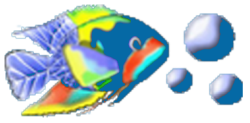
AALBORG UNIVERSITY

Low cost 3D Scanner

Background segmentation and Visual hull obtaining

Master thesis
System Software Enginering
Computer Science Department
AAU Aalborg University

Partnership:
Create It Real



Supervisor(s):
Yannick LE MOULLEC
Alexandre DAVID

External(s):
Jeremie Pierre GAY

Aurthor:
Yi Li

Used libraries and frameworks: Windows Presentation Foundation, AForge.NET, OpenCV, GRAFT

Acknowledgements

The last two semesters are the most fantastic period in my life so far. As a guest student in Aalborg University, I have had quite different experiences both in study and in personal life from my previous life in China.

I want to thank my supervisors Yannick LE MOULLEC and Alexandre DAVID. It is their persistent encouragement and supervision that makes us successfully finish the 3D scanner project. And also thanks for their consistent fast and positive responses to my questions and requests. The nice and easygoing personalities of both of them have really given to me very good impressions.

I will thank the CEO of Create It Real Aps Jeremie Pierre Gay. Thanks to his proposal of 3D scanner project which brings us an interesting and fruitful researching exprience. Also thanks to his tireless instructions during the researching process. He has also provided a lot of convinient and joy to us in terms of working environment, beers and parties in his office and home.

At last, very big thanks to the 3 french groupmates: Vincent FOUILLARD, Romain HERRY and Emmanuel DHIVER. The result of 3D scanner project cannot be as good as it has now without all your hard efforts. I have learned quite a lot from each of you and enjoyed very much the time spending together with you wherever. Sincerely wish you all a bright future!

Abstract

3D scanning technology has stepped into the daily life from the industrial domain in the past few years. 3D scanners which utilize simple hardware but possessing good capability to produce high quality 3D models emerge continuously. This thesis is working on such a 3D scanner, which is based on a single camera, assissted by a special tracker and can produce simple 3D models of objects. This thesis is a continuation of the 9th semester's master project "Low cost 3D scanner" [7] which devides the 3D scanner system into two modules: 3D point cloud acquisition module and 3D mesh reconstruction module. Parts of the acquisition module and a functional reconstruction module have been implemented. So this thesis focuses on the implementing of the acquisition module and the integration of the 3D scanner system.

An iterative algorithm which combines Gaussian Mixture Models (GMM) and space-carving-like method has been proposed to obtain the silhouettes and the visual hull point cloud of the scanned object simultaneously. The video frames and the starting projection silhouettes obtained from the video frames are used to train the initial GMM to generate GMM silhouettes; the GMM silhouettes are used by the space-carving-like method to generate visual hull point cloud which is then projected to some virtual pinhole cameras to obtain new projection silhouettes; the new projection silhouetts and the frames are used to update the GMM to generate new GMM silhouettes. The iteration goes on like this until some convergence criterion is satisfied. To support the space-carving-like method, a perspective projection algorithm is implemented and later integrated with Aforge.net's glyph estimation result. The old tracker is also modified by increasing the glyphs' kinds from 1 to 4 to facilitate the space-carving-like method and attaching to it some chessboard pattern to help shadow elimination and background subtraction.

Finally, the 3D scanner system is integrated and can generate 3D mesh model from video frames acquired by a normal camera. The 3D scanner is easily manipulated and can generate the 3D mesh model within an acceptable time period. It is thus hopefully to meet the practical requirement after some optimizations.

Contents

Chapter 1

Introduction

1.1 Origin of the project

This project is a continuation of the 9th semester's project called "Low cost 3D scanner" [7]. It is a proposal of the Danish startup company *Create It Real Aps*. The aim of the project is to establish a 3D scanner with low hardware requirements and easy usability. Last semester, the project had its initial research conducted. The main architecture and the basic theory have been established. Parts of the functionalities have been implemented. This project aims at finishing the remaining implementation tasks and put the system to practical use. Now let us recall some motivations and accomplished work first.

1.1.1 Review of 3D printing and 3D scanning

3D printer is used to make real objects from digital files. Opposite to the traditional subtractive manufacturing, 3D printing uses an additive process. It melts some materials such as metal or plastic to make a layer above the previous layer to construct an object gradually. In the recent years, 3D printing has entered the personal market from the industrial domain. The demand for 3D printing for entertainment and other personal use has dramatically increased. And lots of companies are making efforts to reduce the cost of the 3D printer to meet the demand. *Create It Real Aps* is one among them.

Meanwhile, a complementary technology to 3D printing is also gaining more and more attentions. That is 3D scanning. It collects the 3D information of an object like its shape and appearence. In the domain of the entertaiment industry, it is used for producing movies and video games, industrial design, orthotics, prosthetics and reverse engineering, prototyping, quality control and documentation of cultural artifacts[18], 3D scanning has significant usages both on accelerating the design process of 3D model and on digitalizing realistic objects. This project is focusing on how to make a low cost and practical 3D scanner.

The combination of a 3D printer and a 3D scanner can realize the concept of "3D photocopy" which is to make a replicate of a real object.

1.1.2 The project's relationship to *Create It Real Aps*

Create It Real Aps (below will be stated as CIR for short) is a Danish startup company founded in March 2009 by its current CEO, Jeremie Pierre Gay. The main aim of the company is to produce low cost 3D printers that can be afforded by both companies and normal families. Now it is working on its third generation prototype and has a suite of software to control the 3D printer. Except the personal market, it is also exploring the potential market like organ printing and teaching prop printing.

Involving a 3D scanner into the current software suite has both been the company's will and the market's need. With the 3D scanner, it is easier to obtain a 3D model to print. And then the company's product can really peform "3D photocopy" that will benefit in many potential aspects.

So the company has proposed the idea of 3D scanner to Aalborg University. Then it became the subject of the 9th semester's project of the department of computer science. Below, the former work is summarized.

1.1.3 Former work

The accomplished work is supposed to use a single camera (normally a webcam of a laptop) to obtain the object's video and then reconstruct the 3D model. To assist the acquisition, a tracker is used to locate the scanned area, the tracker is actually an A4 paper with printed glyphs [2] which are a kind of markers. As a feedback, a virtual cube is drawn on the tracker on the screen to indicate that the system is functioning and to specify the area to place the scanned object. The object is then placed in the specified area and rotated with the tracker in front of the camera to supply the object's 360° view. The illustration of how the system works is in Figure 1.1

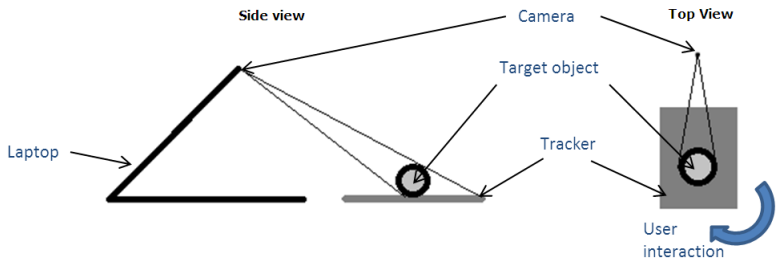


Figure 1.1: The illustration of the 3D scanner system.

Visual hull method [9] is the foundation of the system. It exploits silhouette pictures to reconstruct the object's 3D model. Aforge.net [1] is the main framework used to process the video and the images. Aforge.net is used to recognize the glyphs on the tracker and to estimate their rotations, thus to locate the tracker and trace the rotation of the tracker to extract multi-view key frames according to a predefined angle. Point Cloud Library (PCL) [15] is the main inspiration for the 3D mesh model reconstruction. A triagulation algorithm called greedy projection used in the project was inspired by the PCL document.

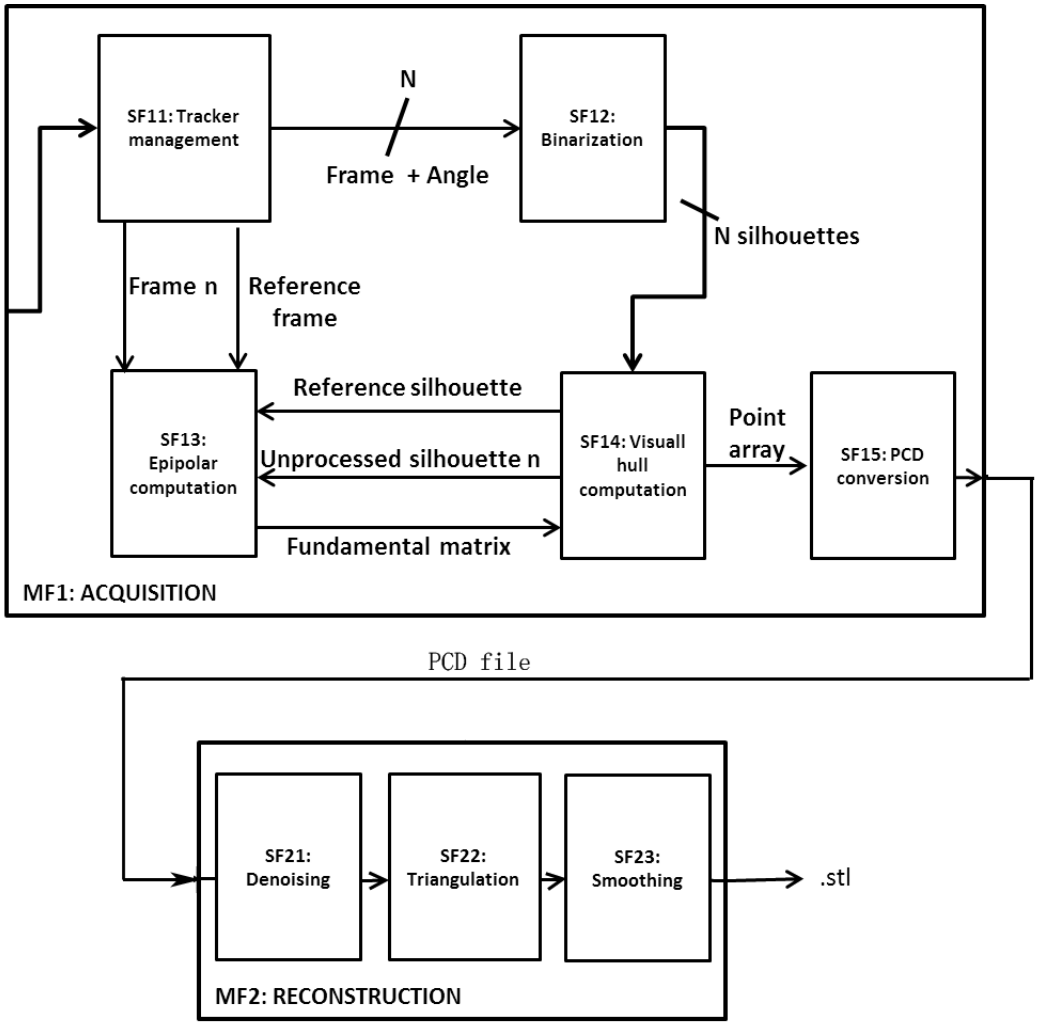


Figure 1.2: Basic architecture of a 3D scanner, as proposed in our former work [7]

Figure 1.2 is the basic architecture of the previous work. The whole system is split into two modules : MF1:Aquisition and MF2:Reconstruction.

The acquisition module is in charge of generating the point cloud which is the representation of the object in the form of a set of points. It has five submodules described in details below.

- **SF11** is supposed to perform the glyph recognition and rotation estimation tasks exploiting Aforge.net to extract N key frames from the video stream according to a predefined angle and pass the N key frames with their rotation angles to SF12. SF11 also forwards N key frames to SF13, but this time, the virtual cube is invovled in the frames with 8 points on the cube edge highlighted.
- **SF12** provides the silhouette pictures obtained from the key frames through background subtraction and binarization. The silhouette pictures contain the contour of the object with the object in white color and the background in black color.
- **SF13** provides the fundamental matrix needed by SF14. SF14 asks SF13 for two silhouette pictures' fundamental matrix each time, SF13 checks the corresponding key frames of the silhouette pictures got from SF11 and computes the fundamental matrix through the 8 virtual points in each key frame based on epipolar geometry.
- **SF14** compares the reference silhouette which is the first silhouette with all the other silhouettes to compute the point cloud describing the surface of the viual hull of the object, and pass on the point cloud to SF15.
- **SF15** stores the point cloud into a PCD file format and passes the result onto MF2.

The reconstruction module accepts the PCD file obtained from MF1, and builds a 3d mesh model from that. It has three submodules described in details below.

- **SF21** calculates the statistical distribution of the mean distance between one point and its K nearest neighbors. Then some proportion of the distribution is labeled as outliers and is eliminated from the point cloud.
- **SF22** uses a greedy projection method to reconstruct the 3D mesh model. The greedy projection tries to find as many neighbors as possible of one point to form triangles with this point, and grows the mesh gradually in this way.
- **SF23** smoothes the surface of the mesh to make a refined model.

So far, only SF11 is completely implemented in MF1. Some ideas have been generated for SF12, SF13 and SF14 and are waiting to be realized. For MF2, SF21 and SF22 are generally finished. 3D mesh can be reconstructed on giving some point cloud. But they still need optimization to increase robustness and performance. And SF23 stays in theoretic stage.

To know more details about the implementation of the system, please refer to the report of last semester [7].

1.2 Scope of work

1.2.1 Constraints from the existing software

The 3D scanner system which this project is aiming at is supposed to be integrated into the software suite of CIR. So it has to follow the framework and programming language that CIR is currently using, and also the open library choosing policy of CIR. Specifically, the following constraints should be followed:

- .NET/C# based software.
- Compatible with the WPF (Windows Presentation Foundation) framework.
- Complying with the MVC architecture when new made module is integrated into the current system.
- Free open source libraries which will not expand its open source property to the CIR's system.

1.2.2 Constraints from former work

The design principle of the 3D scanner inherits from last semster's work, that is to make a 3D scanner with low hardware requirement and low environment requirement which can be used by a user without preliminary knowledge of 3D scanning and under a normal scene like on a desk in an ordinary room. The 3D scanner should be financially affordable, so free open source libraries are promoted.

