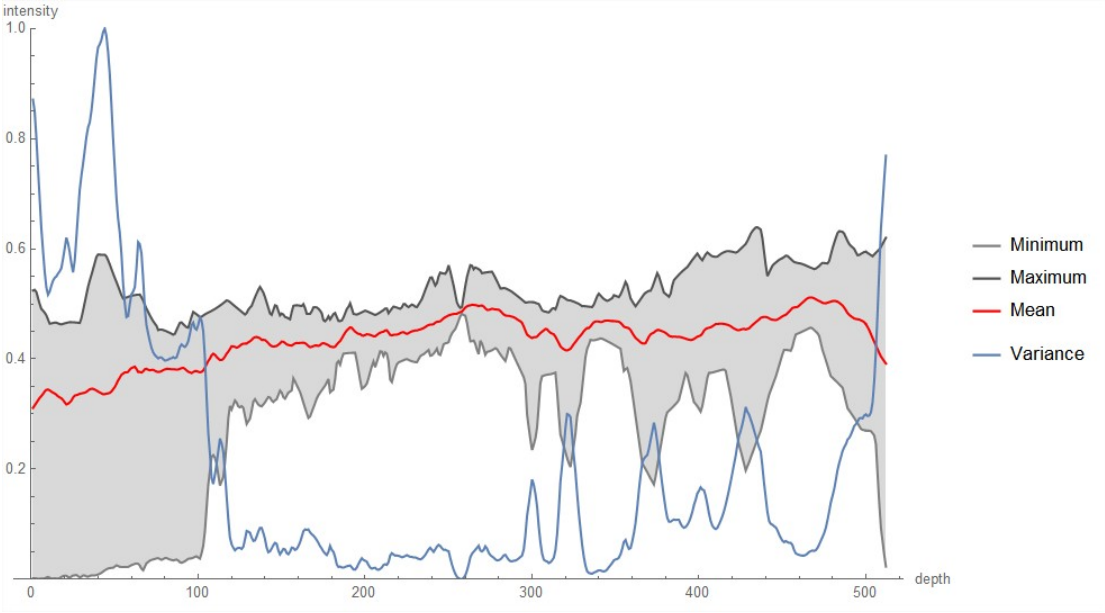
Figure 5.8: Minimum and maximum costs along depth line

(a) Min max range of costs for the first 10 cameras.

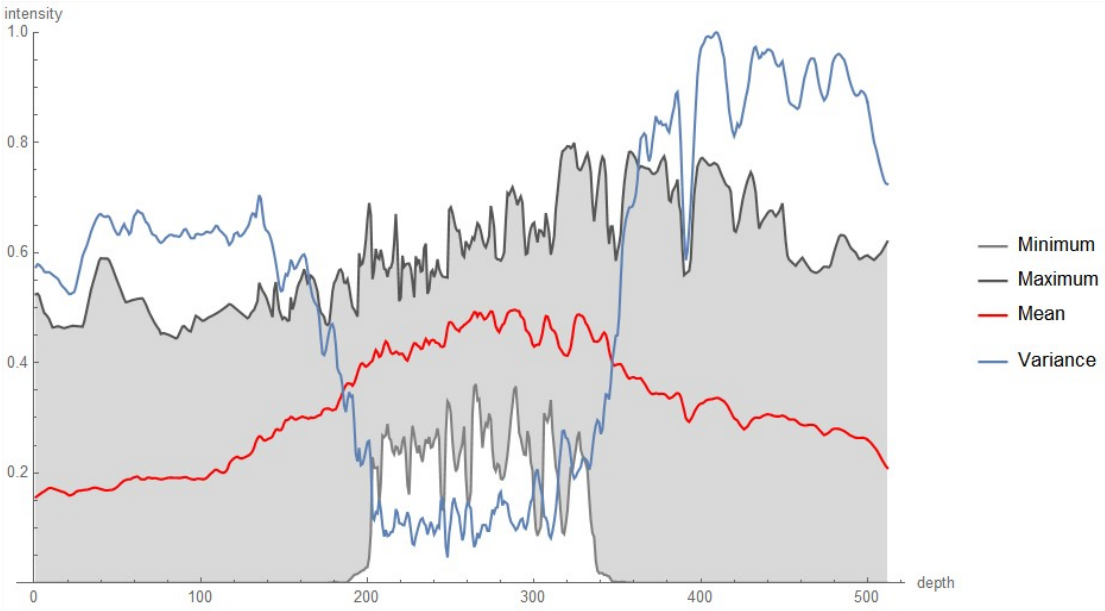


(b) Min max range of costs for the first 10 cameras.

Figure 5.9: Range of costs along depth line



(a) Variance for 10 cameras.



(b) Variance for all cameras.

Figure 5.10: Cost along depth line (without quality factor)

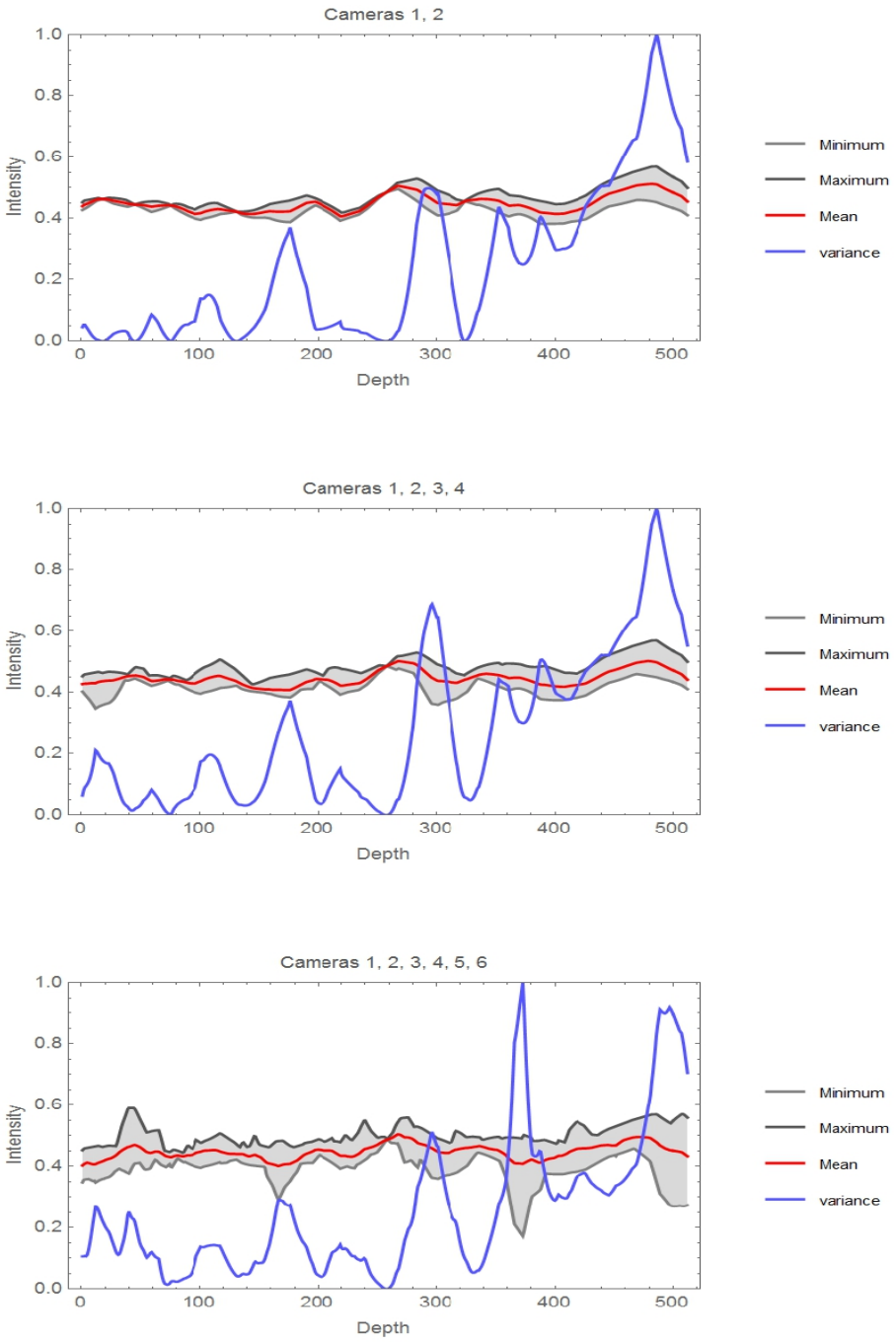


Figure 5.11: Quality pondered cost function (2 to 6 cameras). The minimum value is located at the correct depth(258).