So this project will keep the solution of a single camera assisted by a tracker which is demonstrated in Section 1.1.3, as well as keeping the user-machine interaction routine defined in Section 1.1.3. The scanned object will be placed within a $15cm \times 15cm \times 15cm$ cube decided by the tracker. And the object should be some simple object which is convex, well lightened, not shiny or transparent, because the visual hull approach cannot display too many details and cannot deal with concave surfaces.

Most parts of the architecture which have been revealed in Figure 1.2 will be preserved. Only some modifications will take place on the inside structure of some module. The visual hull approach will still be the main idea for obtaining the point cloud.

1.2.3 Task description

This semester, the research group has decreased from 4 to 1 person. So the research scope will be concentrated. The main research target is the finishing of MF1, the point cloud acquisition module.

For the acquisition part, the research will start from the background segmentation job of the obtained key frames to extract silhouette pictures of the object. And then practical visual hull computating method need to be found. According to the new findings on this area, the submodules SF12, SF13 and SF14 will be altered by another algorithm. After the point cloud is successfully generated, multiple tests will be performed to check MF1's performance.

For the optimization of the reconstruction part, a thorough examination of the current work will be performed to eliminate implicit bugs. And then based on the new obtained point cloud, testing will be performed to find potential optimization methods.

Chapter 2 provides theories on background segmentation and an iterative algorithm to obtain the visual hull. Chapter 3 illustrates the whole picture of the system after modification of the acquisition part. Chapter 4 gives the detailed implementation of the acquisition part and Chapter 5 lists possible optimizations for the reconstruction part. At last, Chapter 6 summarizes this semester's work and point out the possible future work.

Chapter 2

Related work

2.1 Silhouette segmentation and visual hull obtaining

2.1.1 A brief summary about various silhouette segmentation methods

A typical background subtraction practice assumes the background is always constant while the foreground can vary. So some photometric information such as gray scale, color and texture is taken into account to separate foreground and background. For example, the chroma-keying approaches are included in this category. They usually use a uniform blue or green background, and are widely used in video making industry.

Statistic models are used for nonuniform background. Usually, a learning step is required to construct the statistic models for background or foreground or both. Then the pixels on the image can be identified as foreground or background according to the models. Sometimes, the models can evolve when the background changes, or under an iterative algorithm; some user interventions or assumed constraints may also be needed [5]. Gaussian Mixture Model (GMM) is one of most popular statistic models.

Except this, graph cuts method has been introduced by some papers, like [4]. It exploits an energy function which often has two parts. The first part evaluates the discrepancy of the segmentation result to the original image based on some probabilistic statistic model. The second part enhances the smoothness or boundary constraints over the image regions. Graph cuts is often performed through a max-flow/min-cut algorithm to minimize the energy which is called energy minimization [3]. Usually, a user interaction is required for the implementation of this method [16] and the workload can be significant according to the complexity of the image. It is originally performed in 2D space but can also be expended to 3D space which is called volumetric graph cuts [17].

Regarding 3D space, it is obvious to know that different images of the same object have a common 3D region in the space. This is the spatial coherence among the images. Utilizing this 3D information can help improving the quality and the correctness of the segmentation. Some depth maps from stereo camera systems are first used to combine with color information. Then several iterative methods are introduced to utilize the 3D information. [27] has proposed a way using color consistency in individual image and projection consistency among different images. First, it segments the images into different regions according to the color information. Then it carves the

silhouettes and the visual hull iteratively from big and rough results to smaller and more accurate results. Both [5] and [10] also use an iterative way to improve the result. But they have combined the GMM and graph cuts method to improve the quality of final result. The difference between them is that[5] computes a visual hull result simultaneously with the silhouettes while [10] does not perform the visual hull computation except the initializing stage.

2.1.2 Gaussian Mixture Models(GMM)

Gaussian distribution(also known as normal distribution) is a continuous probability distribution. It has a bell-shaped density function, See in Figure 2.1

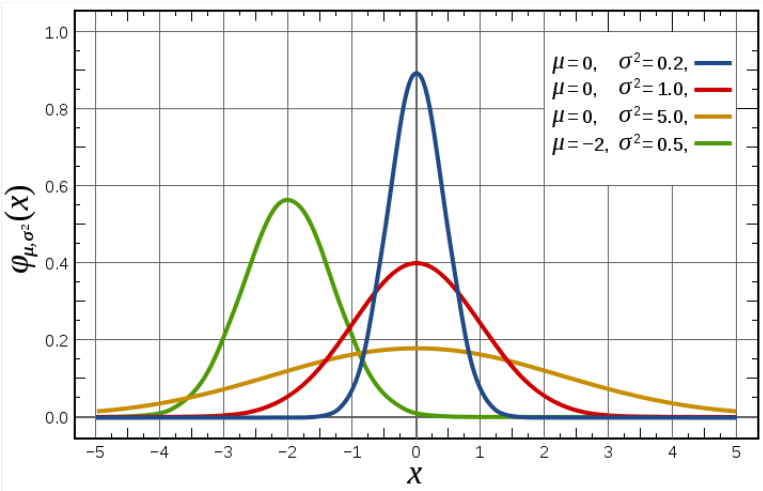


Figure 2.1: Bell curves of several single gaussian distributions. [23]

And multi-dimensional gaussian distribution's density function is like this in mathmatical expression:

$$\mathcal{N}(x;\mu,\Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}}\,|\Sigma|^{\frac{1}{2}}} \exp[-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)] \tag{2.1}$$

x is a n-dimensional sample, μ is the mean value and Σ is the covariance matrix and is often diagnoal.

Then Gaussain Mixture Model(GMM) is a weighted sum of several gaussian distribution components. It often takes the form of :

$$p(x) = \sum_{i=1}^{M} \omega_i \cdot \mathcal{N}(x;\mu_i,\Sigma_i) \tag{2.2}$$

M is the number of the components it contains. ω_i is each component's weight.

GMM is widely used for *clustering*. It can classify several clusters corresponding to the number of its components. Clusters are assigned by selecting the component providing the maximum probability. See in Figure 2.2.

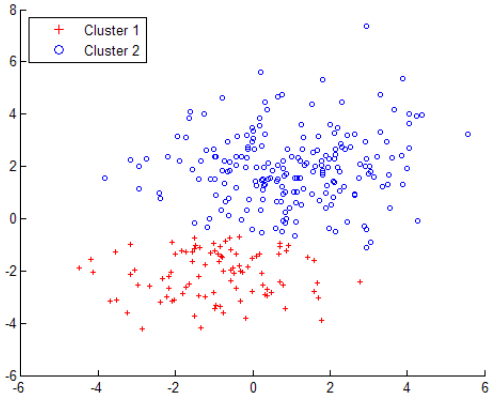


Figure 2.2: Clustering by GMM with two components. [23]

The combination of weighted gaussian components gives GMM the power to approximate arbitrary feature densities which is called *density estimation*. See in Figure 2.3. The feature densities can come from speeches or images, etc, thus give GMM the ability to perform the speech recognition and image segmentation tasks and so on.

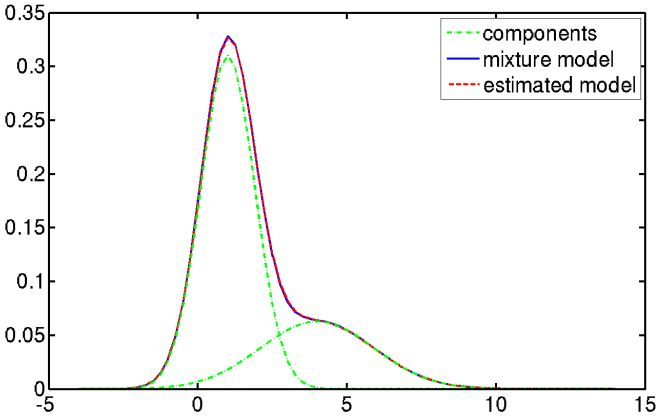


Figure 2.3: Mixture Curve of 2 components. [12]

Usually, we assume an unknown distribution can be approximated by GMM. Given a set of sample $X = \{x_1, x_2, \ldots, x_n\}$ of the distribution, we then estimte the parameter $\theta = \{\omega_i, \mu_i, \Sigma_i | i = 1, 2, \ldots, M\}$.The solution is to find the parameter θ which maximizes the likelihood of GMM given the sample data. In another word, to maximize the likelihood function $p(X|\theta)$.

$$p(X|\theta) = \prod_{i=1}^{N} p(x_i|\theta) = \prod_{i=1}^{N} \sum_{j=1}^{M} \omega_j \cdot \mathcal{N}(x_i; \mu_j, \Sigma_j) \tag{2.3}$$

And the most popular method to sovle this problem is: Expectation-Maximization (EM) algorithm.

This method starts with a initialized parameter θ, and then uses a two-step iterative way to optimize the parameter. Because the quality of the initial data affects the result seriously, we need a fast and good initialization procedure. And often K-Means is utilized.

The two-step iteration [28]:

- The E-step computes the *a posteriori* probability of component m of generating a single sample x_i:

$$\gamma(i,m) = \frac{\omega_m \mathcal{N}(x_i|\mu_m, \Sigma_m)}{\sum_{j=1}^{M} \omega_j \mathcal{N}(x_i|\mu_j, \Sigma_j)} \tag{2.4}$$

- The M-step then caculates each parameters of θ that will maximize the likelihood function.

$$\mu_m = \frac{1}{N_m} \sum_{i=1}^{N} \gamma(i,m) x_i \tag{2.5}$$

$$\Sigma_m = \frac{1}{N_m} \sum_{i=1}^{N} \gamma(i,m)(x_i - \mu_m)(x_i - \mu_m)^T \tag{2.6}$$

$$\omega_m = \frac{N_M}{N} \tag{2.7}$$

Where $N_m = \sum_{i=1}^{N} \gamma(i,m)$

In image segmentation, GMM can be used to construct models for both foreground and background and then provide the possibility of a pixel belonging to foreground or background.

2.1.3 Graph cuts and energy minimisation

Many early vision problems such as image restoration, reconstruction and segmentation can be considered as a discrete or continuous optimization problem. And frequently, they are formulated in terms of energy minimization. An energy function is presented to describe the properties of the problem. The minimum value of the energy is to be obtained to get the global optimization of the problem. The energy function can be used to construct a graph, and the energy minimization problem can then be sovled by finding a minimum cut of the graph through a max-flow/min-cut method which will be depicted later.

The energy function is generally in the form like in Equation 2.8 [3]:

$$E(L) = \sum_{p \in P} D_p(L_p) + \sum_{(p,q) \in N} V_{p,q}(L_p, L_q), \tag{2.8}$$

Here $p, q \in P, P$ is the image and p, q are pixels in the image P.$L = \{L_p | p \in P\}$ is a labeling of the image. Depending on the concrete problem, the labeling has different expressions. For example for the image segmentation, labeling means that to label a pixel as foreground or background by L_p. N is a set of all pairs of neighboring points. The function $D_p(\cdot)$ describes the disagreement between the labeling and the observed data based on some prerequisited likelihood function. $V_{p,q}$ enhances

smoothness by penalizing the discontinuities between neighboring points. Different papers [5, 10] have presented their own implementations for the functions $D_p(\cdot)$ and $V_{p,q}$.

The graph $G = V, E$ is constructed from the energy function and the image. V is the set of vertices of the graph which includes all the pixels in the image and two special terminal points. The two special terminal points are called source and sink, for example in image segmentation, representing foreground and background respectively. E is the set of the edges. Each pair of neighboring points in N has an edge. Each point also has two edges connecting it to the source and the sink. Each edge has a weight which is obtained from the two parts of the energy function.

Here we give a simple description of the max-flow/min-cut method: if the edge of the graph can be interpreted as a water "pipe" and the "pipe"'s capacity of letting how much water go through it is corresponding to its weight. Then the maximum flow is the maximum amount of water that can be sent from the source to the sink. When the maximum flow is achieved, a set of edges will be in their full capacity. These edges will divide the graph into two separated parts, one belongs to the source and the other belongs to the sink. So the set of edges is corresponding to a minimum cut of the graph.

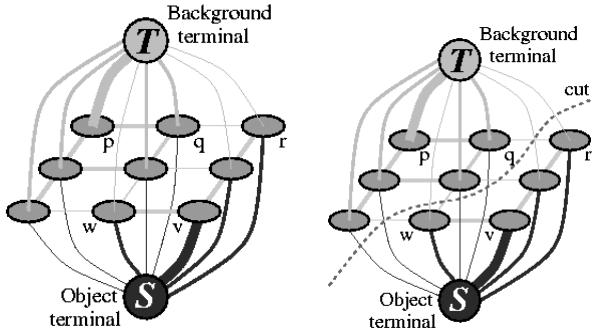


Figure 2.4: Illustration of the graph and the max-flow/min-cut method. [4]

The graph cuts method can also be used to 3D voxels besides 2D pixels to get the visual hull.

2.1.4 Space carving

Space carving is a method to construct an unknown 3D shape from multiple photographs of the shape taken at known but arbitrarily distributed viewpoints. A concept named *photo-consistency* [8] is defined for the method. A point in the 3D space is *photo-consistent* only if ,on the photographs it can be seen, 1)it does not project to the background pixels(when the background can be explicitly identified); 2)its projections on these photographs have high color similarity. On the photographs that the point can not be seen, the point is trivially *photo-consistent*.

The general algorithm of space carving is:

"**Step 1:** Initialize $\mathcal{V}$ to a volume containing the true scene.
Step 2: Repeat the following steps for voxels $v \in Surf(\mathcal{V})$ until a non-photo-consistent voxel is found:

a. Project v to all photographs in $Vis_{\mathcal{V}}(v)$. Let $col_1, ..., col_j$ be the pixel colors to which v projects and let $\varepsilon_1, ..., \varepsilon_j$ be the optical rays connecting v to the corresponding optical centers.
b. Determine the photo-consistency of v using $consist_K(col_1, ..., col_j, \varepsilon_1, ..., \varepsilon_j)$.