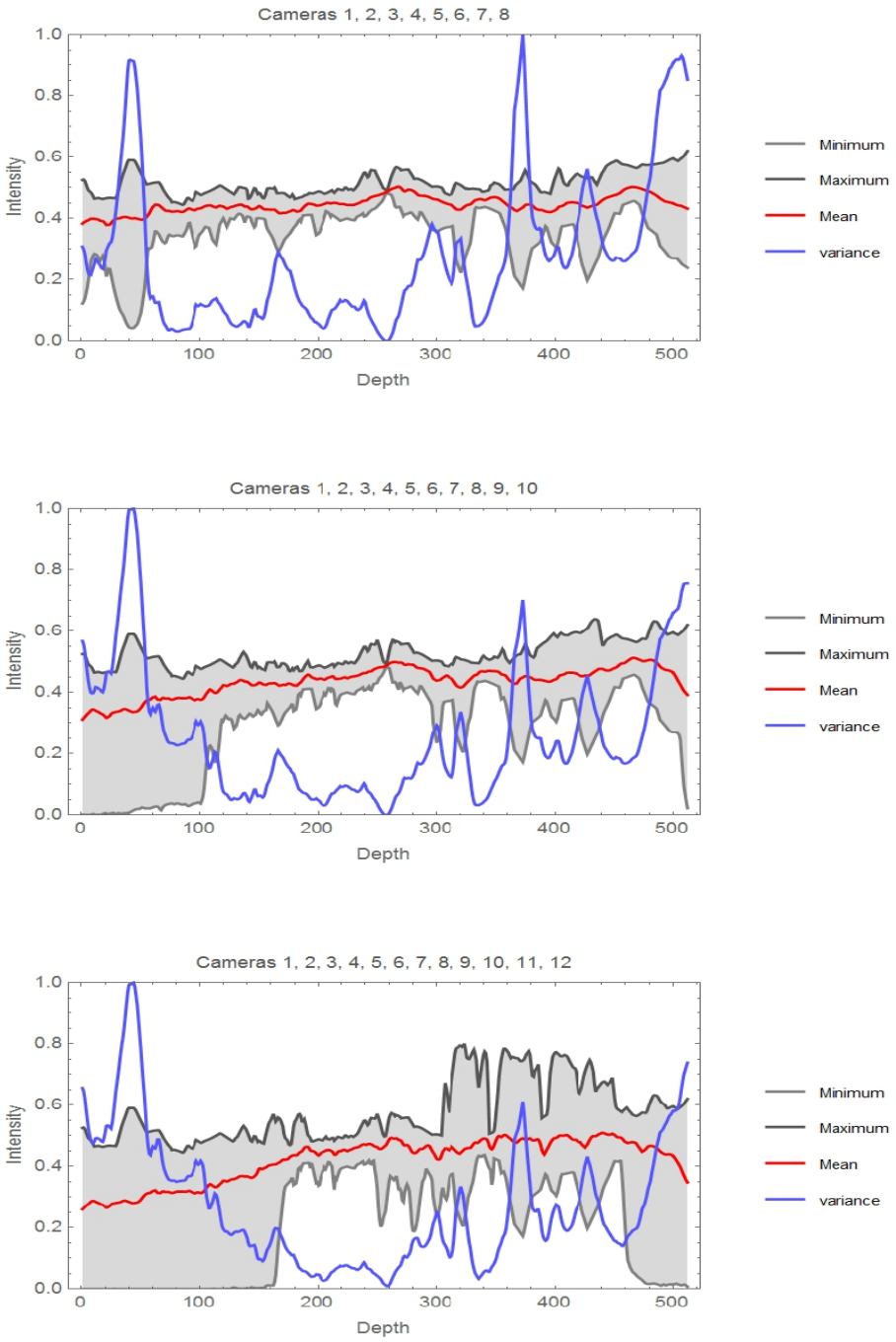


Figure 5.12: Quality pondered cost function (8 to 12 cameras). The minimum value is located at the correct depth(258).

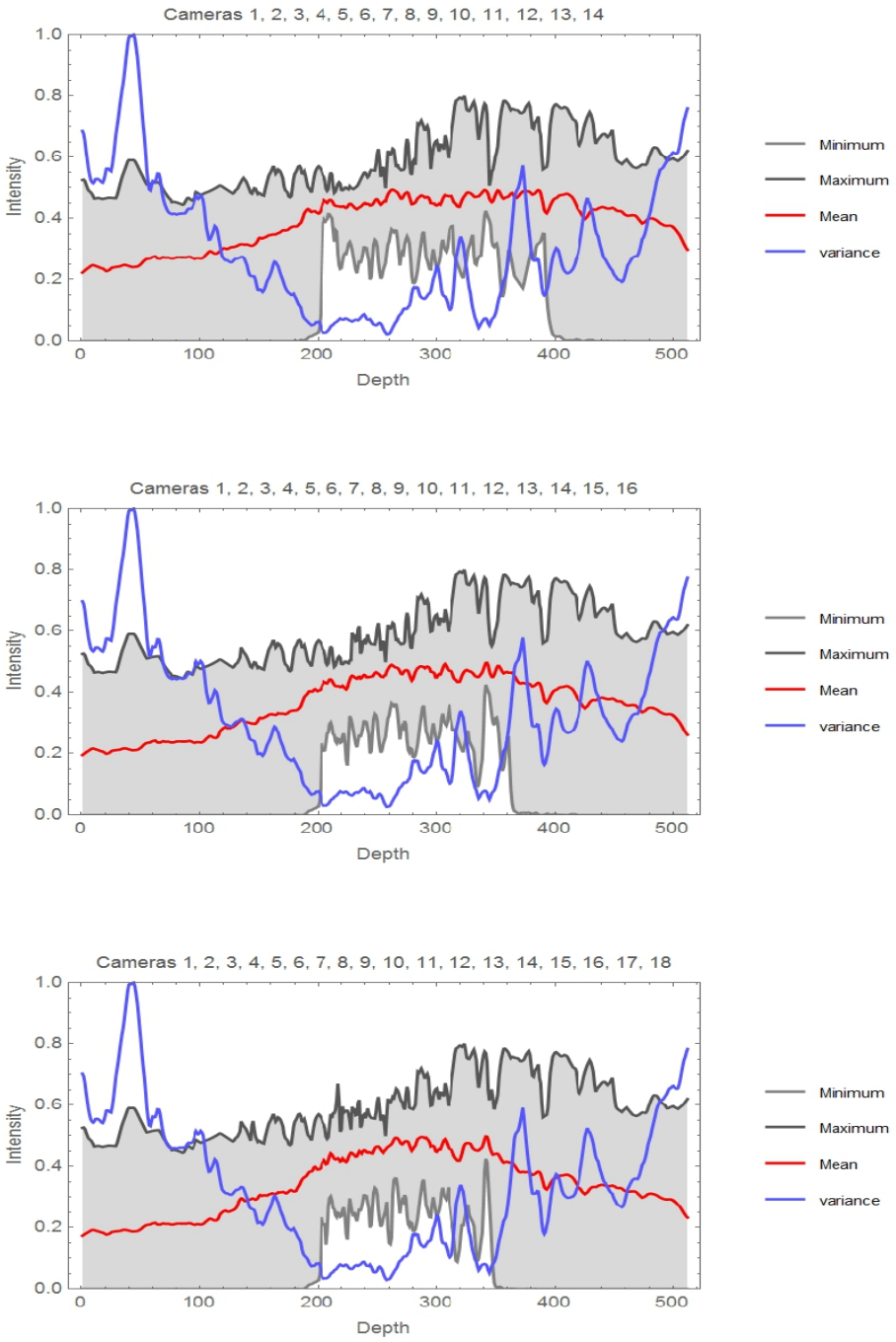


Figure 5.13: Quality pondered cost function (14 to 18 cameras). The minimum value is located at the correct depth(258).

Figures 5.14 and 5.15 demonstrate the model at level 8 retrieved only using the cost function and without any smoothing. We can see some noise in the depth which impacts the quality of the reconstructed model. Nevertheless, the model is still looking good.

Figure 5.14: Carving with pondered cost (no smoothing).

Figure 5.15: Carving with pondered cost (no smoothing) another point of view.

Figure 5.16 shows what would be the 3D model if the preliminary visual hull step was not used. We can hardly recognize the object.

Figure 5.16: Carving with cost only without visual hull.

Chapter 6

SEMI-GLOBAL MATCHING

Once a cost function is available, we need to find good matches and eliminates bad matches. A wide variety of methods is available for this, from Winner-take-all [17] to slower global method like graph-cuts [18],[7]. A good candidate is the semi-global matching method [1] which is considered one of the best algorithm for stereo. In this chapter we will see the way the semi-global matching method is adapted to find good matches and help in the reconstruction of a virtual 3D model based on a virtual camera reference frame.

6.1 Semi Global Method

The algorithm from Hirschmuller [1] and its newer more efficient version [19] performs smoothing on a cost cube by adding a value to the cost when a disparities change occurs. This disparity smoothing helps the cost function when there is noise and/or occlusion. SGM encourages a surface to be continuous by adding a 0 value to the cost when the depth does not change, adding a small value P_1 when the depth changes by one pixel and, finally adding a larger value P_2 when the depth changes by more than 1. This happens when there is a depth discontinuity. Equation 6.1 shows how a SGM penalty is added to the cost. Since the viewpoint is not necessarily fronto-parallel to a surface, the penalty P_1 must be small. Penalty P_2 is added when there is a discontinuity so it must be bigger than P_1.

The semi-global matching cost function, which can be solved using dynamic programming is:

$$
\begin{aligned}
L_r(\mathbf{p}, d) \; &= C(\mathbf{p}, d) + \\
&\min(L_r(\mathbf{p} - \mathbf{r}, d), \\
&\quad L_r(\mathbf{p} - \mathbf{r}, d-1) + P_1, \\
&\quad L_r(\mathbf{p} - \mathbf{r}, d+1) + P_1 \\
&\quad \min_i L_r(\mathbf{p} - \mathbf{r}, i) + P_2) - \\
&\min_k(\mathbf{p} - \mathbf{r}, k))
\end{aligned}
\tag{6.1}
$$

SGM is applied to every surface on every viewpoint. A contribution is done by a modification to the SGM method where in the original algorithm P_2 is dynamically adapted to take the intensity of the image into account but when working with virtual cameras, P_2 cannot be adapted since no information of intensity is directly available on the viewpoint (it is a virtual camera) so P_2 has a fix value.

Another modification done to the SGM algorithm is when the previous pixel is empty for all disparity along the dynamic programming line. This can be seen in figure 5.14 on page 49 when a previous pixel is between the pillars and the current pixel is on the pillar border. In this case, the modified SGM will not try to add a penalty to the cost since none of the disparity of the previous pixel offers a surface.

SGM will perform a modification of the cost values that will result in a smoothing. This processing is performed along 8 or 16 lines, as shown on figure 6.1. The cost results from all these lines is added and the minimum values will form a surface used for carving the octree.

The SGM method works efficiently compared to method like Maximum flow which works globally [18] because SGM relies on dynamic programming which is efficient to compute. Dynamic programming is used in SGM when calculating the cost along a line; the previous values are kept and used for current cost values. Also, SGM is able

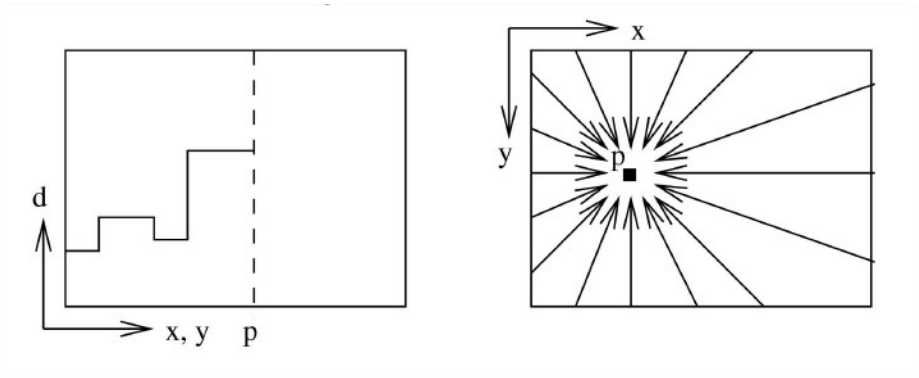


Figure 6.1: Aggregation of costs in disparity space. (Image from [1])

to work on sub-section also referred to as cells.