Step 3: If no non-photo-consistent voxel is found, set $\mathcal{V}^* = \mathcal{V}$ and terminate. Otherwise, set $\mathcal{V} = \mathcal{V} - v$ and repeat Step 2." [8]

Surf($\mathcal{V}$) is the surface of the volumn $\mathcal{V}$. $Vis_{\mathcal{V}}(v)$ is the set of photographs on which v on the surface of $\mathcal{V}$ can be seen. $consist_K()$ is a function which decides whether v is *photo-consistent*. K is the least number of photographs that the function should consider with and is less than the number of all the photographs. $\mathcal{V}^*$ is the carved result of $\mathcal{V}$. And the "optical centers" refers to the camera centers of the photographs.

The space carving algorithm needs to deal with the visibility of the voxels on different photographs and employs a $consist_K()$ function to decide the photo-consistency of the voxels.

2.1.5 Details about the growing iterative algorithm

As described in [5], an iterative way has been used utilizing the common visual hull of the images to increase the quality of the segmentation. It exploits GMM to construct the probabilistic models for the foreground and the background. Then the GMM parameters are used in an energy function. A volumetric graph cuts method is employed to get the visual hull through energy minimization at each iteration. As an extra, it has a fixation constraint that assumes the object is always in the center of the image. At first, the obtained visual hull is small. Then it will grow as the iteration continues to reach some convergence criterion,for example, the growing proportion of the viusal hull reaches a threshold..

Figure 2.5 shows the flowchart of the algorithm:

The initialization data of GMM comes from the fixation constraint. The pixels in a small circle which has the same center of the image are used to train the starting foreground GMM model. The other pixels are used to train the background GMM model. The GMM models are global because they sample the pixels from all the views.

Then the visual hull is obtained through an energy minimization method via graph cuts. The data structure of the visual hull is an array stores voxels. A voxel is usually a 3D point in the 3D space. The energy function in this method is given by Equation 2.9 [5]:

$$E(O, B, \{I_m\}, \Theta) = \lambda E_{vol}(O, B, \{I_m\}, \Theta) + (1 - \lambda)E_{surf}(O, B, \{I_m\}) \tag{2.9}$$

O is the set of voxels for the object. B is the set of voxels for the background. $O \cup B \subset V, V$ is the set of all the voxels.$\{I_m\}$ are the images, Θ is the parameters of the GMM models of both foreground and background.E_{vol} is the volume term that provides the preference for a voxel to be inside or outside the object. E_{surf} is the boundary term which helps to get a smoother boundary for the object. λ is a coefficient regarding the different requirements on the result.

If the increment of the visual hull at the end of one iteration is within a threshold, then the visual hull has converged, and the algorithm can stop. Otherwise, new silhouettes are extracted for each image from the projection of the currently obtained visual hull and then used to train new GMM models and obtain new visual hull afterwards.

Figure 2.6 shows the growing process of the obtained silhouette.

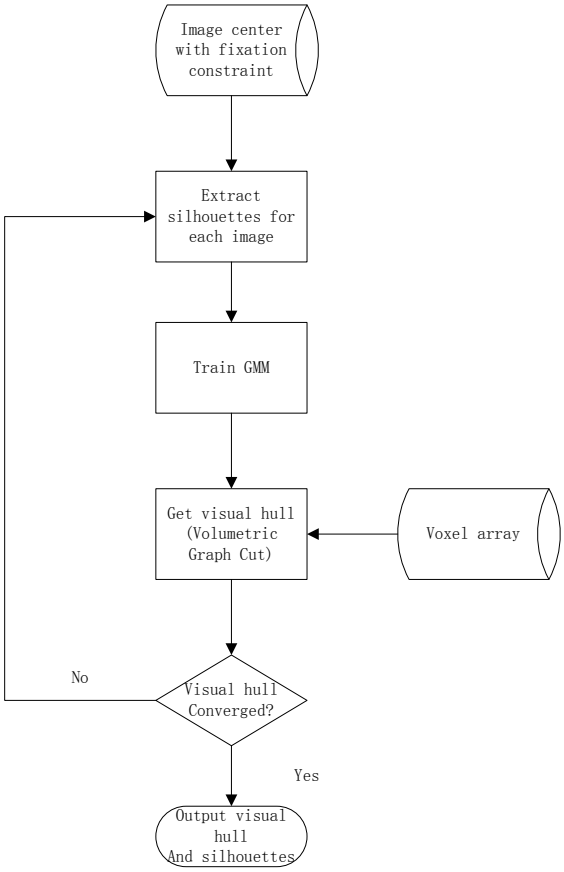


Figure 2.5: The flowchart of the growing iterative algorithm which utilizes GMM and graph cuts for silhouette segementation and visual hull reconstruction.

2.1.6 Details about the shrinking iterative algorithms

In contrast with the growing iterative algorithm, there is another kind of iterative algorithm: the shrinking iterative algorithm. For this kind of method, the starting silhouettes include most parts of the input images. Then the silhouettes will shrink according to some carving algorithm. The intersection of the input images is used to be the initial common visual hull. Through the projections of this visual hull, the initial silhouettes are obtained,see Figure 2.7. The "3D visibility volume" is the initial visual hull and the "Unknown region" is the initial silhouette.

One of the shrinking method described in [10] is quite similar to the method introduced in Section 2.1.5. They both exploit GMM to compute the probability and graph cuts method to obtain the information of the common visual hull. The difference lies between them is that the shrinking one does not really construct a visual hull.

The shrinking process is explained in Figure 2.8

Another shrinking method [27] does not utilize GMM or graph cuts method but uses a combination of the image segmentation method and two carving algorithms respectively for the silhouette

Figure 2.6: The growing process of the silhouette. [5]

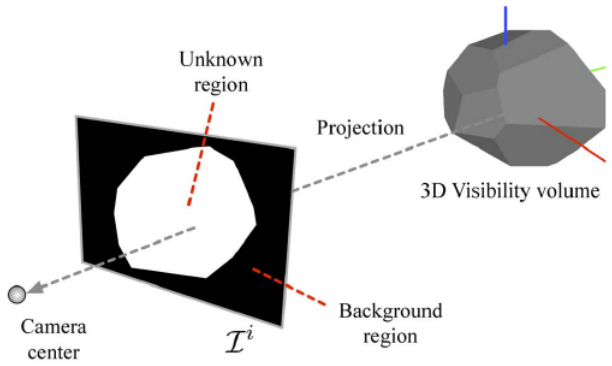


Figure 2.7: The initial visual hull and silhouette. [10]

and the visual hull. At first, the images are segmented into small regions like in Figure 2.9. Then the carving algorithms are used. Firstly, the visual hull is projected onto the image and an projection area is obtained. The image regions that are not fully inside the projection area are carved. After that the new silhouette on the image consists of the rest regions and it will be smaller. Secondly, the voxels of the visual hull whose projection are outside the new silhouette will be carved. This step for voxel carving is described as a "space-carving-like" method in [27]. It is based on the space carving method we discussed in Section 2.1.4, but without the consideration of the visibility of the voxel or the function to decide the photo-consistency, thus is simpler than the space carving method. These two steps will be performed to each image iteratively. The flowchart of the algorithm is given in Figure 2.10.

2.1.7 Comparison of different iterative algorithms

Three different iterative methods for silhouette segmentation have been presented above. In this section, a comparison about their advantages and drawbacks in several aspects have been performed in Table 2.1

Depending on the concrete situation and requirement, different combinations for these methods can be obtained to form new iterative algorithms. And our final choice of the combination will be described in Section 3.2.3 and 4.2.

	The growing algorithm [5]	**The shrinking algorithm 1 [10]**	**The shrinking algorithm 2 [27]**
Method for obtaining the initial data	Fixation in the center of the image	Intersection of the cones corresponding to the input images	Intersection of the cones correponding to the input images
Method for foreground and background classification	GMM	GMM	Image segmentation
Advantage	Accurate to each pixel	Accurate to each pixel	Only need to perform once
Drawback	Need retraining at each iteration	Need retraining at each iteration	Segments may be too big thus not accurate enough
Method for viusal hull reconstruction	Graph cuts	Graph cuts	Space-carving-like method
Advantage	Optimal result in practice	Optimal result in practice	Easy to implement
Drawback	Complicated to implement	Complicated to implement	Less optimal than graph cuts

Table 2.1: The comparison of the iterative algorithms for silhouette segmentation.

Figure 2.8: The shrinking process of the silhouette. [10]

Figure 2.9: The segmentation of the image. [27]

2.2 Integrating MATLAB code into C# project

Normally, MATLAB code can be used in C# project in three ways.

1. Using the *Automation* protocol which is based on Component Object Model (COM) [20]. COM is a framework for integrating reusable and binary code components into an application. COM can be written and used in any language that supports COM, like C#,C++,Visual Basic and MATLAB,etc. In *Automation* protocol, there is a client that is also called a controller and a server. Automation Client can control the automation server which is encapsulated as a COM. MATLAB engine is a library exposing interfaces to other software to control MATLAB and acts as a automation server. So other programs can access the MATLAB engine as a auomation server to execute MATLAB command as in the MATLAB environment. Using this method one can debug both in visual studio and MATLAB. But one needs to have an installed version of MATLAB, not just a MATLAB Compiler Runtime (MCR) which is the runtime engine for MATLAB.

2. Using a MATLAB builder for .NET [13]to build a .NET assembly. The assembly which is a Dynamic link library (DLL) provides IntelliSense, automatic data marshalling and garbage collection can thus be used directly as a native .NET component. MATLAB function can be called as a method in a class. This assembly can be deployed to machine that do not have MATLAB.

3. Using a MATLAB compiler to create a C shared library. Because the library is *unmanaged code*(cf. Section 2.7 in [7]), a wrapper is needed to provide access to the methods in the library and data exchange between *managed* and *unmanaged* code. This assembly can also be deployed to machine without MATLAB. But this method is fragile and complicated to program, thus is not a prior method.

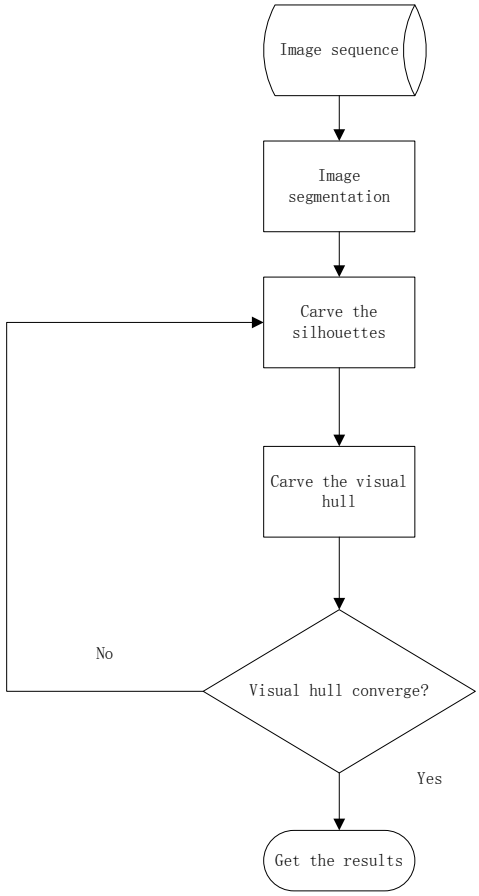


Figure 2.10: The flowchart of the shrinking algorithm which utilizes image segmentation and space-carving-like algorithm for silhouette segmentation and visual hull reconstruction.

2.3 Two coordinate systems

2.3.1 Pinhole camera and its coordinate system

Pinhole camera is a kind of camera without a lens and with a very small aperture, for example, the webcam on a laptop can be considered as a pinhole camera. In this 3D scanner system, every camera invovled will be treated as a pinhole camera.

A pinhole camera's coordinate system model is shown in Figure 2.11. In this figure, the pinhole camera can be seen as a point in the coordinate system and it is the origin O. The pinhole camera has two direction properties. One is the **looking direction** which is the positive direction of Z axis, and the other is the **up direction** which is the positive direction of Y axis. The looking direction is the direction that the pinhole camera is facing at, and the up direction points out which is the up side of the image projected onto the image plane of the pinhole camera. The image plane is where the image is projected by a pinhole camera and it is the plane I in the figure which is

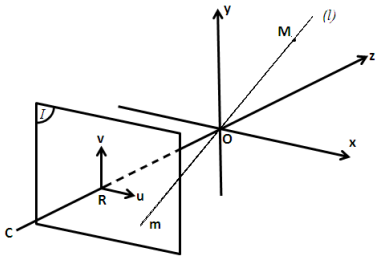


Figure 2.11: The pinhole camera's coordinate system model.

perpendicular to the Z axis. It has its own 2D coordnate system (R, u, v). As the pinhole camera does not have a lens, it does not have a real focus. Its focal length is actually decided by how long the distance is from the pinhole camera to its image plane. M is a 3D point in the figure, and its projection is m which is the intersection of a line l that crossed M and O and the image plane. As in the figure, m is an inverted projection of M.

In practice, a not inverted projection is expected for processing. To achieve this, the image plane can be assumed to be located in front of the pinhole camera, like in Figure 2.12. Now, the projection m is not inverted anymore. If the coordinate of M is $M(x, y, z)$ and the coordinate of m is $m(u, v)$, then the projection procedure can be formulated by Equation 2.10.

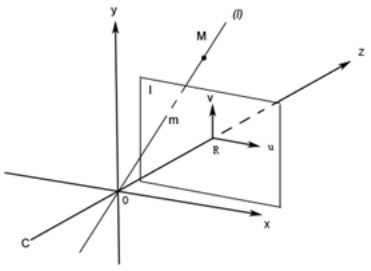


Figure 2.12: The assumed location of the image plane of a pinhole camera.

$$\begin{pmatrix} u \\ v \end{pmatrix} = \frac{f}{z} \begin{pmatrix} x \\ y \end{pmatrix} \tag{2.10}$$

Pinhole camera's projection is called perspective projection in which the further a object locates, the smaller it is, thus similar to human's eyes.