6.2 SGM by cells

SGM requires a lot of memory when working on a complete surface with a significant disparity range. It is also not trivial to prevent SGM from using CPU on areas that we know upfront are empty. The empty areas are generated by the convex hull step or by another viewpoint that has already carved the octree. One solution to this is to work on the surface by section. Figure 6.2 shows the subdivision done at the octree level 3 for viewpoint #2.

SGM will be performed on all cells that are not empty. SGM will calculate a smoothed cost from the base cost calculated by the method described in section 5.6 on page 36.

6.3 SGM results

To illustrate the usefulness of the convex hull, figure 6.3 shows 2 images where the convex hull step was not executed. We can see that when there is an ambiguous background (uniform color) the reconstruction using SGM has some problems figuring the correct depth. But on the textured area SGM does a pretty good job at reaching

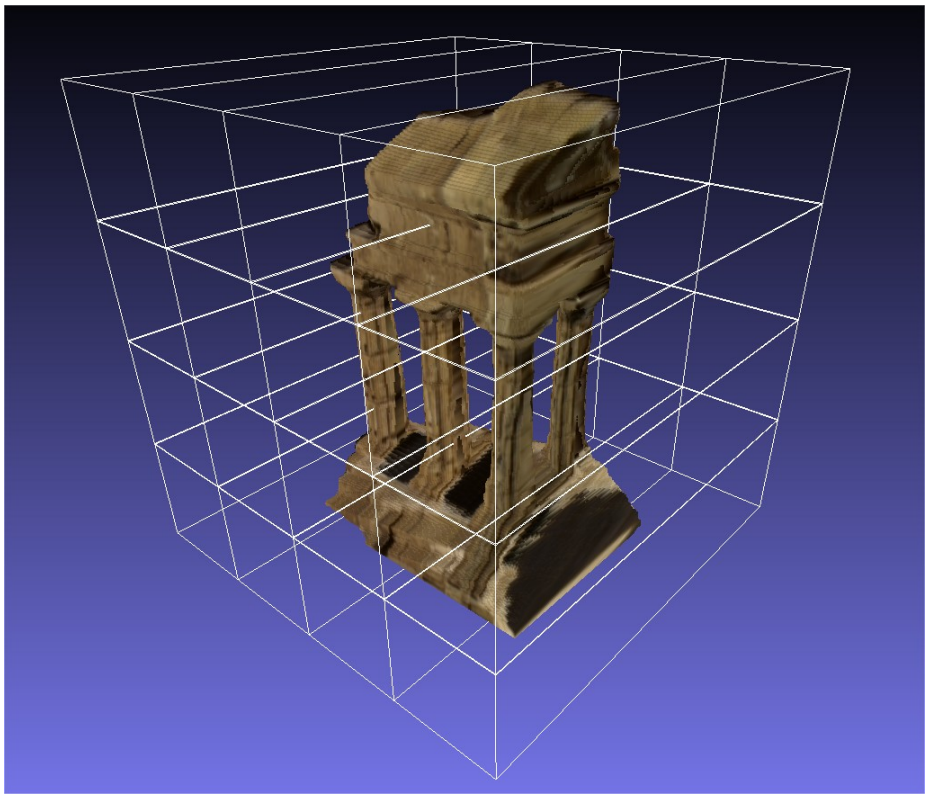
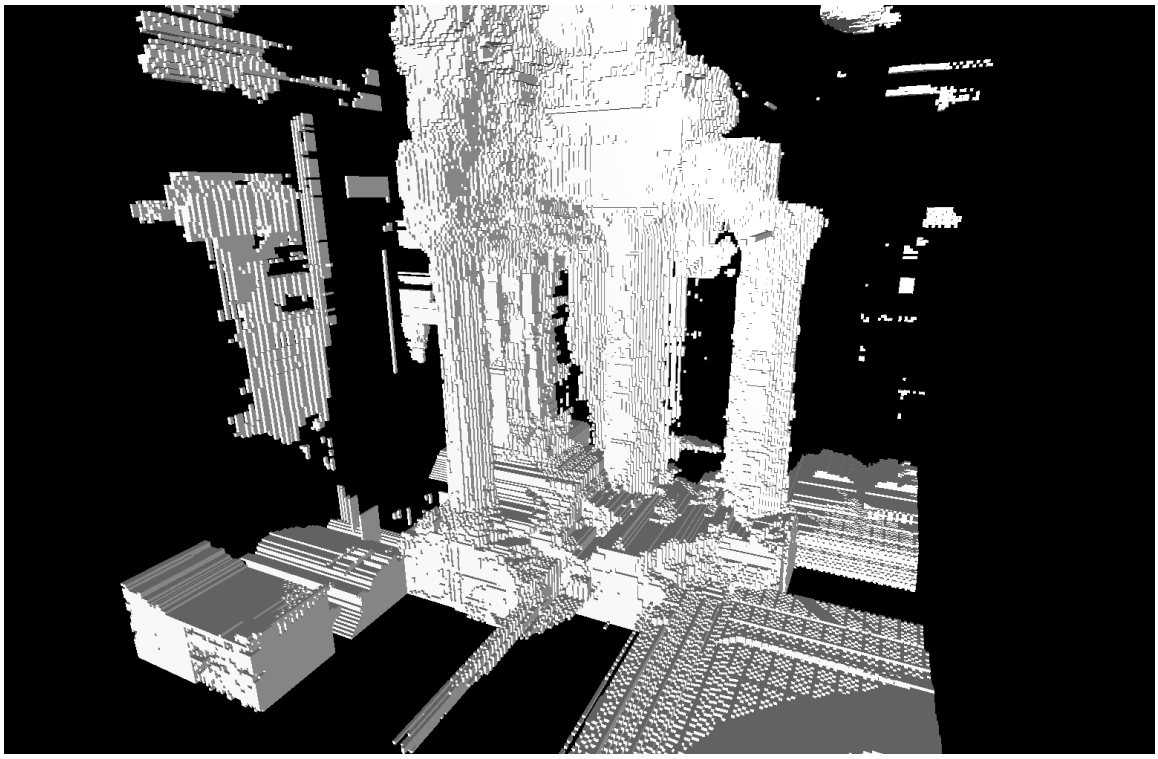


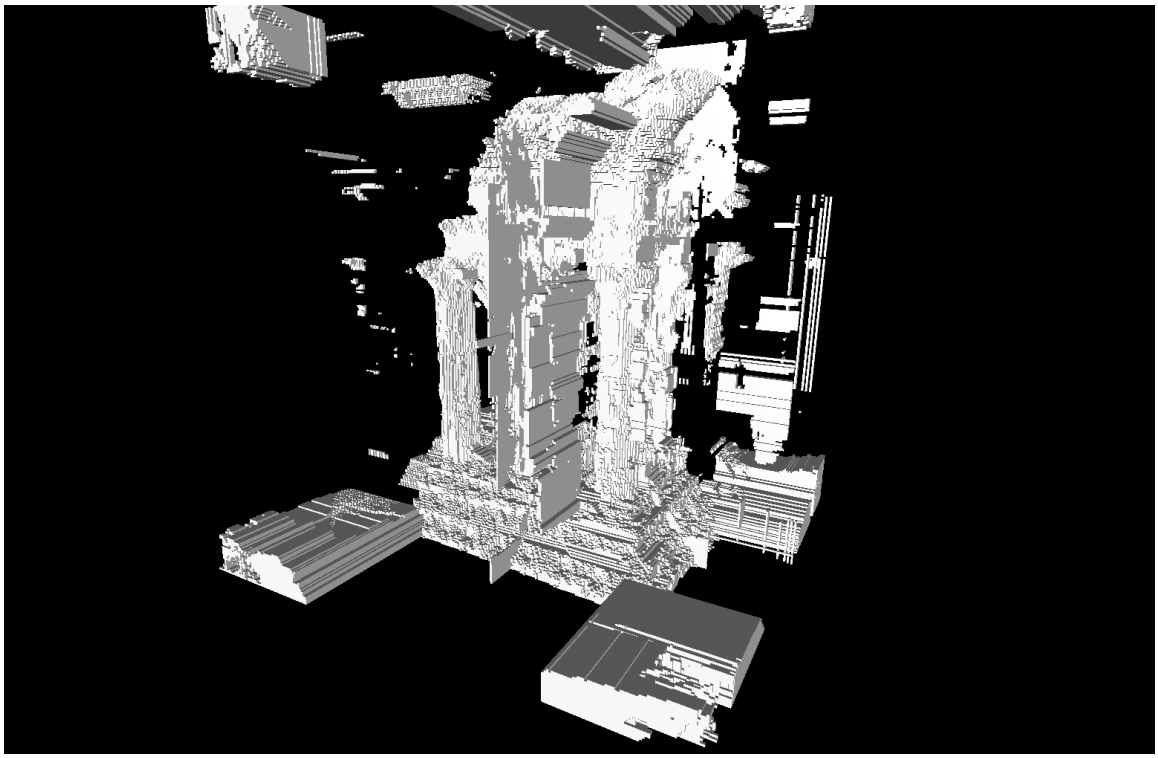
Figure 6.2: Division of model in multiple sections.

the surface.

Figures 6.4 and 6.5 illustrates the usefulness of SGM. If you look at the vertical walls you will see that the cost only reconstruction is noisier than the SGM reconstruction. The stairs are also more defined and less noisy.

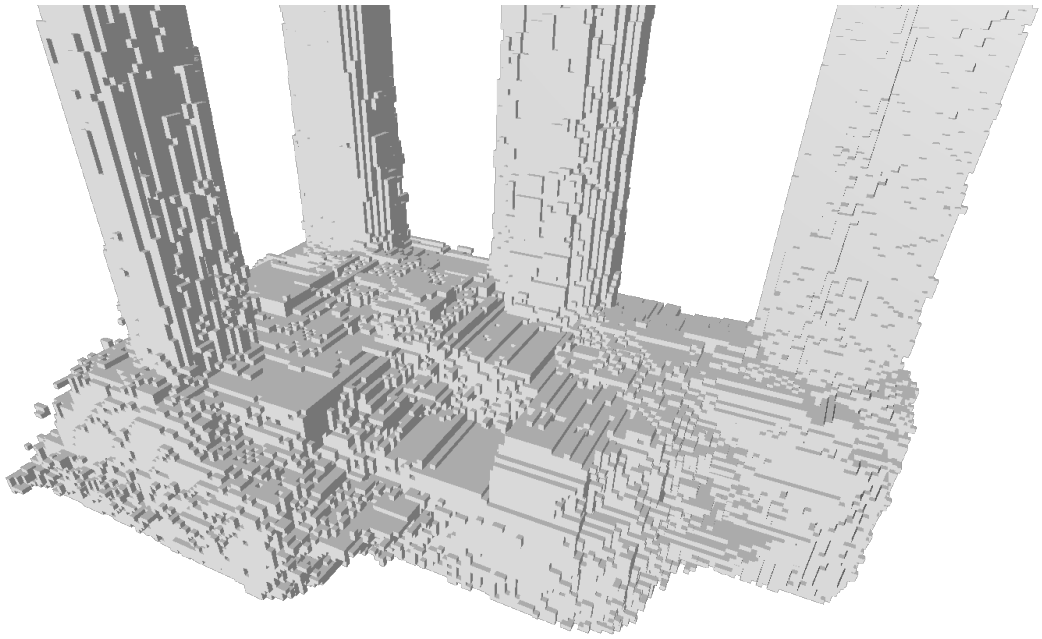


(a)

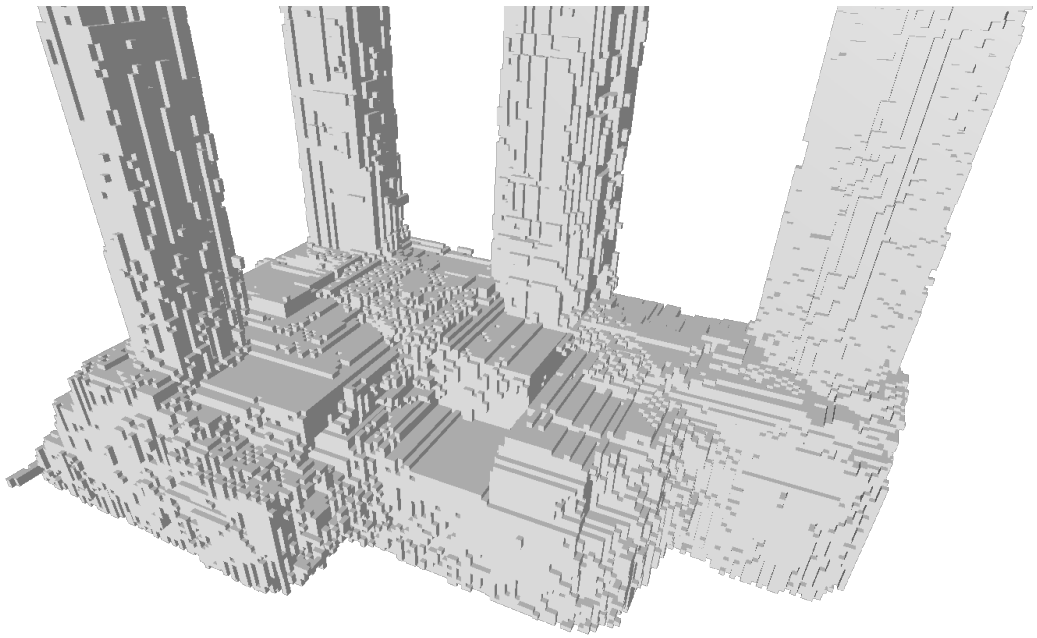


(b)

Figure 6.3: Carving with SGM and without visual hull.

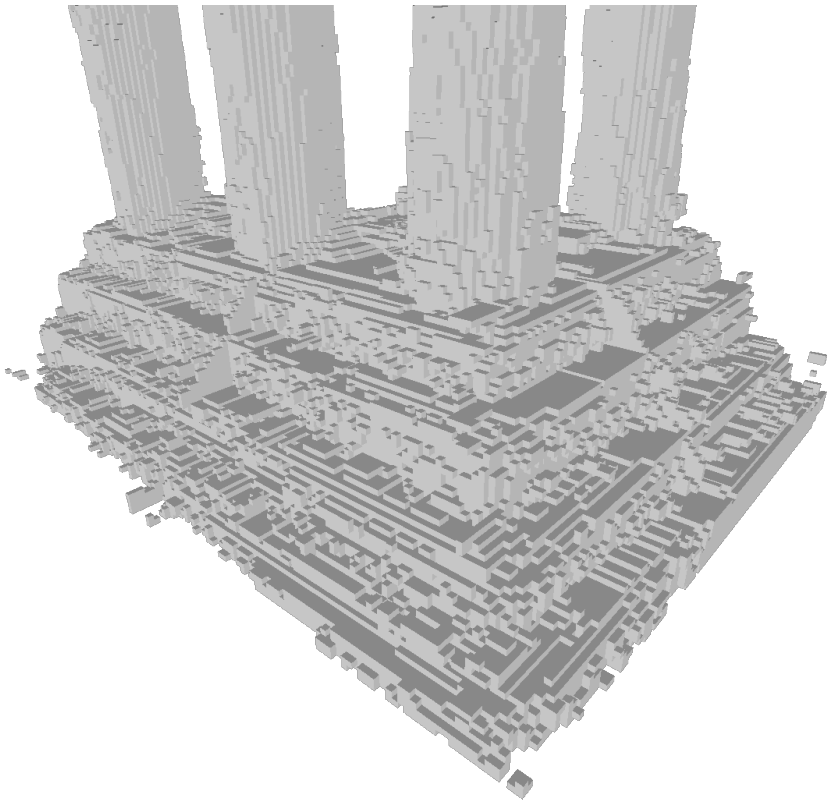


(a) 3D reconstruction with cost only

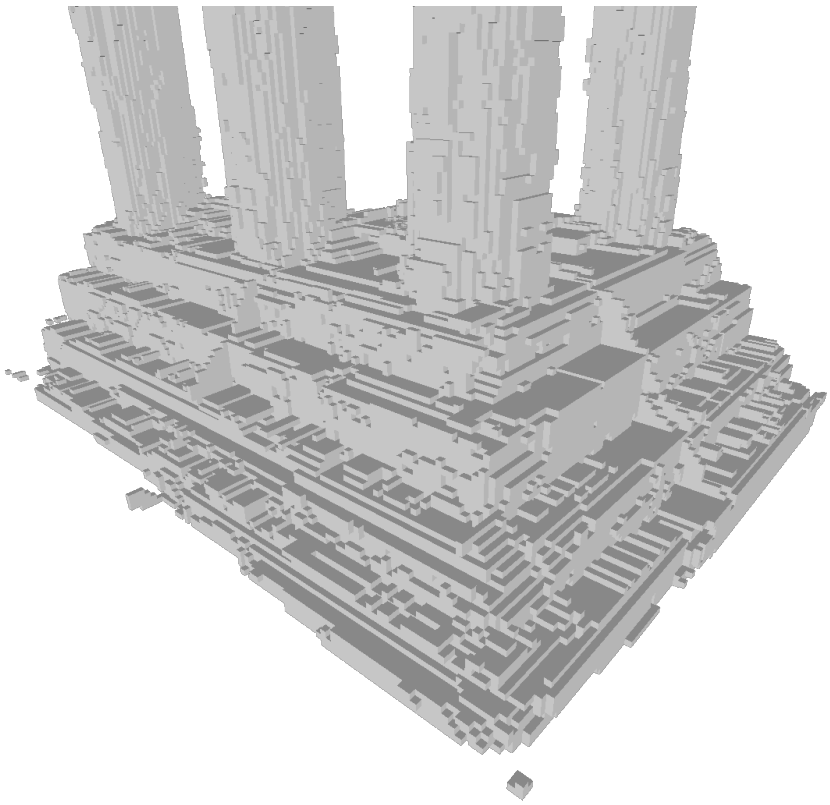


(b) 3D reconstruction with SGM

Figure 6.4: cost only reconstruction vs SGM reconstruction.



(a) 3D reconstruction with cost only



(b) 3D reconstruction with SGM

Figure 6.5: cost only reconstruction vs SGM reconstruction.

Chapter 7

IMPLEMENTATION

This chapter gives some details about the application developed during this thesis.

7.1 Application overview

An application called mview was developed for this master thesis and it was used to visualize the surfaces computed from a set of images. Figure 7.1 shows a snapshot of the application.