2.3.2 Aforge.net's coordinate system and the glyph's pose

Aforge.net [1] is the framework used in last semester's project [7] for video and image processing. When Aforge.net recognizes the glyphs(cf. Section 1.1.3) and estimates their positions in the 3D space relative to the camera, there is a 3D coordinate system involved - the Aforge.net's coordinate

system. This coordinate system is left-handed. The camera which is considered as a pinhole camera in Aforge.net is located at the origin of the coordinate system. So, this coordinate system is actually the same as the one described in Figure 2.11.

The glyph's pose includes its position and rotations relative to the camera. Aforge.net can estimate a glyph's pose from a picture including the glyph. At the beginning, before the estimation of the glyph's pose, the glyph's center is located at the origin of the Aforge.net's coordinate system and the glyph lies in the xOz plane with its edges parallel to x axis and z axis respectively, see in Figure 2.13. This is called the starting pose of a glyph.

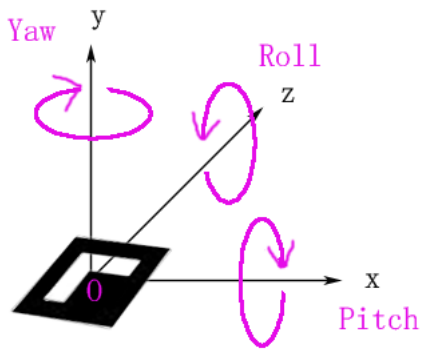


Figure 2.13: The starting pose of a glyph and Yaw-Pitch-Roll angles.

When the pose estimation is finished, the glyph may have performed some rotations around the axes and the glyph's center may have been located at another coordinate in the coordinate system. So the pose information actually includes two parts: several rotations around the axes and a translation for the glyph's center. The order for the two operations is fixed as rotations first and translation later. Reversed order will give a quite different result. The rotations are described by Yaw-Pitch-Roll angles. As shown in Figure 2.13, Yaw means the rotation around Y axis while Pitch corresponds to X axis and Roll corresponds to Z axis. The positive value of each angle means the clockwise direction when the corresponding axis is pointing at you, also see in Figure 2.13. When glyph is rotating, the axes are fixed in the space. Finally, the pose estimation is stored in a transformation matrix by Aforge.net.

Chapter 3

Solution Proposal

As stated in Section 1.2.2, this project inherits from last semester's work in terms of basic theory, main architecture, user-machine interaction and other programming constraints. In this chapter, more specific demonstrations will be given on these aspects of this project.

3.1 General constraints of the project

In Section 1.2.1 and Section 1.2.2, both constraints from the CIR software suite and the former work have been shown. Here a reminder which is also a summary is listed below.

- .NET/C# based software.
- Compatible with the WPF(Windows Presentation Foundation) framework.
- Complying with the MVC architecture when new made module is integrated into the current system.
- Free open source libraries which will not expand its open source property to the CIR's system.
- All computation occurs on a standalone computer.
- Visual hull theory based point cloud acquisition using just a single camera.
- The object should fit in a $15cm \times 15cm \times 15cm$ cube which is in the center of the glyphs on the tracker, also should be convex, not shiny or transparent with a simple shape.

3.2 Solution overview

The main achitecture of the system has been shown in Figure 1.2. This project still follows this architecture, but some changes are made to the MF1 module. Here the file formats of PCD and STL are recalled, then the changes to the acquisition module are illustrated.

3.2.1 The result files' format of different modules

In this 3D scanner system, a PCD file is used to be the intermedia file between the two modules. And a STL file is used to store the final 3D mesh result. A PCD file's format looks as below in Figure 3.1.

```
VERSION .7
FIELDS x y z rgb
SIZE 4 4 4 4
TYPE F F F F
WIDTH 213
HEIGHT 1
VIEWPOINT 0 0 0 1 0 0 0
POINTS 213
DATA ascii
0.81915 0.32 0 4.2108e+06
...
0.944 0.29474 0 4.2108e+06
```

Figure 3.1: Structure of a PCD file

The fields are explained below:

- **Version** defines the version of the file header; this field has no actual use in this project;
- **Field** defines the formatting of the data. In Figure 3.1, each point is stored as follows: x coordinate, y coordinate, z coordinate and color encoded in RGB. The color information is optional and can be replaced by normal information according to the application's need: for our purpose, as we consider binarized silhouettes, color information is useless;
- **Size** defines the size for each data, usually 4 bytes;
- **Type** defines the type for each data, usually float (F)
- **Width** indicates the total number of points in a row;
- **Height** indicates the dimension of the array containing the data; if it is not organized,its value is 1, and the Width field will indicate the total number of the points in the whole point cloud.
- **Viewpoint** field is used to merge several point clouds depending on their referential (not used for this project)
- **Points** is the total number of the points in the point cloud (if *Height* is 1, *Points* should be equal to *Width*);
- **Data** contains the points as defined by the parameters above, in an ASCII or binary representation.

```
solid vcg
  facet normal -9.102796e-001 3.934934e-001 -1.286626e-001
        outer loop
          vertex   -9.231440e-002 1.323640e-001 1.822220e-002
          vertex   -9.218020e-002 1.323480e-001 1.722380e-002
          vertex   -9.276740e-002 1.309920e-001 1.723110e-002
        endloop
  endfacet
...
endsolid vcg
```

Figure 3.2: STL file example

A STL file's example is shown in Figure 3.2.

The object is defined between keywords *solid* and *endsolid* and its name is behind *solid*. Then each facet or triangle is described between *facet* and *endfacet* keywords. Each facet is made of the facet normal computed by the right hand law and the three vertices of the triangle. Each vertex is described by its three-dimensional coordinates. All numbers are stored in floating point representation.

3.2.2 Change on the user-machine interaction

Some changes have been made to the tracker. The glyph kinds of the tracker have been increased to 4 to improve the camera calibration (cf. Section 4.1). A chessboard pattern has been added to the tracker to help the subtraction of the tracker itself and improving the shadow elimination. The new tracker is shown in Figure 4.1.

3.2.3 The modification of the acquisition module

As stated in Table 2.1, several combinations are available for an iterative algorithm which can perform background segmentation and visual hull obtaining simultaneously. Considering the project time limit, a space-carving-like method is more advisable for visual hull computation compared to the graph-cuts method. And GMM is more practical than the image segmentation method, because the latter may not be reliable enough according to our research and also needs more time in implementation. So the final idea of the combination of the iterative method is utilizing GMM and a space-carving-like method. Because the space-carving-like method will obviously make the visual hull to shrink, this algorithm belongs to the shrinking way. The initial visual hull is the intersection area of the cones corresponding to the frames.

This will make some difference in the internal structure of MF1, as illustrated in Figure 3.3.

Now SF11 still performs the tracker management task; but different to Figure 1.2, with each frame, not only the angle information will be stored, but also the entire pose estimation (cf. Section 4.1). SF22 accepts the frames and their pose estimations, and then uses the iterative algorithm to obtain the silhouettes and the visual hull simultaneously. At the end, a point list will be forwarded to SF23. SF23 converts the point list into a PCD file, and passes the file onto MF2.

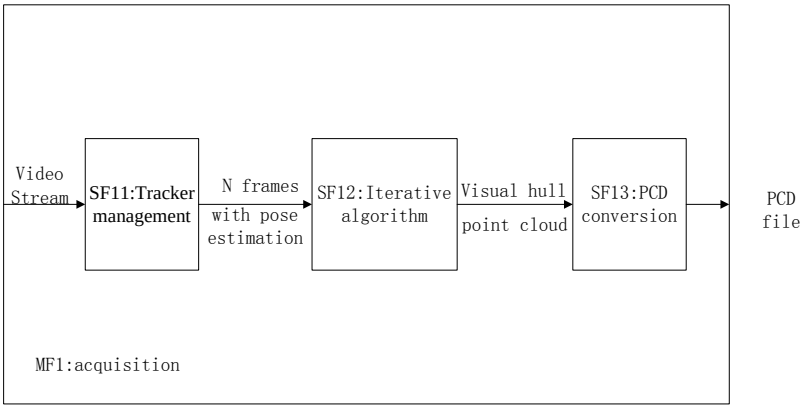


Figure 3.3: The new architecture of MF1.

Chapter 4

The re-development of the acquisition module MF1

In last semester's project [7], the acquisition module MF1 has only been partially implemented. In the end, it can extract key frames of the scanned object out of the input video stream and provide the rotation angle of the tracker in each frame compared to the first frame. In this project, the rest work for generating the point cloud has been accomplished, including SF12 and SF13 in Figure 3.3. Also some changes have been made to SF11, and the tracker.

4.1 The modification to SF11: Tracker Management

The old tracker used for last semester's project and the new tracker used for this project are both shown in Figure 4.1. There are two main changes from the old one to the new one: the glyph kinds have been increased from 1 to 4 and a chessboard pattern has been added. Some reasons have been given in Section 3.2.2. In terms of camera calibration, it means in this project to obtain the camera's extrinsic parameters [21] which denote the camera's position and orientation in the world coordinate system (cf. Section 4.2.1) and are needed in the space-carving-like algorithm (cf. Section 4.2.2). The camera's position and orientation are estimated from the glyph's pose estimation (translation and rotations). As the old tracker uses the same glyph, it is impossible to differentiate the different glyphs at different positions and thus impossible to place the cameras at the right position. So in the new tracker, there are four different glyphs at different positions which are given different names (cf. Section **??**). The tracker management submodule now records the camera's position and orientation of the key frames in the form of the glyph's pose estimation and also the glyph's name, then passes these information to SF12.

4.2 The implementation of SF12: Iterative Algorithm

As stated in Section 3.2.3, the combination of the GMM and the space-carving-like algorithm is chosen to implement an iterative algorithm to generate the point cloud. The flowchart of this algorithm is shown in Figure 4.2. At first, the initial silhouettes and the video frames are used to

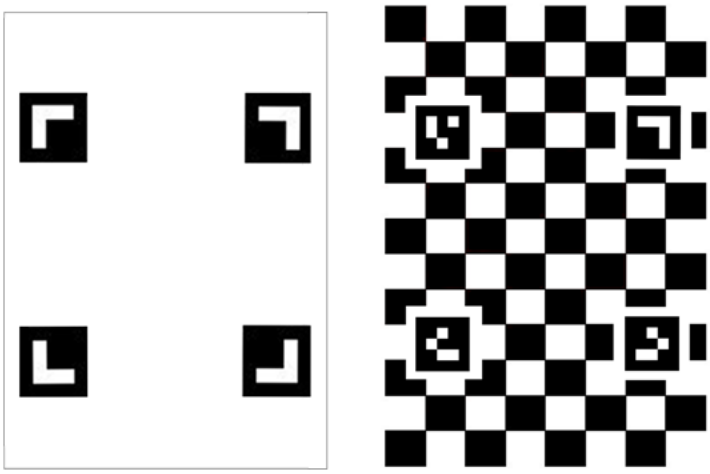


Figure 4.1: The old tracker (a) and the new tracker (b).

train GMM to obtain **GMM silhouettes**. Then GMM silhouettes are used by the space-carving-like algorithm to carve visual hull. The obtained visual hull projects to several virtual pinhole cameras corresponding to the frames to obtain **projection silhouettes**. Projection silhouettes are then used with video frames to update GMM to extract new GMM silhouettes. The iteration goes on like this to improve the accuracy of the point cloud gradually. The initial silhouettes are obtained in this way: four rectangle areas with a small width located along the four edges of the frame are defined as background and the rest part of the frame is defined as foreground. The "small width" is defined as one tenth of the width of the frame currently. As the whole algorithm includes several separate subfunctions, this section will introduce each subfunctions and their intermedia results first, then give the integration of the whole algorithm.

4.2.1 The implementation of the perspective projection

Section 2.3.1 has introduced the pinhole camera and the perspective projection. All the cameras mentioned below are pinhole cameras. The goal of the perspective projection function is to obtain the projection of an virtual 3D object on the image plane of a camera. Section 2.3.1 has just described how to get the projection in a camera's coordinate system, but didn't tell how to get the projection when the object and the camera are both at a random position and the camera has a random orientation in another coordinate system. To describe the process of calculating the projection of an object under this situation, a coordinate system is needed to contain the 3D object and the camera. See in Figure 4.3, there is a coordinate system which contains a 3D object near the origin composed of 3D points and 8 cameras around it with each camera rotated 45° around Y axis from the neighboring camera. The coordinate system is named as world coordinate system for description.

To get the projection of the object on these 8 cameras, those steps are need which can be seen in Figure 4.4.

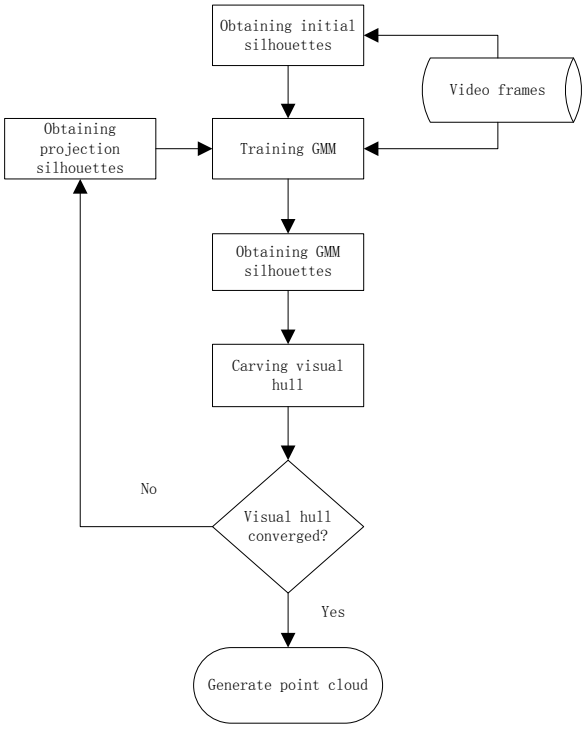


Figure 4.2: The flowchart of the iterative algorithm in this project.