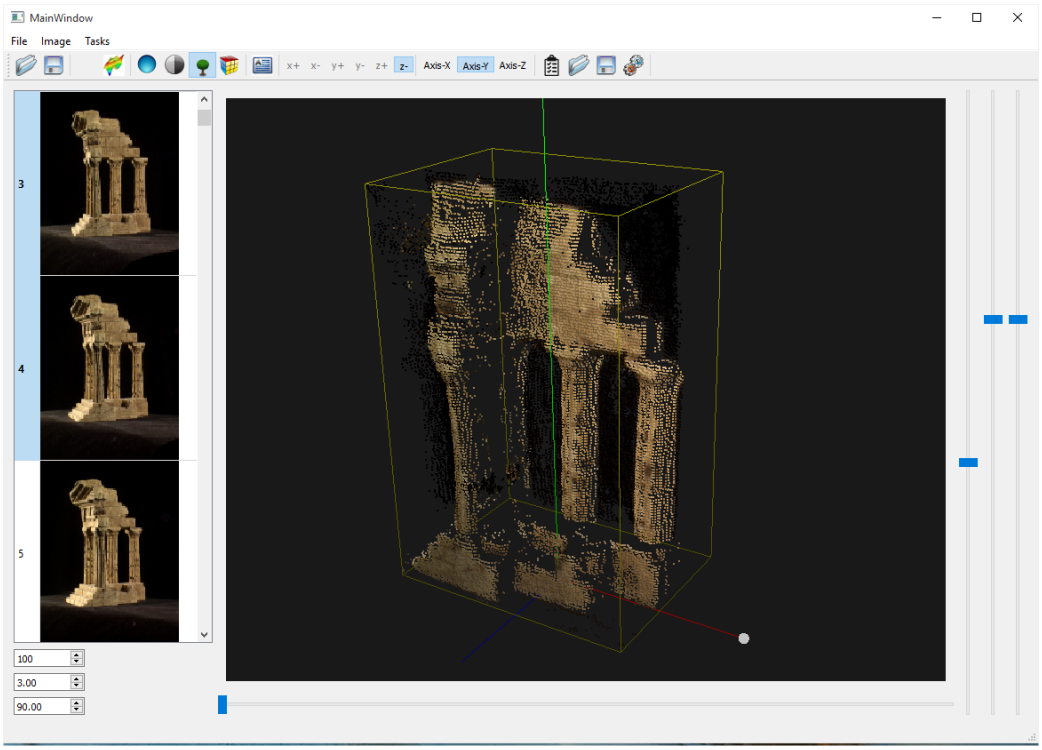


Figure 7.1: mview application.

7.2 Open source components

Multiple open source components were used in this application:

- cmake 2.8.12 (http://www.cmake.org/)
- opencv 2.4.8 (http://opencv.org/)
- Open Scene Graph 3.3.0 (http://www.openscenegraph.org/)
- Qt 5 (http://www.qt.io/)
- Intel threading building block (tbb) 4.2 (https://www.threadingbuildingblocks.org/)
- eigen 3.2.4 (http://eigen.tuxfamily.org/)
- google glog (https://code.google.com/p/google-glog/)
- ceres solver (http://ceres-solver.org/)
- Qt creator (http://www.qt.io/ide/)
- google test (https://code.google.com/p/googletest/)

The application was targeted for a x64 cpu. A target of 64 bits, as opposed to 32 bits, was used because the memory usage could go higher than 4 Gigs. The application was written in C++11.

7.3 Scripting capability

The initial usage of the application was to interactively process a selected set of images, but as the application improved a scripting capability was added to the program. This allowed to have a listing of tasks to execute specified in a text file. The scripting capability are simple, here is an example:

```
clear;
boundbox( 0.0, 0.00189, 0.0, 0.10822, -0.05784, 0.06249 );
directory;
setimagefile( "dinoRing\\dinoR_par.txt" );
theta(0, 1.570796326, 3.1415926, 4.7123889 );
convexhull( 0.09, 20 );
sgm( 0.00005, 0.02 );
```

7.4 Multi-threading

The application uses Intel TBB for multi-threading. It mostly relies on the parallel-for command has can be seen in the following code snippet:

```
tbb::parallel_for(blocked_range<int>(0, voxside),
          [&](const blocked_range<int>& r) {
      for (int x = r.begin(); x != r.end(); ++x)
        for (int y = 0; y < voxside; y++)
          for (int z = 0; z < voxside; z++) {
            int vxlndx = voxelndx(m_maxlevel, x, y, z);
            Eigen::Vector4d pw = getWorldPosition(m_maxlevel,
                                        m_viewpointsvec[0], x, y, z);
            if (pw[0] < m_boundbox[0][0] || pw[0] > m_boundbox[0][1] ||
                pw[1] < m_boundbox[1][0] || pw[1] > m_boundbox[1][1] ||
                pw[2] < m_boundbox[2][0] || pw[2] > m_boundbox[2][1]) {
              m_octree[vxlndx] = VOXEL_EMPTY;
            }
          }
});
```

Intel TBB was selected because of the pipeline functionality it offers. This especially useful when multiple functions need to be processed when after another. Figure7.2 shows the utilisation of the cpus while rendering the 3D model. It takes 309 seconds to process the complete 3D model at maximum voxel resolution on the temple ring data set which is composed of 47 images.

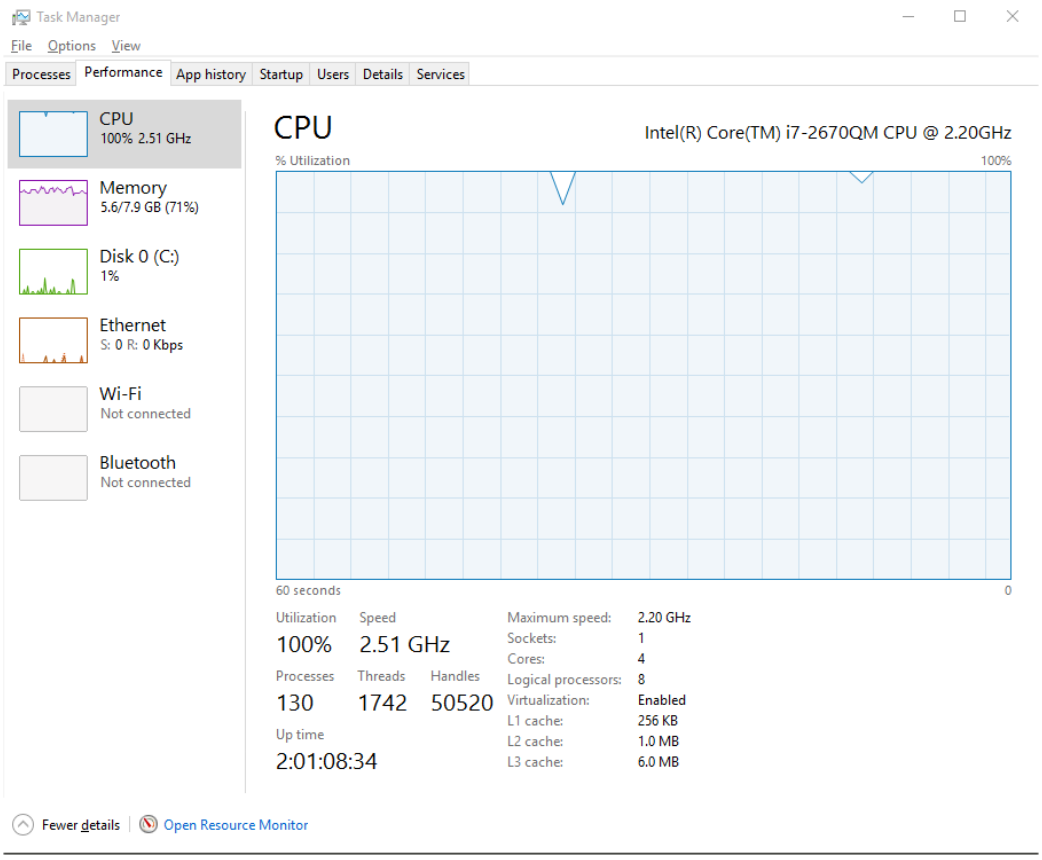


Figure 7.2: 100% Cpu usage on a 8 cpu machines

Chapter 8

RESULTS

We tested on the Middlebury multi-view dataset[1] on the dino and temple ring datasets. The evaluation of the results was subjective and was done using a mesh viewer [20]. It was subjective because the dataset used did not have a ground truth[2].

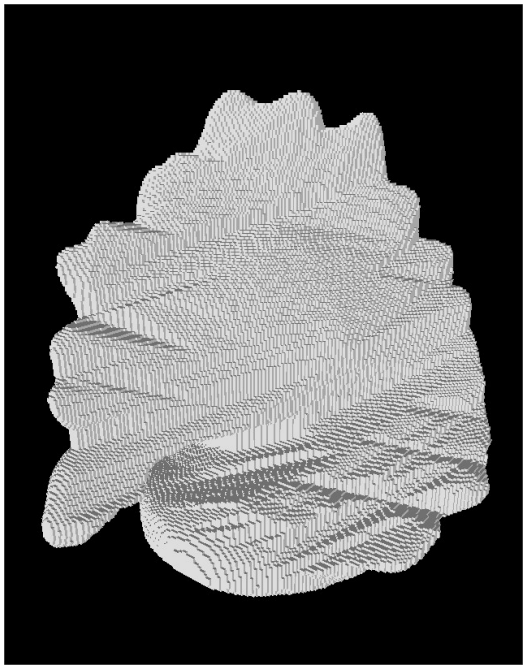
Figure 8.1 shows some results obtained from the 48 images of the Middlebury's dino ring dataset. Figure 8.1c and figure 8.1d have been generated with parameters $p1$ equal to 0.0005 and $p2$ equal to 0.05. Figure 8.2 shows a comparison with other results.

Figure 8.3 shows some results obtained from the 48 images of the Middlebury's temple ring dataset. Figure 8.3c and figure 8.3d have been generated with parameters $p1$ equal to 0.001 and $p2$ equal to 0.011. Figure 8.4 shows temple results for multiple levels (resolution). Figure 8.5 shows a comparison with other results. Figure 8.6 shows multiple views of temple at level 8.