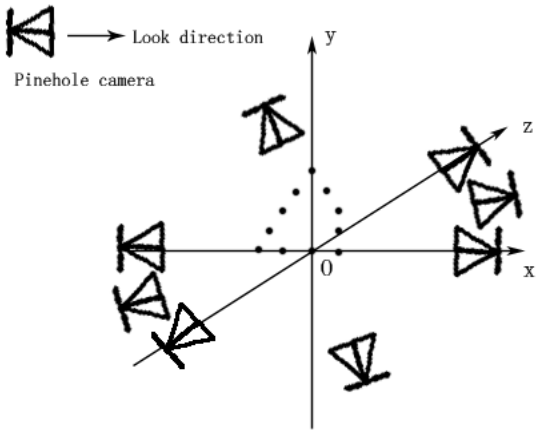


Figure 4.3: The world coordinate system contains a 3D object and several cameras.

Fisrtly, the world coordinate is transformed into the pinhole camera coordinate which is the coordinate in a pinhole camera's coordinate system like in Figure 2.11.

Secondly, the 3D pinhole camera coordinate is transformed into the 2D image plane coordi-

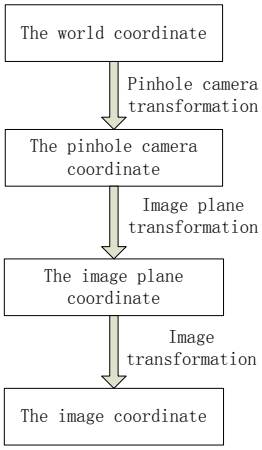


Figure 4.4: The transformations needed to transform the world coordinate into the image coordinate.

nate which is the coordinate on the image plane of a pinhole camera and this is done through Equation 2.10.

Thirdly, the image plane coordinate is transformed into the image coordinate. The image coordinate is used inside an image. Its origin is at the left-top corner of an image, its X axis points to the right side and its Y axis points to the bottom. If the image's width and height are presented as $imageWidth$ and $imageHeight$ and the image's center is at the origin of the image plane coordinate system, then the image plane coordinate of the left-top point of the image is $(-imageWidth/2, imageHeight/2)$. If the image plane coordinate of a random point in the image is (u, v), the random point's image coordinate (a, b) can be obtained by Equation 4.1.

$$\begin{aligned} a &= u + imageWidth/2 \\ b &= imageHeight/2 - v \end{aligned} \tag{4.1}$$

To perform those transformations, a data structure describing the propeties of the camera is introduced, see in Figure 4.5.

Camera
+Position : double[] +Orientation : double[] +FocalLength : double +BmpCamera : object +ImageWidth : int +ImageHeight : int

Figure 4.5: The data structure describing the camera's properties.

Position is the world coordinate of the camera.

Orientation is the pose of the camera in the world coordinate system. The concept of the pose of the camera is the same as the glyph mentioned in Section 2.3.2. It is also described by Yaw-Pitch-Roll angles. The starting pose of the camera is like this: its looking direction is along the positive Z axis and its up direction is along the positive Y axis.

FocalLength is the focal length of the camera.

BmpCamera is the Bitmap storing the projection image of the camera.

ImageWidth and **Image Height** are the width and height of the **BmpCamera**.

Position and **Orientation** are used in the pinhole camera transformation; **FocalLength** is used in the image plane transformation, it is the f value in Equation 2.10; **ImageWidth** and **ImageHeight** are used in the image transformation in Equation 4.1.

The pinhole camera transformation using **Position** and **Orientation** is performed like this: assuming the random point in the world coordinate system doesn't move, but the axes of the world coordinate system move. When the axes of the world coordinate system overlap with the axes of the pinhole camera coordinate system, the random point's world coordinate is then the same as its pinhole camera coordinate.The moving of the axes of the world coordinate system is separated into two steps:

1. Translating the oringin of the world coordinate system to the oringin of the pinhole camera coordinate system whose world coordinate is **Position**.

2. Rotating the axes of the world coordinate system according to the **Orientation** of the camera to make them overlap with the pinhole camera coordinate system's axes.

The pinhole camera transformation performed on a world coordinate is the inverse operation of each step above. The detailed mathimatical process of the pinhole camera transformation can be referred to [24]. Aforge.net has provided a *CreateFromYawPitchRoll* function which can generate a rotation matrix from the Yaw, Pitch, Roll angles, and the rotation transformation (step 2 above) can be obtained just by multiplying the coordinate and the rotation matrix.

The projection results of the 3D points in Figure 4.3 is shown in Figure 4.6

4.2.2 The implementation of the space-carving-like algorithm

In Section 2.1.6, the space-carving-like algorithm was mentioned. The space-carving-like algorithm in this project is quite similar with that one. The voxels are projected to the silhouette image. If the voxel is inside the foreground, it will be kept, otherwise it will be carved.

As told by the constraints of the project(cf. Section 3.1), the scanned object should be inside a $15cm \times 15cm \times 15cm$ cube decided by the glyphs on the tracker. So, the set of voxels to be carved just needs to be initialized to fill that cube and is called **voxel cube** below. The voxel cube is placed in the center of the world coordinate system with several cameras around it. The voxels are projected to the cameras and are carved or kept by the projection results. The position relationship of the voxel cube, the glyphs of the tracker and the cameras are shown in Figure 4.7.

The space-carving-like algorithm has 3 main functions: cube initializing function, space carving function and surface extraction function. They are stated in details below.

Cube initializing function

The prototype of the cube initializing function is:

void Initialize(int nNumber, int nInterval);

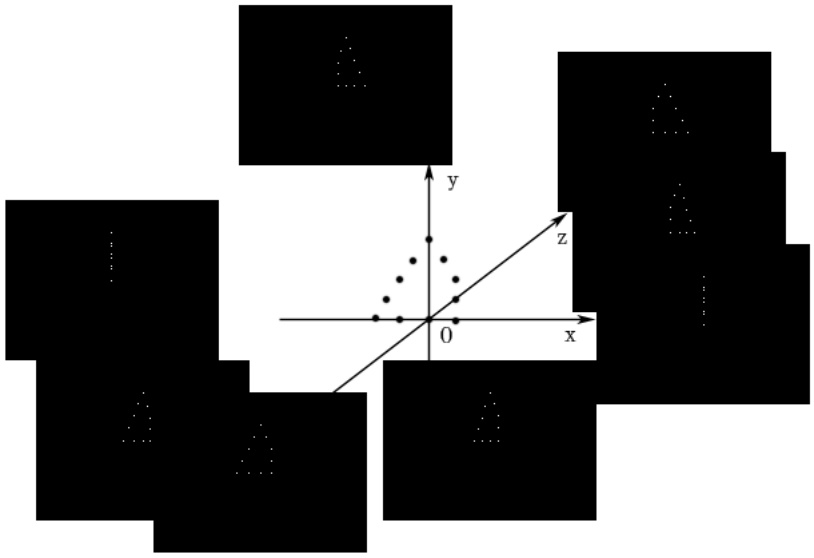


Figure 4.6: The perspective projection results of the 3D points.

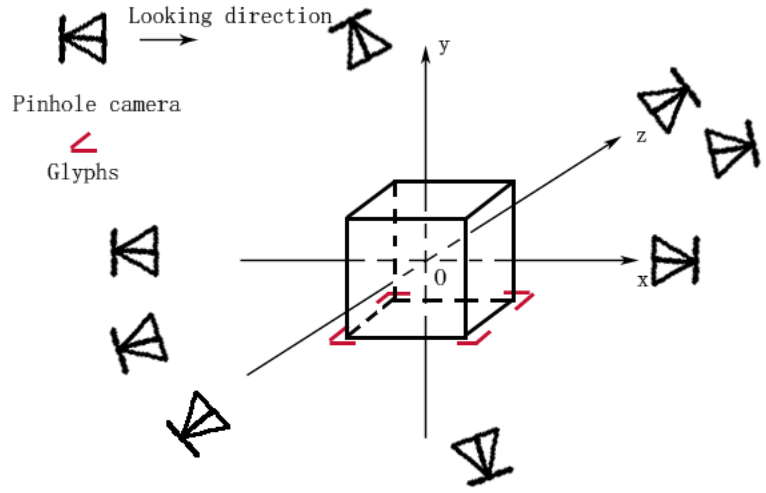


Figure 4.7: The voxel cube, the glyphs and the cameras in the world coordinate system.

Parameters: *nNumber* is the number of voxels of one dimension of the cube. *nInterval* is the interval of two voxels.

A voxel is presented as a 3D point in the function. So the voxel cube is the collection of the

coordinates of 3D points. The side length of the cube is $nNumber \times nInterval$. The side length of a voxel is defined as 1, so it is a unit voxel. In the space carving function, the side length of a voxel is adjustable, but it is smaller than *nInterval*. *nInterval* can change the size of the cube without change the number of the voxels. The computational complexity of the space carving function is $O(n^3)$, n is the voxel number of one dimension. Keep low voxel number is crucial to keep the computational complexity low.

The data structure of the voxel cube is a linked list. Because during the carving process, the number of the voxels will be decreased, using a linked list will benefit the efficiency of the whole iterative algorithm. This function stores the coordinates of voxels into the linked list.

Except for the linked list, this function also initialize a 3D array called **voxel state array**. The voxel state array stores the state of each voxel of being carved or not. 1 means not carved, 0 means carved. All the initial values of the elements in the array is 1 and this array is very useful for the surface extraction function.

Space carving function

The prototype of the space carving function is:

void CarvingBasedOnImage(List⟨Camera⟩ listCamera, int nVoxelSideLength);

Parameters: *listCamera* is an array storing the camera data structure (cf. Figure 4.5). *nVoxelSideLength* is the side length of the voxel's projection (defined as a square) on the image.

This function consists of a nested loop. The outter loop traverses the array of the cameras, extracting each camera's properties to process. The inner loop traverses all the voxels, projecting the voxel to the camera and deciding whether to carve it according to if the projecton's position is inside the background or not.

nVoxelSideLength is the side length of one voxel's projection on the image. Although the voxel is presented as a point here, it actually is a small cube and its projection on the image should be a quatrilateral. To be convenient, the projection is defined as a square. Calculating the proportion of the pixels inside the square projection which belongs to the background and using this proportion to decide whether to carve the voxel or not is more optimal. Meanwhile, this function updates the voxel stata array in real time.

Surface extraction function

The prototype of the surface extraction function is:

void ExtractSurface();

It doesn't have any parameter. It checks the voxel state array, and sorts out the points on the surface of the voxel cube. Then the surface points are output as the point cloud. This is because the reconstruction module just reconstructs the surface of the object, it only needs the points on the surface. The process of sorting out the surface points is kind of similar as the 3D printing process. Fisrt, all the points are scanned layer by layer along the Y axis. Then for each layer, the points are scanned by the straight lines parallel to X axis and Z axis. The points at both ends of the lines are surface points, see in Figure 4.8

Intermedia results of the space-carving-like algorithm

To test the space-carving-like algorithm, some synthetic images are used. The testing process is: in the world coordinate system described by Figure 4.7, around the voxel cube, placing a pair of

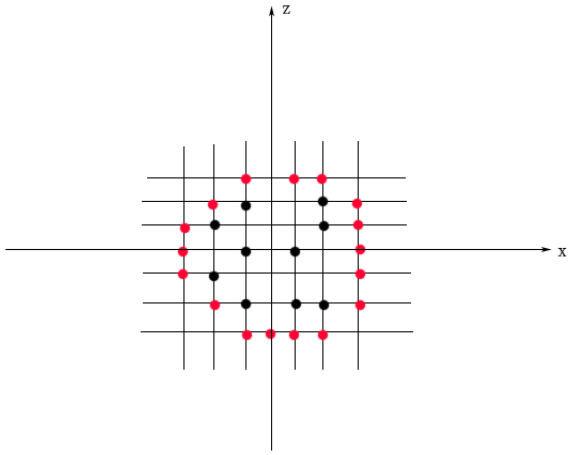


Figure 4.8: The way to sort out surface points (red points are surface points).

cameras simmetrically along x axis and z axis respectively, totally 4 cameras. For each pair of cameras, they are set the same synthetic images, totally 2 images. Then the space-carving-like algorithm is performed to generate the point cloud. The corresponding synthetic images and the generated point cloud are shown in Figure 4.9,4.10,4.11.

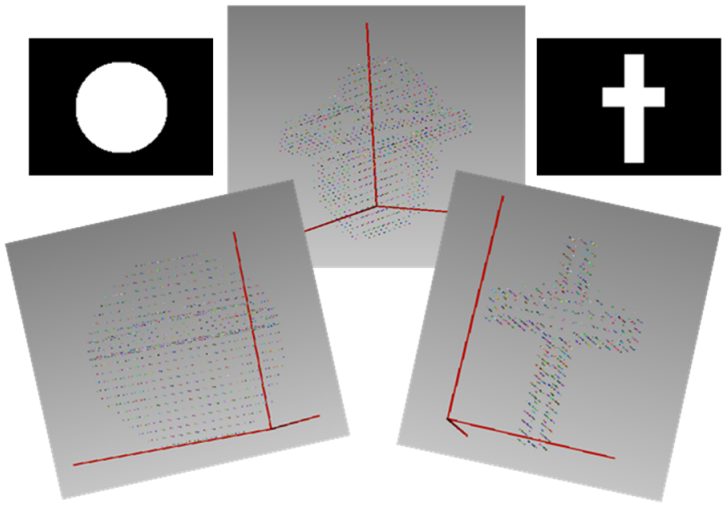


Figure 4.9: The intermedia result 1 of the space-carving-like algorithm.

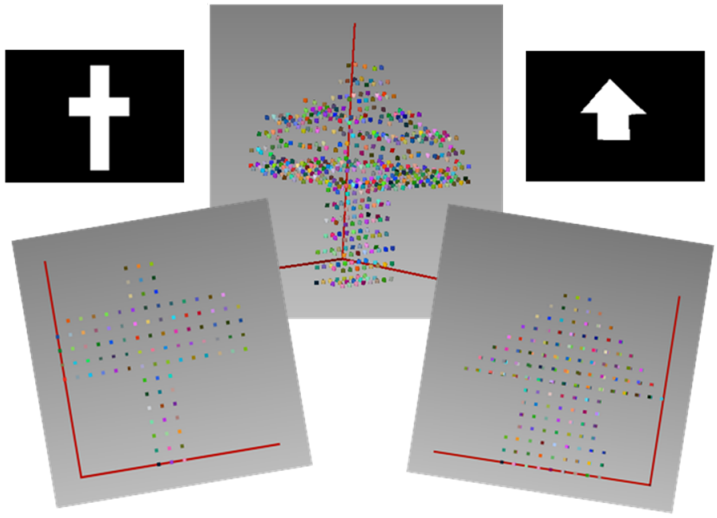


Figure 4.10: The intermedia result 2 of the space-carving-like algorithm.

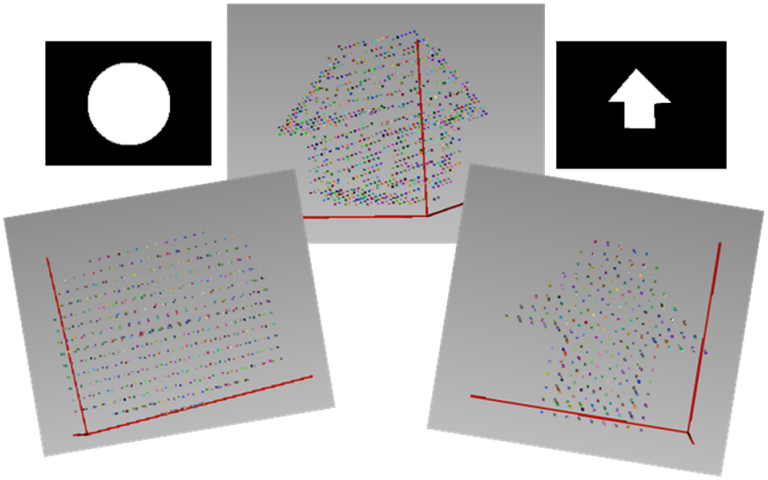


Figure 4.11: The intermedia result 3 of the space-carving-like algorithm.

4.2.3 The implementation of the GMM

The EM algorithm mentioned in Section 2.1.2 is used to obtain the parameters of GMM. In the development process of GMM, C# and MATLAB solutions are both tried. The MATLAB solution is settled as the final solution because of its convenience and high efficiency. The GMM silhouettes obtained by GMM is shown together with the integration results in Section 4.2.4. Here, the problems encountered in the implementation of the C# solution will be illustrated and discussed, then the MATLAB solution will be described.

Matrix multiplication efficiency of C#

To implement the matrix multiplication involved in the EM algorithm, Math.net [11] library is used. This library provides matrix data structure representations and matrix operation functions. But as C# is managed code, it cannot utilize SIMD (Single Instruction Multiple Data) [25] instructions to parallel the matrix multiplication. The matrix multiplication of Math.net which is totally based on C# code has very low efficiency, and thus cannot accomplish the computation needed by the EM algorithm within a short time. Additionally, the implementation of the EM algorithm needs to optimize the use of the memory and deal with singular matrix and algorithm convergence. After some efforts, the trial on C# implementation of the EM algorithm failed. After discovering the *gmdistribution* class in MATLAB which is used to manage GMM, the project adopted to the MATLAB solution.

Math.net also includes matrix multiplication implementations based on Atlas (Automatically Tuned Linear Algebra Software) [19] or Intel Math Kernel Library [22] which have better performance. [6] introduced how to optimize C# matrix multiplication using jagged arrays and Task Parallel Library [14], and it got astonishing benchmarks. As the time limit of the project, these methods have not been tried and thus can be considered as future work.

The GMM functions of MATLAB

MATLAB uses *gmdistribution* class to deal with the GMM. There are two main functions being used in this project. The first one is:

obj = gmdistribution.fit(...,param1,val1,param2,val2,...)

This function is used to train GMM. The parameters include the training set and the number of Gaussian distributions in GMM. There are also some parameters controling the detail ways of the calculation of the EM algorithm. Please refer to [26] to find out the full introduction of this function and its parameters. When using this function, there may be some problem of getting singular covariance matrix during the calculation and thus cannot get a result. In this project, this is sovled by setting a trivial positive value to the 'Regularize' parameter. [26] has given more solutions on this.

The output *obj* of the *fit* function is an object represents the GMM. On obtaining it, the next function is used to calculate the probability of one pixel belonging to this GMM.

y = pdf(obj,X);

By comparing which of the probabilities of one pixel belonging to the foreground GMM and the background GMM is bigger, the pixel can be classified into foreground or background.

Passing parameters between C# and MATLAB

MATLAB includes a MWArray.dll to provide several array data types for parameter passing. These array data types include:

MWArray, MWCellArray, MWCharArray, MWIndexArray, MWLogicalArray, MWNumericArray, MWStructArray.

MWNumericArray is used by C# to pass $N \times 3$ matrix (N is the number of pixels, 3 is the dimension of RGB color space) to MATLAB, then MWArray is used to receive the returned values from MATLAB. The data passed to MATLAB is stored in C#'s multidimensional arrays, then transformed into MWNumericArray data type. The returned values stored in MWArray data type are then transformed to corresponding C# data type.

The algorithm to obtain GMM silhouette images

As the foregrounds of all the frames represent the same object, a unified foreground GMM is used. As the backgrounds of all the frames are different to each other, for each frame, a unique background GMM is used.

The GMM silhouette image used to train GMM in the first iteration is obtained like this: On each edge of the frame, a rectangle area with the width which is the one tenth of the frame's width and with the height which is the same as the length of the edge is defined as background, the rest part of the frame is foreground, see in Figure 4.12. The background defined here is considered as default background and treated as background in each iteration. The foreground defined here is considered as default foreground and treated as foreground in each iteration. In every iteration, the silhouette is extracted only from the default foreground.

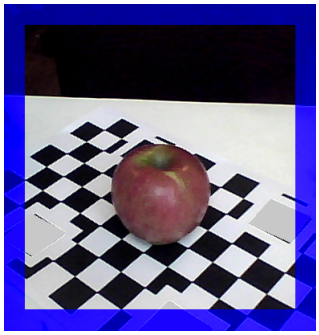


Figure 4.12: The default background of each frame (The blue area).

The pseudo-code of the algorithm that obtains the GMM silhouette images is below:

```
BEGIN obtaining GMM silhouette images
    N is the number of the frames;
    M is the iteration index;
    //M starts from 1
    IF M != 1
        ArrayProjectionSilhouette[N] loads the projection silhouette images from last iteration;
    ENDIF

    ArrayFrames[N];
    //An array stores the bitmaps of the N frames.

    ArrayBackground[N,X,3];
    //A 3 dimensional array stores the RGB information of the pixels of the background of each frame.
```

```
    //X is the background pixel number in each frame which is not fixed.

    ArrayAllForeground[Y,3];
    //A 2 dimensional array stores the RGB information of the pixels of the foregrounds of all frames.
    //Y is the foreground pixel number of all frames.

    FOR each frame i = 0: N-1
        IF M == 1
            Add the RGB values of the default foreground pixels in ArrayFrames[i]
            to ArrayAllForeground[ , ];

            Add the RGB values of the default background pixels in ArrayFrames[i]
            to ArrayBackground[ i, , ];
        ELSE
            Add the RGB values of the pixels in ArrayFrames[i] corresponding to
            ArrayProjectionSilhouette[i]'s foreground to ArrayAllForeground[ , ];

            Add the RGB values of the pixels in ArrayFrames[i] corresponding to
            ArrayProjectionSilhouette[i]'s background to ArrayBackground[ i, , ];
        ENDIF
    ENDFOR

    Train a foreground GMM using ArrayAllForeground[Y,3];
    Train N background GMM using ArrayBackground[i,X,3] respectively;

    FOR each frame i = 0: N-1
        FOR each pixel
            Calculate the probability Pf using the foreground GMM;
            Calculate the probability Pb using the background GMM i;
            IF Pf ≥ Pb
                Assign this pixel to foreground (Set its color as white);
            ELSE
                Assign this pixel to background (Set its color as black);
            ENDIF
        ENDFOR
        Generate GMM silhouette image i;
    ENDFOR
END
```

To accelarate the EM algorithm, several pixels are combined together to decrease the pixel numbers of the training set. $N \times N$ pixels are merged as one pixel, the mean values of their RGB values are used as the RGB values of the merged pixel. The more pixels are merged into one pixel, the quicker the training process is and the lower the accuracy of the result is. There must be a trade-off between the time performance and the accuracy of the image. After some experiments, N is fixed as 3 in the project.

4.2.4 The integration of the whole iterative algorithm

After integrating the perspective projection algorithm, the space-carving-like algorithm and the GMM algorithm like in Figure 4.2, the whole iterative algorithm will be obtained. In order to integrate perfectly with the pose estimation results of Aforge.net, some modifications are made to the perspective projection algorithm.

The integration of the perspective projection with Aforge.net

The perspective projection algorithm needs the camera information described by the camera data structure shown in Figure 4.5. The *Position* and *Orientation* data are estimated from the glyph's pose estimation stored with the key frames. But after initial experiments of using *Position* and *Orientation* values in the former perspective projection algorithm, no correct result is obtained. The possible explanation for this is: The glyph's pose estimation is obtained in Aforge.net's coordinate system; the estimated *Position* and *Orientation* are then used in the world coordinate system; the following pinhole camera transformation brings the coordinates back to Aforge.net's coordinate system (a pinhole camera coordinate system) again. The pose estimation contains only approximate values, so it has some deviations. The above process involves transformations between two coordinate systems, so deviations continue to accumulate and the result cannot be correct. As the cube can be presented along with a glyph in Aforge.net's coordinate system, the perspective projection algorithm can be modified to make the pose estimation and the pinhole camera transformation both happen in Aforge.net's coordinate system, and thus can reduce the deviations and obtain accurate result.

Before describing the new perspective projection algorithm, some introductions are given on the glyph's coordinate system and the position relationships between the voxel cube and the four glyphs. In Figure 4.1 (b), there are four different glyphs. According to the glyhps' position in the figure, from left to right, from top to bottom, they are named as "LeftTop", "RightTop", "LeftBottom" and "Right bottom". If the four glyhps are placed in the positions shown in Figure 4.7, their position relationships with the voxel cube look like in Figure 4.13.

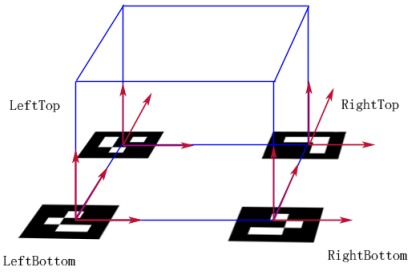


Figure 4.13: The glyphs' position relationships with the virtual cube.

In Figure 4.7, keeping the axes' direction fixed and making the glyphs' center as oringins, four glyphs' coordinate systems can be established respectively as in Figure 4.13. In Figure 4.13, each

glyph is at its starting pose (cf. Section 2.3.2) in itself's coordinate system and each glyph's coordinate system is the same as the Aforge.net's coordinate system which is a pinhole camera coordinate system. The voxel cube has different positions in different glyph's coordinate system in Figure 4.13 and these positions are called the starting positions of the voxel cube, so the corresponding coordinates of the voxels are called *starting pinhole camera coordinates*. The project defines that the voxel cube's position relationship with a specific glyph in that glyph's coordinate system is fixed, so when some pose transformation is performed to the glyph, it is also performed to the voxel cube, making the glyph and the voxel cube still together in this glyph's coordinate system. Because the pose transformation is estimated by Aforge.net, the voxel's coordinates after transformation from the *starting pinhole camera coordinates* in the glyph's coordinate system are called *estimated pinhole camera coordinates*. Both *starting pinhole camera coordinates* and *estimated pinhole camera coordinates* are concepts used in the new pinhole camera transformation.

To simplify the processing of frames, part of the frame decided by the virtual cube is extracted as the final frame. The extracted area is still a rectangle; the edges of the rectangle are parallel to the edges of the original frame; the positions of the top edge, the bottom edge, the left edge and the right edge of the rectangle are decided by the toppest, bottommost, leftest and rightest vertices of the virtual cube on the frame.

Now in SF11, the information stored with the key frame should also include the following contents:

1. The name of the glyph that is on the bottommost position of the frame (Because the virtual cube is drawn based on its estimation, cf. Section 4.2.3 in [7]).
2. This glyph's transformation matrix (cf. Section 2.3.2).
3. The coordinate of the left-top corner of the extracted frame in the original frame's coordinate system (an image coordinate system).

To store these information, 4 data members are added to the data structure of the camera, see in Figure 4.14.

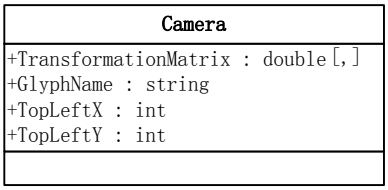


Figure 4.14: The added data member for the camera's data structure.

TransformationMatrix stores the transformation matrix of the bottommost glyph. **GlyphName** is this glyph's name. **TopLeftX** and **TopLeftY** are the coordinates of the left-top corner of the extracted frame in the original frame's coordinate system.