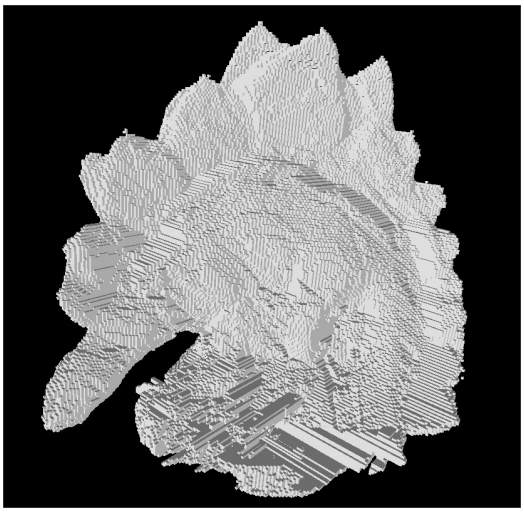
[1] http://vision.middlebury.edu/mview/data/

[2] While this is not a precise measurement, it is the method that gives the best results as far as human appreciation is concern. The idea is the same when evaluating video compression codecs where a simple PSNR does not tell much, even with a low PSNR the compressed video can still be unappealing compared to another stream with an higher PSNR value

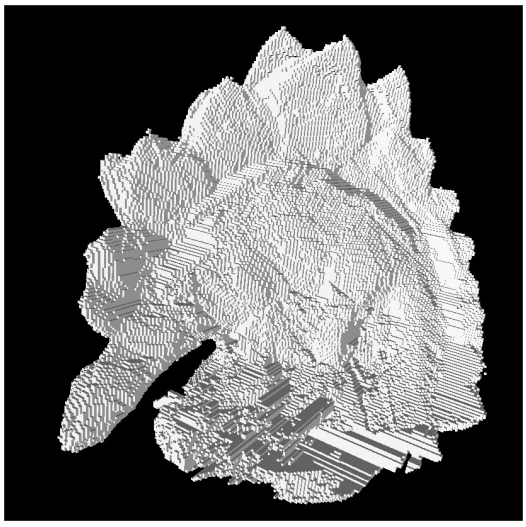
(a) Original image



(b) Convex hull

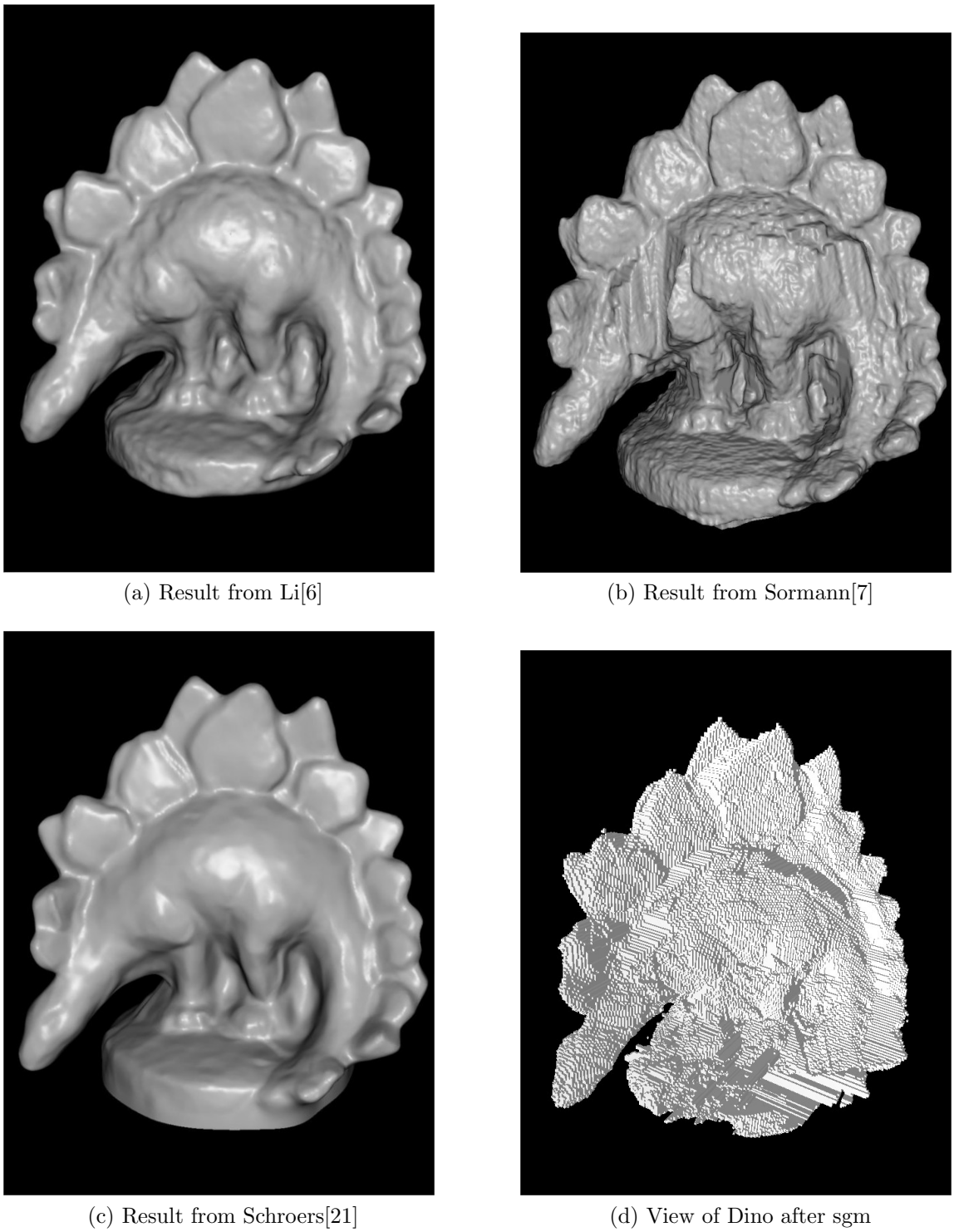


(c) View of dino after sgm ($p1$=0.0005 and $p2$=0.05)



(d) View of dino after sgm ($p1$=0.0005 and $p2$=0.05)

Figure 8.1: Results on Dino



(a) Result from Li[6]

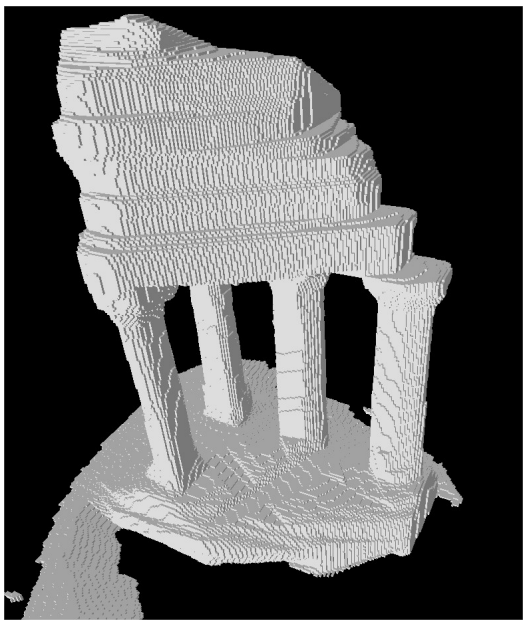
(b) Result from Sormann[7]

(c) Result from Schroers[21]

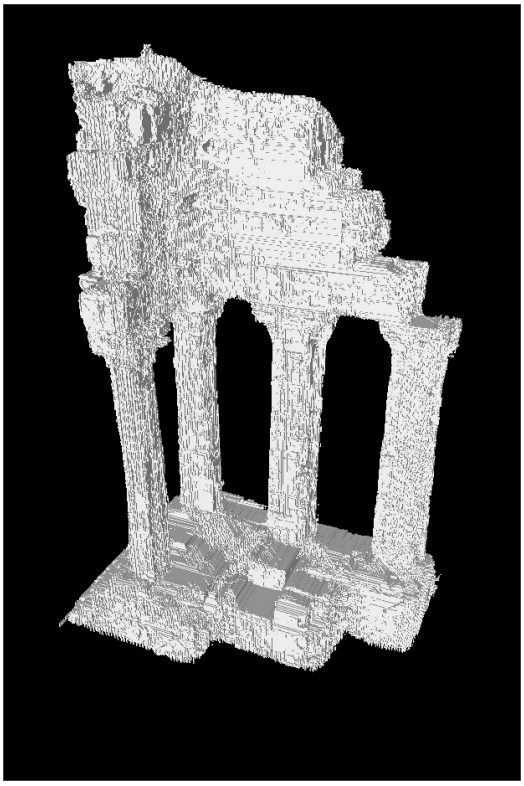
(d) View of Dino after sgm

Figure 8.2: Dino result comparisons

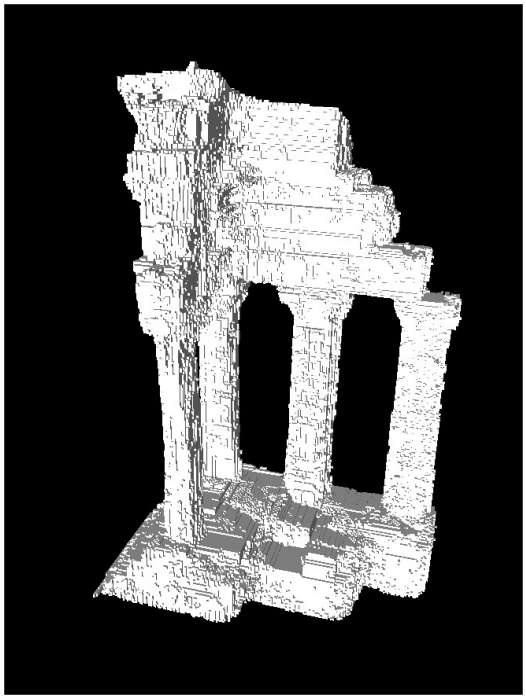
(a) Original image



(b) Convex hull

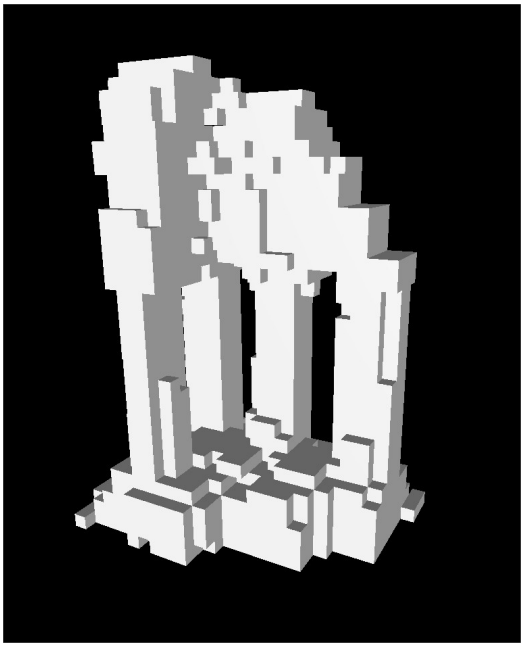


(c) View of temple after sgm at level 9 ($p1$=0.001 and $p2$=0.011)