Then the voxel cube is introduced into the glyph's coordinate system, and the new perspective projection algorithm is below:

BEGIN the perspective projection algorithm based on Aforge.net

$P(a, b, c)$ is the world coordinates of a voxel;

//The new pinhole camera transformation.
SWITCH GlyphName
    Obtain the starting pinhole camera coordinates $Q(x, y, z)$ by translating
    the world coordinates $P(a, b, c)$ to the glyph's coordinate system corresponding
    to the GlyphName;
ENDSWITCH
//Obtain the estimated pinhole camera coordinates.
$Q'(x', y', z') = TransformationMatrix * Q(x, y, z)$;

//Obtain the image plane coordinates
$S(u, v) = f/z' * (x', y')$;

//Obtain the image coordinates
$u' = ImageWidth/2 + u$;
$v' = ImageHeight/2 - v$;

//Transform to the coordinates in the extracted frame's coordinate system (also an image coordinate system)
$m = u' - TopLeftX$;
$n = v' - TopLeftY$;
END

Figure 4.15-4.18 have given the results of the iterative algorithm after processing on an apple, a pear, a bunny and a bear can video frame sequences. The figures have listed the frame sequences, 3 iterations results of GMM sihouettes and projection silhouettes. The 3 iterations are the first iteration, the middle iteration and the last iteration. The subtraction process of the background can be seen in the figures and also how the GMM silhouettes and the projection silhouettes are affecting each other. But it also can be seen that in Figure 4.18, part of the silhouette of the bear can has been mistakenly carved. The reason is that, the principle of GMM to segment the foreground and the background is to calculate and compare the probabilities of one pixel belonging to the foreground and the background, and it doesn't consider the region information. The tracker is made of black and white colors, so black and white colors are classified as background colors. The wrongly carved parts are most pixels with black or white colors. This is the defect of the algorithm, so the algorithm can only be used well on objects which have very different colors with the tracker and other background. The discussion on the improvement of the algorithm will be stated in Chapter 6.

The convergence criterion of the iterative algorithm

The convergence of this algorithm is decided by the proportion of the carved voxels in the whole voxels before this iteration's carving. Figure 4.19 has shown the carved voxel proportions of 19 iterations after the iterative algorithm processing on an apple, a pear and a bunny frame sequences (To see more clearly on the lower parts of the curves, only 5-19 iterations are choosed).

In Figure 4.19, all the curves become stable near the cyan line which represents 0.3%.

Then the projection silhouette obtained when the carved proportion is less than 0.3% for the first time is compared to the silhouettes of its previous iteration and its following iteration in Figure 4.20. There are no obvious differences between them. So the convergence criterion of the

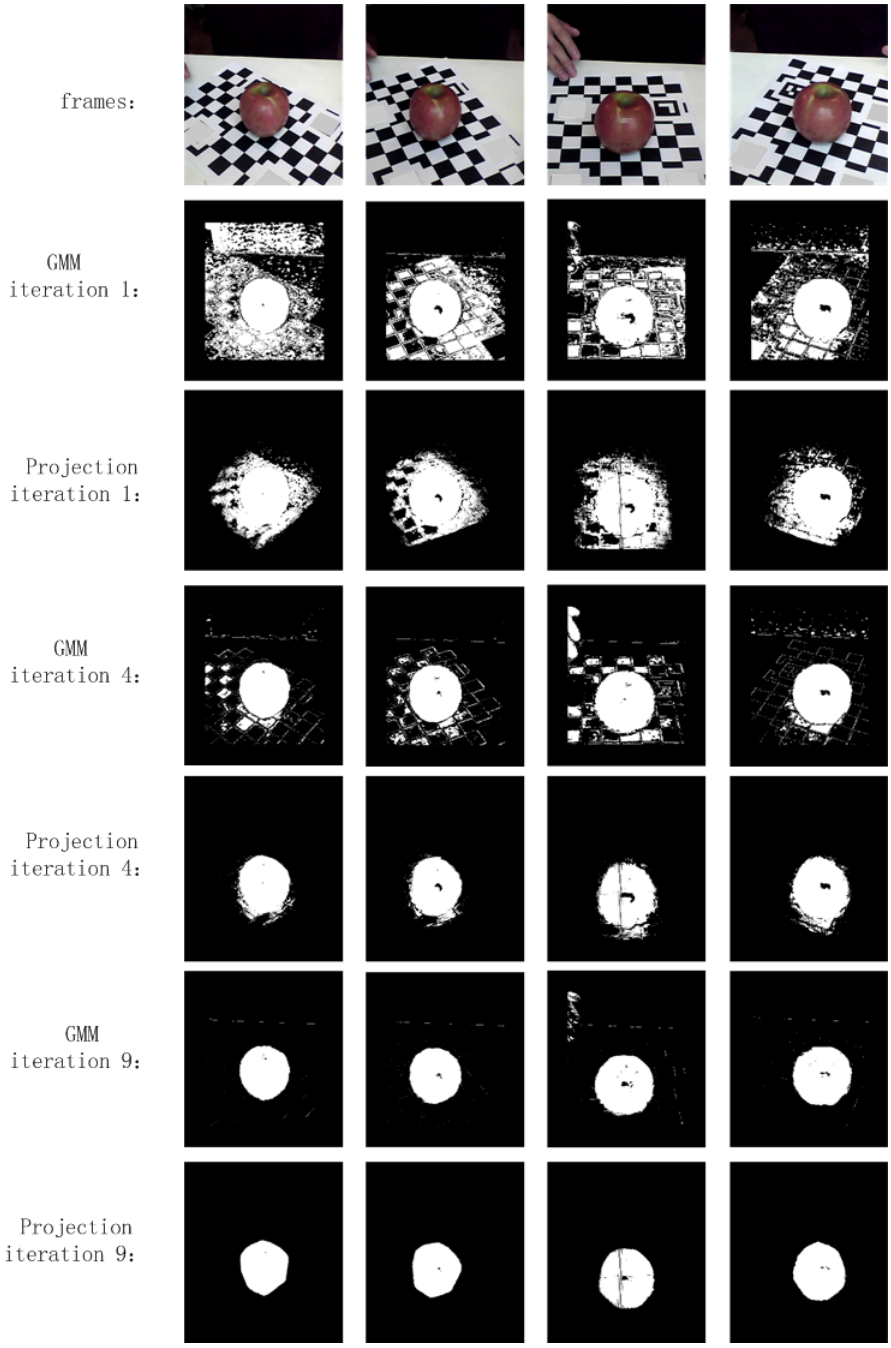


Figure 4.15: The apple results of the iterative algorithm.

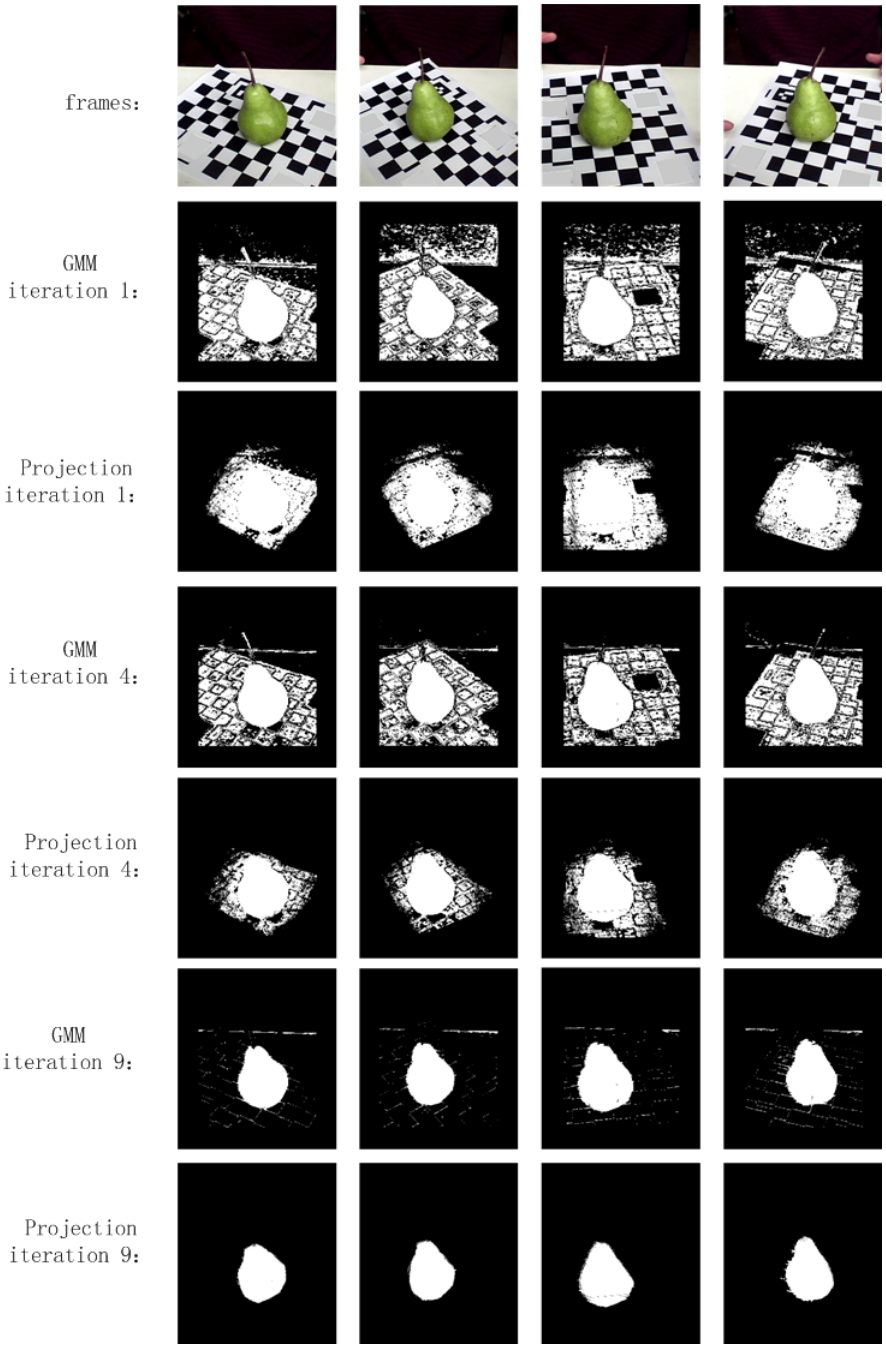


Figure 4.16: The pear results of the iterative algorithm.

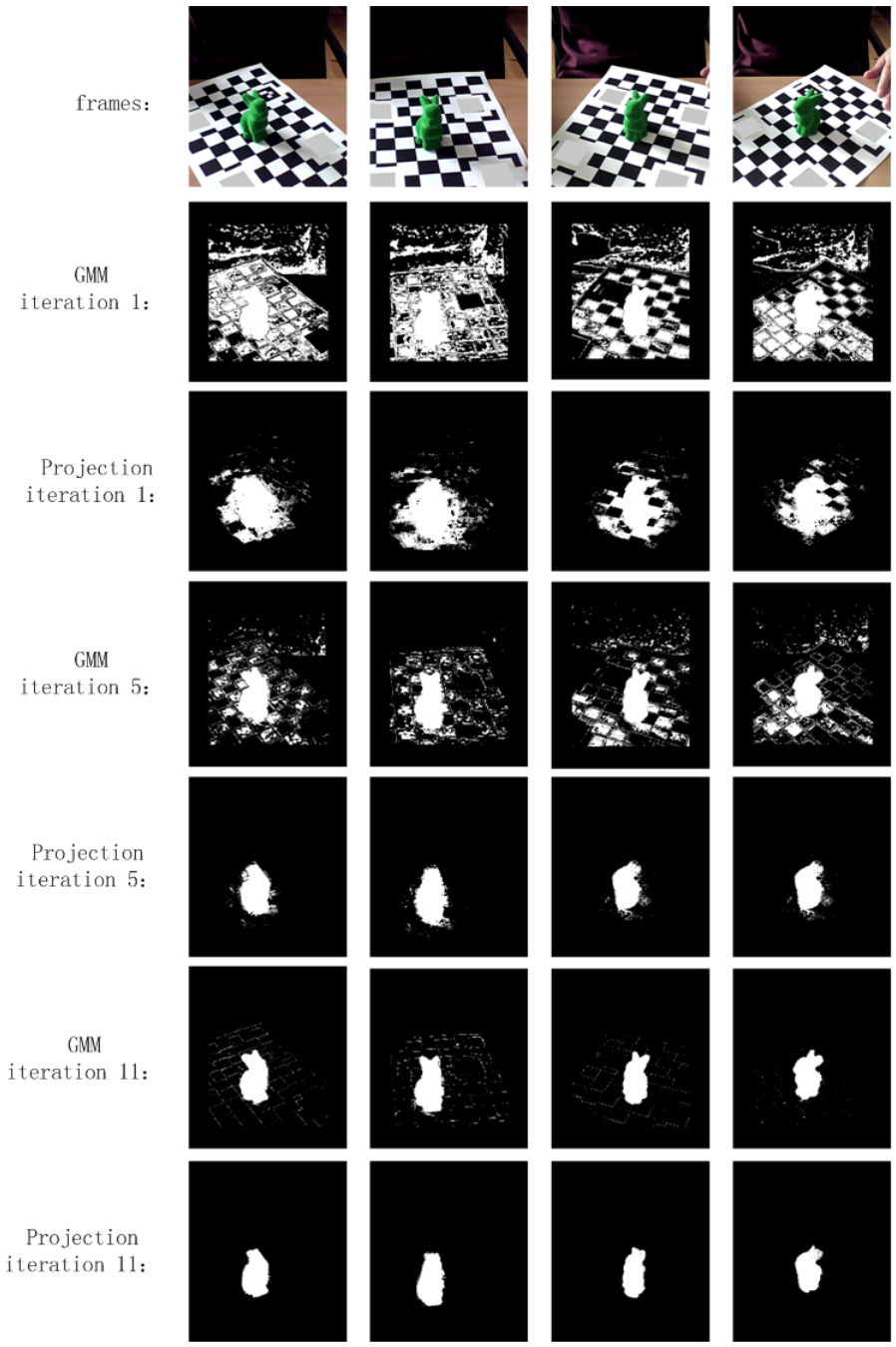


Figure 4.17: The bunny results of the iterative algorithm.