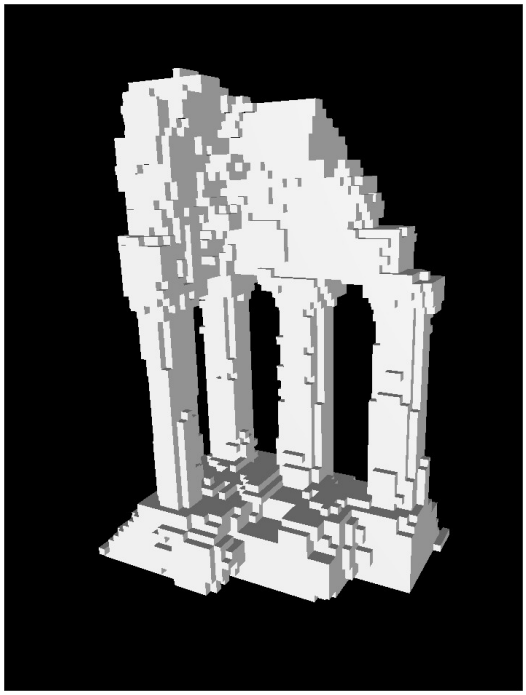


(d) View of temple after sgm at level 8 ($p1$=0.001 and $p2$=0.011)

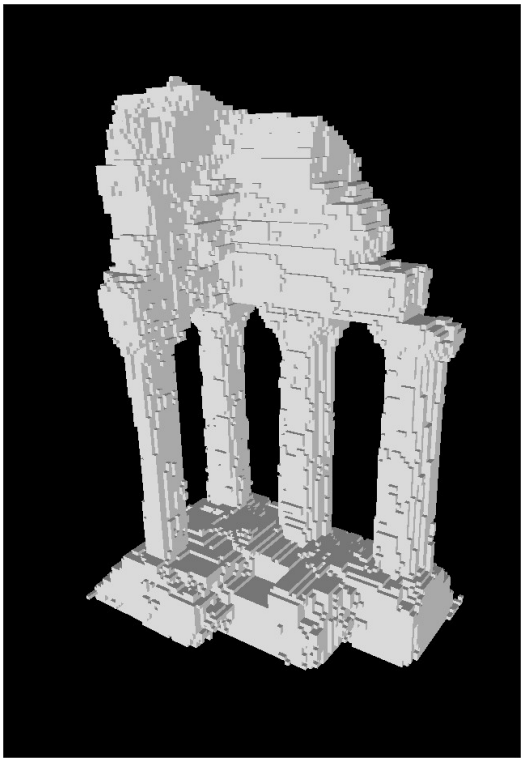
Figure 8.3: Results on temple



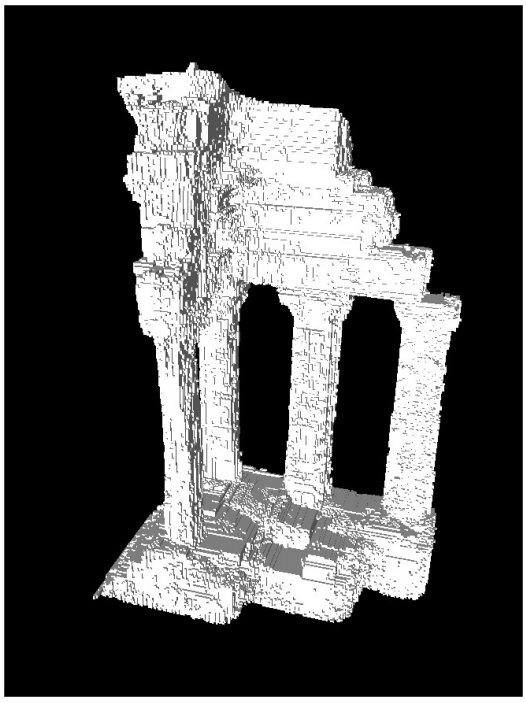
(a) Temple at level 5



(b) Temple at level 6



(c) Temple at level 7



(d) Temple at level 8

Figure 8.4: Temple results at various level

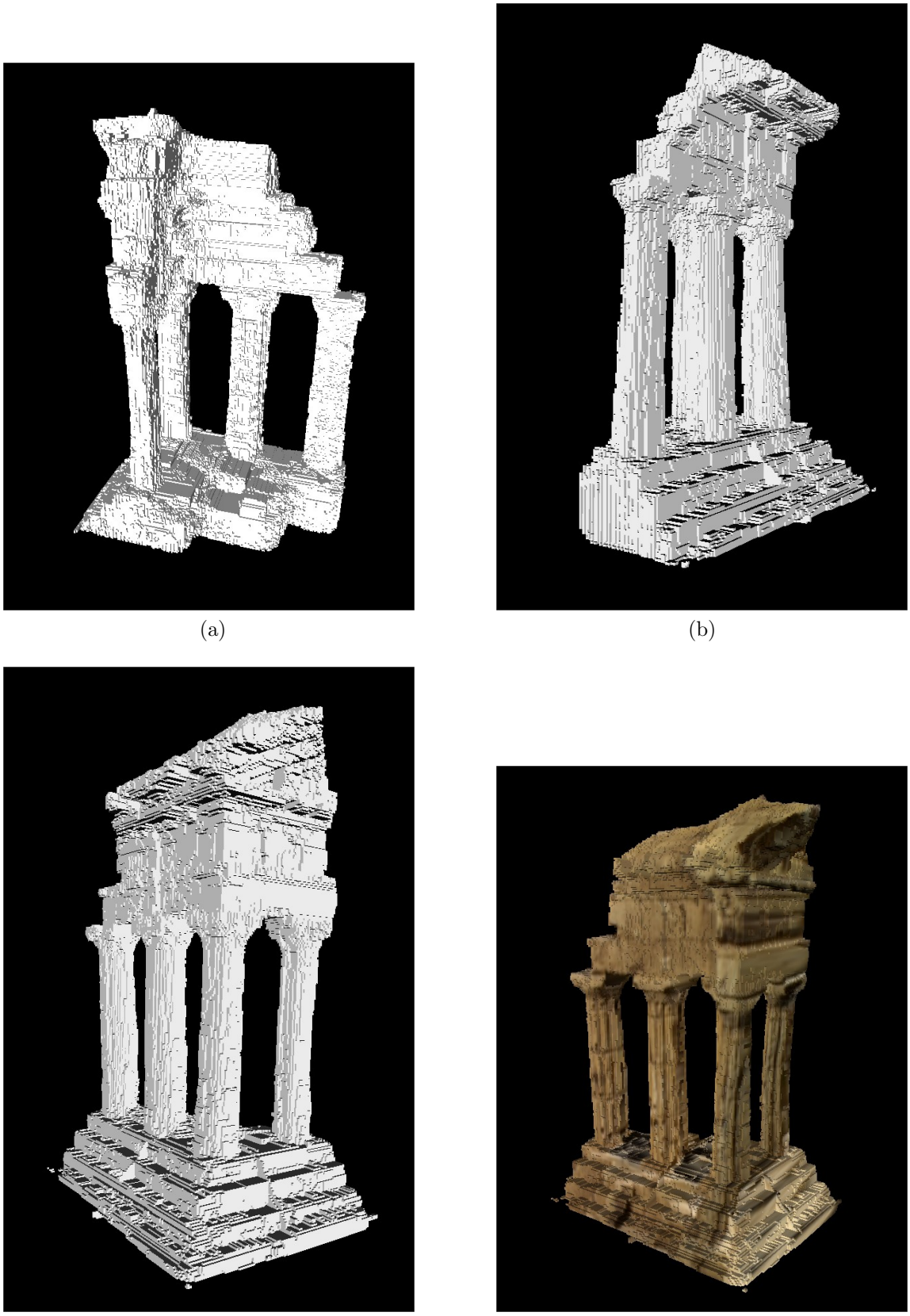
(a) Results from Li

(b) Results from Sormann

(c) Results from Schroers

(d) View of temple after sgm

Figure 8.5: Comparisons on temple



(a) (b)

(c) (d)

Figure 8.6: Results at level 8

Chapter 9

CONCLUSION

Three-dimensional multi-view aims at reconstructing a complete 3D object from the images and cameras information alone. The goal is to find all the surfaces around the 3D object. This is not a simple task because many factors influence this process. The occlusions, the noise in the images, the calibration, the errors in the calculation, and the stitching of surfaces all contribute to render this task quite complex.

The method proposed in this thesis is based on a global reference system on which virtual cameras are positioned at pre-defined viewpoints around the 3D object. A hierarchical voxel-construct, based on this global reference system, is used to reconstruct the 3D model. This voxel-construct is using an octree structure and is initialised using a convex hull. One surface is created on each virtual camera and these surfaces are used to carve the voxel-construct. The surfaces are created based on a cost function that uses multiple cameras per viewpoint. The cost function is made robust by using the angle of the real camera with the virtual camera (viewpoint). This angle is also used to penalize the images containing occluded sections. Finally a smoothing step is used to reduce the effect of noise and errors on the surface using a modified version of semi-global matching. This smoothing step also bring some improvements by using the voxel-construct to prevent surface creation in empty section.

The work in this thesis was done in C++ using openCV [22] for image processing and Intel TBB [23] for the multi-threading framework. Because all the steps involve in the 3D model creation are easily parallelizable, the program was highly scalable and was using all the available cpus at full capacity. The method also provides a way to reduce the cpu usage by working with a hierarchical model where empty areas can be detected at a lower resolution and left alone for higher resolution processing.

While providing a subjectively good model, there are still some issues with noises, scene composition, calibration errors and occlusions that affect the precision of the 3D model. The uniform background caused some problems with the image matching. The computation of a convex hull was required to address this.

Future work

From the Octree-construct, a logical next step would be to use marching-cube [24] to smooth the surface of the reconstructed 3D object. Also, using a dataset that provides a ground truth would help to obtain a quantitive evaluation of our reconstruction method.

An interesting avenue would be to use machine learning (deep neural network or a random forest like the one use on the Kinect [25]) to decide where the depth is on the surfaces.

Other interesting extension would be to support transparency and reflections, which would require the algorithm to be more flexible.

BIBLIOGRAPHY

[1] H. Hirschmuller, "Stereo processing by semiglobal matching and mutual information," *IEEE Transactions on Pattern Anaysis and Machine Intelligence*, vol. 30, pp. 328–341, February 2008.

[2] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski, "A comparison and evaluation of multi-view stereo reconstruction algorithms," *Proc. of IEEE Conference on Computer Vision and Pattern Recognition, New York, NY, USA*, 2006.

[3] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. Silva., "Computing and rendering point set surfaces," *IEEE Transactions on Visualization and Computer Graphics*, January 2003.

[4] Zhang and Zhengyou, "Iterative point matching for registration of free-form curves and surfaces," *International Journal of Computer Vision (Springer)*, vol. 13, p. 119–152, 1994.

[5] Wikipedia, "Octree." http://en.wikipedia.org/wiki/Octree. Accessed: 2015-03-30.

[6] J. Li, E. Li, Y. Chen, L. Xu, and Y. Zhang, "Bundled depth-map merging for multi-view stereo," dans *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pp. 2769–2776, June 2010.

[7] M. Sormann, C. Zach, J. Bauer, K. Karner, and H. Bishof, "Watertight multi-view reconstruction based on volumetric graph-cuts," dans *Image Analysis* (B. Ersbøll and K. Pedersen, éds.), vol. 4522 de *Lecture Notes in Computer Science*, pp. 393–402, Springer Berlin Heidelberg, 2007.

[8] C. Schroers, H. Zimmer, L. Valgaerts, A. Bruhn, O. Demetz, and J. Weickert, "Anisotropic range image integration," dans *Computer Science, Vol. 7476, 73-82, Springer, Berlin, 2012*, vol. 7476, pp. 73–82.

[9] middleburry, "Reference." http://vision.middlebury.edu/mview/eval/referenceTable.php. Accessed: 2015-03-30.

[10] R. Szeliski, *Computer Vision Algorithms and Applications*. Springer-Verlag, 2011.

[11] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International Journal of Computer Vision*, vol. 47, no. 1-3, pp. 7–42, 2002.

[12] H. Hirschmuller and D. Scharstein, "Evaluation of stereo matching costs on images with radiometric differences," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 31, pp. 1582–1599, September 2009.

[13] R. Szeliski, "Rapid octree construction from image sequences," *Computer Vision Graphics and Image Processing*, vol. 58, no. 1, pp. 23–32, 1993.

[14] R. Szeliski and P. Golland, "Stereo matching with transparency and matting," *International Journal of Computer Vision*, vol. 32, no. 1, p. 45–61, 2008.

[15] S. Birchfield and C. Tomasi., "A pixel dissimilarity measure that is insensitive to image sampling.," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 20, p. 401–406, April 1998.

[16] B. Schiele and J. L. Crowley, "Object recognition using multidimensional receptive field histograms," *Computer Vision—ECCV'96. Springer Berlin Heidelberg*, pp. 610–619, 1996.

[17] R. Szeliski, ""Stereo Correspondence"," dans *Computer Vision Algorithms and Applications*, p. 471, Springer, 2011.

[18] S. Roy and I. J. Cox, "A maximum-flow formulation of the n-camera stereo correspondence problem," dans *Proc. Int. Conference on Computer Vision*, pp. 492–499, IEEE, January 1998.

[19] H. Hirschmuller, M. Buder, and I. Ernst, "Memory efficient semi-global matching," *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. I-3, pp. 328–341, September 2012.

[20] P. Cignoni and G. Ranzuglia, "Meshlab." http://meshlab.sourceforge.net/. Accessed: 2015-06-01.

[21] Anonymous, "Gipuma: massively parallel multi-view stereopsis," dans *ICCV 2015 submission 980*, 2015.

[22] OpenCV, "Opencv." http://opencv.org/. Accessed: 2015-05-23.

[23] Intel, "Tbb." https://www.threadingbuildingblocks.org/. Accessed: 2015-05-23.

[24] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," dans *ACM SIGGRAPH Computer Graphics*, vol. 21, pp. 163–169, IEEE, July 1987.

[25] J. Shotton, T. Sharp, A. Kipman, A. Fitzgibbon, M. Finocchio, A. Blake, M. Cook, and R. Moore, "Real-time human pose recognition in parts from single depth images," dans *ACM SIGGRAPH Computer Graphics*, pp. 163–169, ACM, January 2013.

Appendix A

BACKGROUND SEGMENTATION

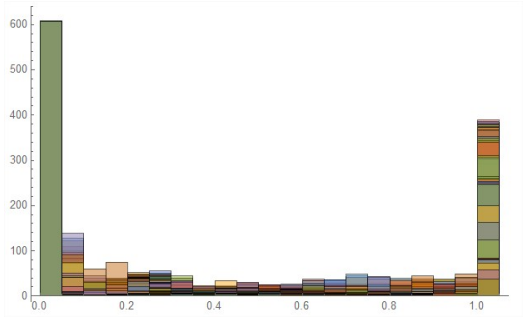
Background segmentation is the process of creating a mask of the image. The mask represents the background area. It has the same dimensions as the image but with a value of 0 for a pixel detected as being part of the background and a value of 1 for the pixels detected as being part of the object. Background segmentation is especially important when the image background is a uniform color since, this uniformity interferes with the matching process. This mask will be used to carve a visual hull as seen in section 3.8.

Figure A.1a is a base image without any modification. On figure A.1b, the intensity of the red, green, and blue channel are added together to form a black and white image. Looking at the normalised histogram of this summed image, on figure A.1c, we see that a lot of pixels are in the lower bin and this is where the background will be thresholded. So based on the histogram a threshold value of 0.12 was selected and used in the figure demonstrated here. Figure A.1d shows the mask with the threshold applied. This mask is not perfect because the inner part of the object are not selected. This is why we need to apply a morphological operation (closing) before thresholding.

Figure A.2a is the summed-image with a rectangular morphological filter applied. The morphological operator fills the inner hole, which is what we want. The result of the morphological operator is perfect because the mask is not smaller than the original image. It is very important that the visual hull created from the masks be equal or bigger than the actual object. The depth retrieved after SGM will further refine the 3D volume. Figure A.2b demonstrates the final results of a mask generated for the image in figure A.1a.

(a) base image

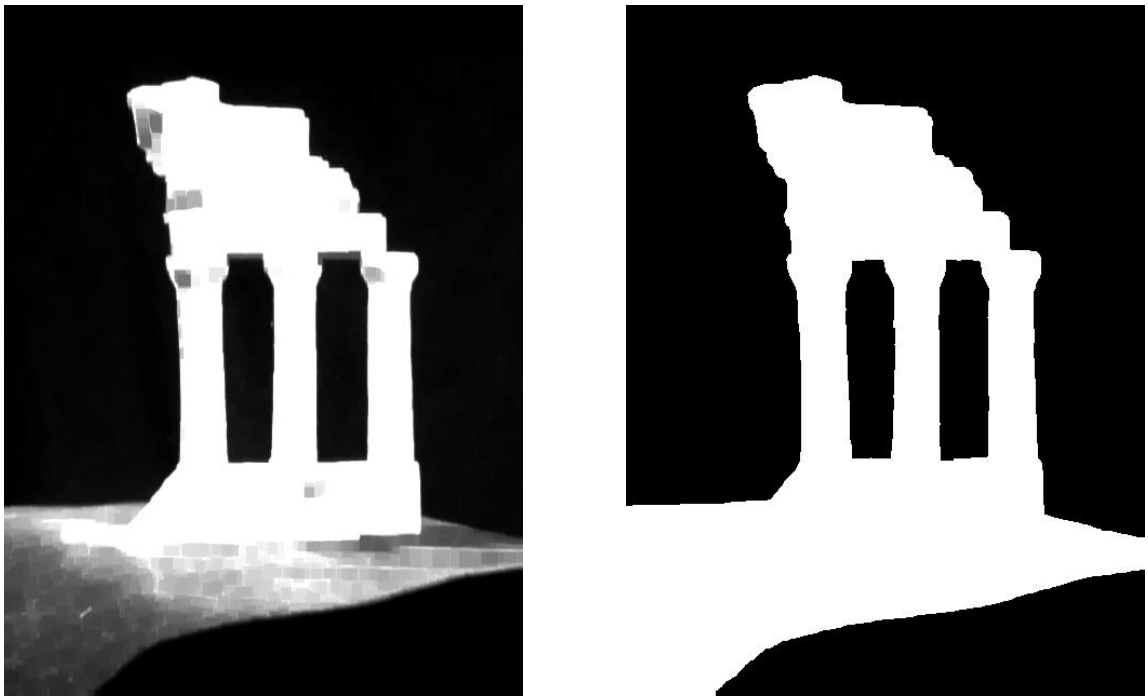
(b) image created by summing all 3 channels R,G,B



(c) histogram of the normalized sum

(d) simple threshold on sum

Figure A.1: Background segmentation



(a) Morphological closing operator applied (b) Threshold on morpho-image

Figure A.2: Morphological operator

Appendix B

OCTREE

This appendix presents how to access an octree of size $\beta \times \beta \times \beta$. Assuming a 0 based position where the position starts at 0 and ends at $\beta - 1$ in each dimension. A voxel at position (l, x, y, z), where l is the level, will be composed of the following voxels:

- position $(l + 1, x * 2, y * 2, z * 2)$
- position $(l + 1, x * 2 + 1, y * 2, z * 2)$
- position $(l + 1, x * 2, y * 2 + 1, z * 2)$
- position $(l + 1, x * 2 + 1, y * 2 + 1, z * 2)$
- position $(l + 1, x * 2, y * 2, z * 2 + 1)$
- position $(l + 1, x * 2 + 1, y * 2, z * 2 + 1)$
- position $(l + 1, x * 2, y * 2 + 1, z * 2 + 1)$
- position $(l + 1, x * 2 + 1, y * 2 + 1, z * 2 + 1)$

As an example, the voxel of level 3 at position (3, 1, 4, 0) will be composed of the voxels of level 4: (4,2,8,0), (4,3,8,0), (4,2,9,0), (4,3,9,0), (4,2,8,1), (4,3,8,1), (4,2,9,1), (4,3,9,1).

It is possible to retrieve the voxel at a level $l - 1$ by using the position $(l - 1,\ x/2,\ y/2,\ z/2)$.

For example, a voxel at position (4,3,9,0) will refer to the voxel at position (3, 1, 4, 0).

Since the octree is a hierarchical construct we must ensure that it remains consistent between levels. When a voxel value is changed, a percolation operation is required to update every parent of the voxel. This operation is very efficient because the number of level is $O(\log \beta)$.