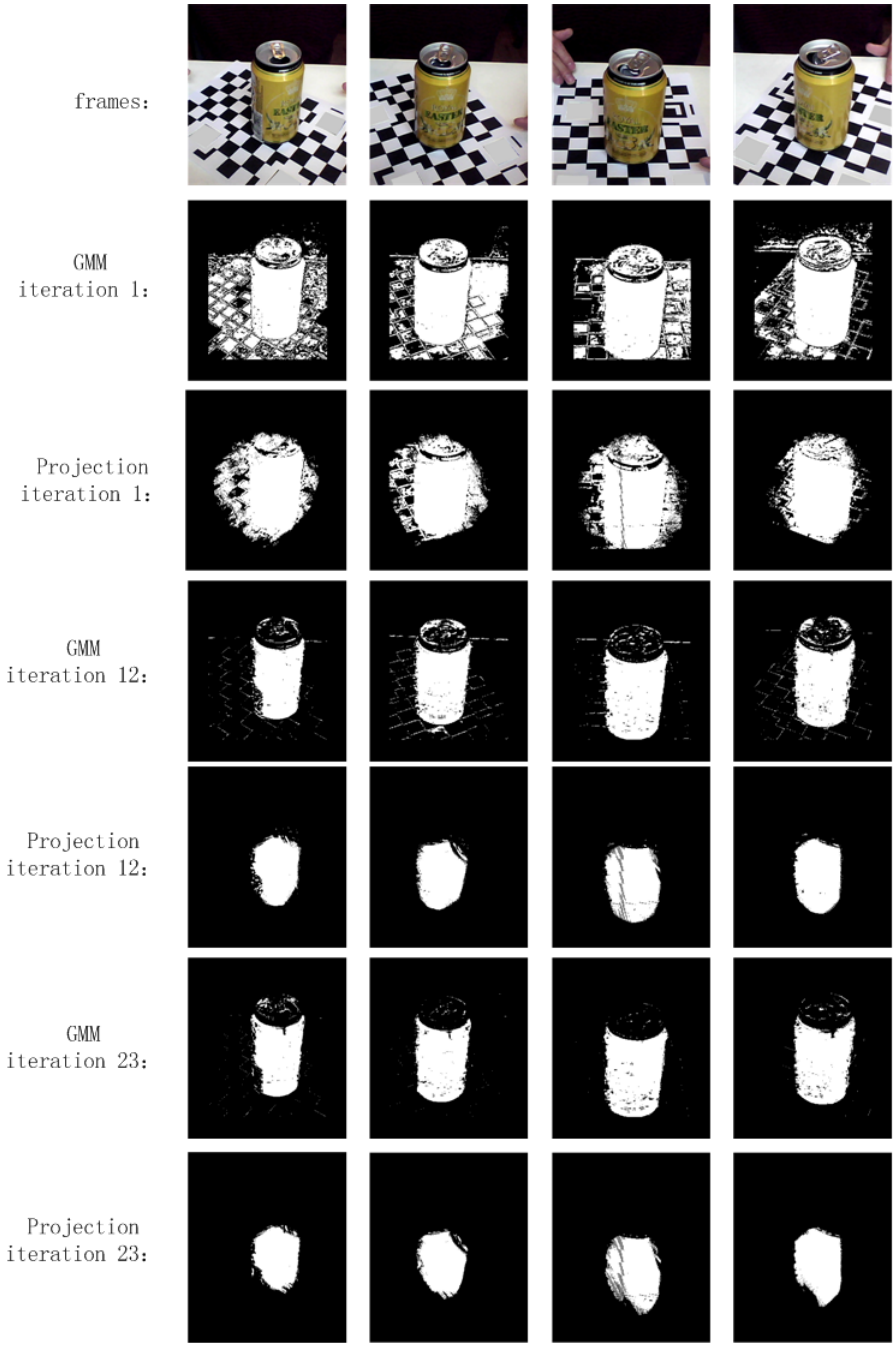


Figure 4.18: The bear can results of the iterative algorithm.

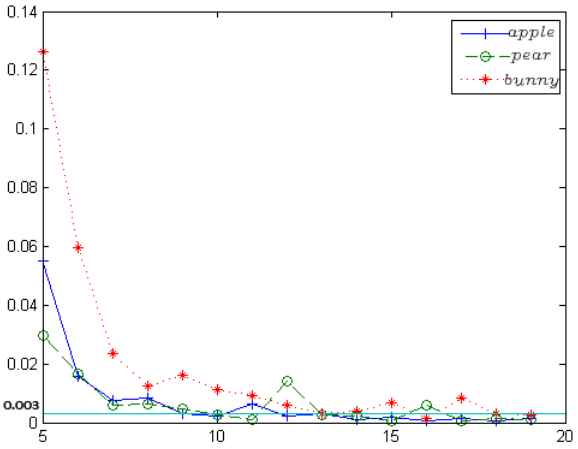


Figure 4.19: The carved proportion in every iteration.

iterative algorithm is that the proportion of the carved voxels in the whole voxels before carving is less than 0.3% in this iteration. In practice, the number of 0.3% can be enlarged a little to increase the time performance of the system.

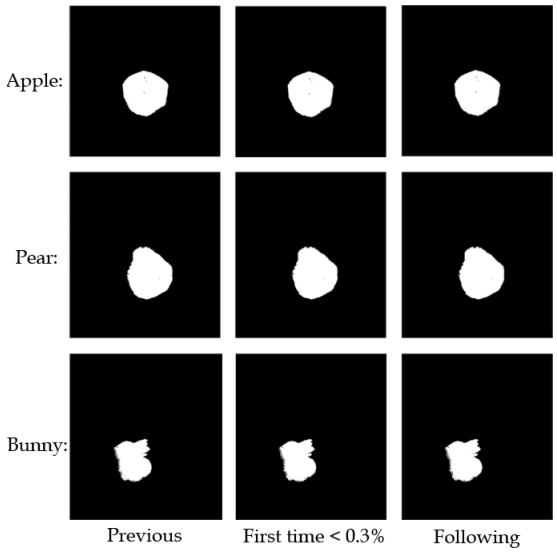


Figure 4.20: The comparsions of the silhouette obtained when the carved voxel proportion is less than 0.3% for the first time to the silhouettes of the previous and following iterations.

4.3 The implementation of SF13: PCD File Conversion

Section 3.2.1 has introduced the format of the PCD file. The point cloud generated by SF12 is a linked list containing the 3D coordinates of all the points in the point cloud. The **FIELD** domain of the PCD file used in this project only stores the coordinates of x, y, z without any other information. **WIDTH** stores the number of all the points. **HEIGHT** is 1. **VIEWPOINT** is not used and the **DATA** is in ASCII format.

Additional, the coordinates in PCD file are normalized values which are between $(-1, +1)$. The coordinate which has the biggest absolute value is extracted out and the biggest absolute value is described as A, then every coordinate is divided by $A + 1$ to obtain the normalized value.

Chapter 5

The demonstration of the 3D scanner system and the time performance analysis

After the accomplishment of the acquisition module MF1, the 3D scanner system can be finally integrated to perform the complete 3D scanning process which is demonstrated in this chapter. After the demonstration, several scanned results are shown in the sequence of video frame, GMM silhouette, point cloud and 3D mesh model. In the end, the testing results of the time performance of the 3D scanner are placed.

5.1 The demonstration of the 3D scanner system

Section 1.1.3 has given the brief introduction of how to use the 3D scanner system. Here in Figure 5.1, is a real example of using the 3D scanner. An apple is placed on the tracker in front of a laptop which has a webcam on top of the screen. The webcam's looking direction has an about 45° angle with the surface of the tracker. The environment is well illuminated.

The software environment of CIR's software suit is shown in Figure 5.2. This environment has the overall functions controling the 3D printer. The third button from right to left on the top is the switch of the webcam which is also the switch of 3D scanner. The 3D scanner function can be launched by switch it on.

When the webcam is on, adjust the webcam's position and orientation and the tracker's position, making the four glyphs of the tracker all inside the visible area of the UI (user interface). Then the tracker will be recognized by the 3D scanner. After recognizing the tracker, the virtual cube is drawn. When the virtual cube is shown on the screen, keep adjusting the webcam's position and orientation, making the virtual cube completely inside the visible area of the UI , like in Figure 5.3.

Then rotate the tracker with hands slowly and smoothly. During the rotating process, keep the virtual cube completely inside the visible area of the UI. If the virtual cube disappears in the rotating process, it means that either the rotating speed is too fast or some glyphs are not inside the visible area. If this happens, the rotating should be slowed down and check whether there is some glyph outside the visible area. When the virtual cube shows up again, the rotating can be

Figure 5.1: The using of the 3D scanner in reality.

Figure 5.2: The graphic user interface of CIR's software suit.

continued. After acquiring enough key frames, there will be some notice under the buttons on the UI, like in Figure 5.4. Then the user can switch off the camera by clicking the same button again and the point cloud generating starts.

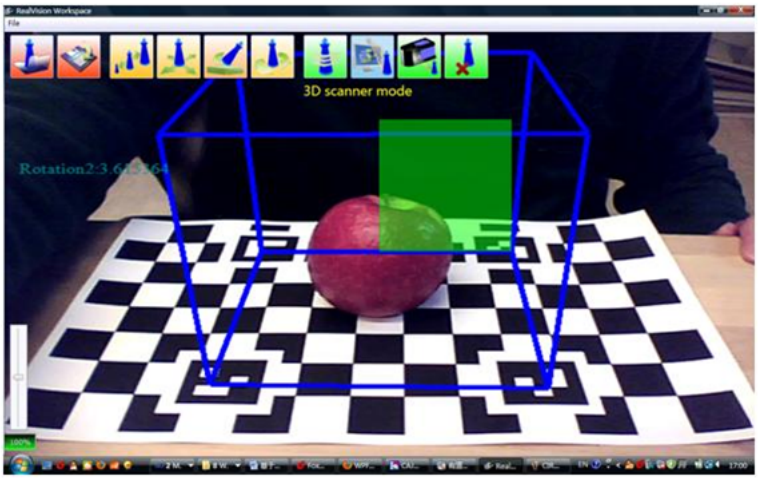


Figure 5.3: The recognition of the tracker and the displaying of the virtual cube.

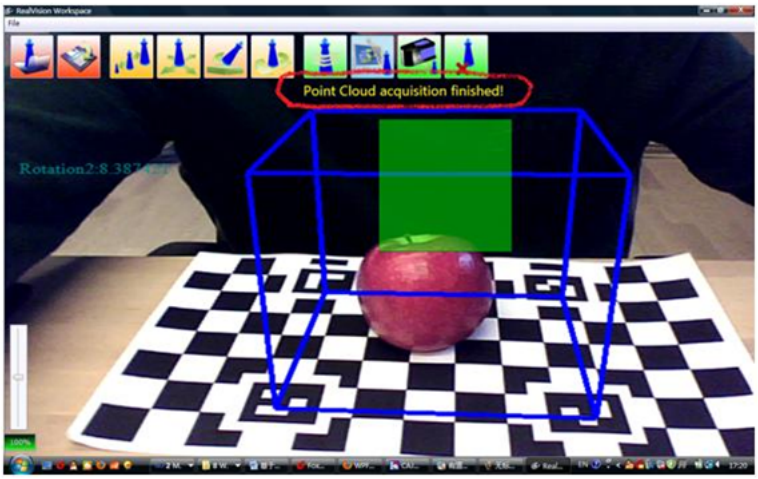


Figure 5.4: The completion notice of the frame acqusition.

During the iterative generating process, the notice area on the UI keeps informing the user about the current iteration index. After finishing the point cloud generating, the system writes the point cloud information into a PCD file and opens a popup window. The popup window which is the 3D reconstruction window loads the PCD file and displays the point cloud on the UI, like in Figure 5.5.

The 3D reconstruction window contains the functions of denoising, normal computation and greedy projection (cf. Chapter 5 in [7]) which are shown in Figure 5.6, 5.7 and 5.8.

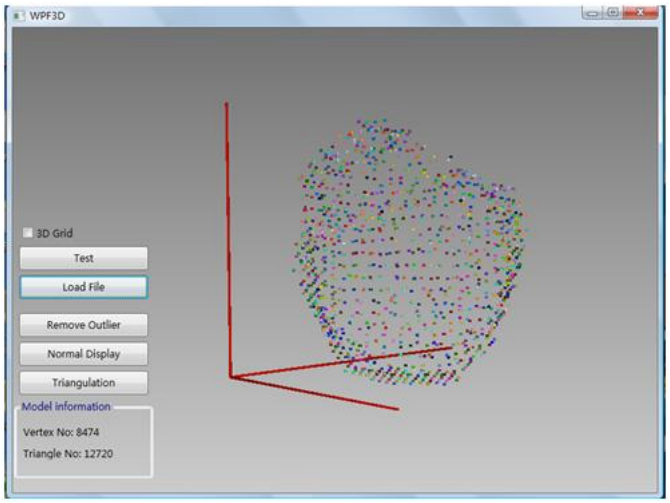


Figure 5.5: The 3D reconstruction window and the point cloud.

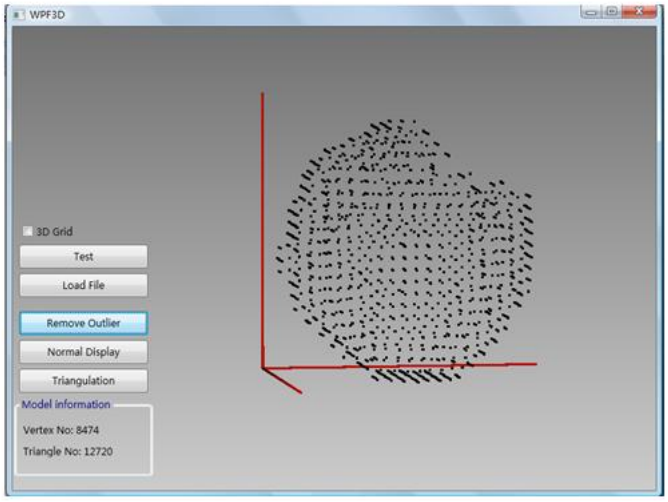


Figure 5.6: The denosing function.

5.2 The result sequences of the 3D scanner system

This section demonstrates the test results after integrating the two modules into the complete system. In order to illustrate the concrete process of how video frames turn into 3D mesh models, the testing results consist of the video frame, the GMM silhouette, the point cloud and the 3d mesh model.

Figure 5.9 is the results of an apple, Figure 5.10 is for a pear and Figure 5.11 is for a bunny.

In these figures, the point cloud and the 3d mesh model have both recovered the basic shape of

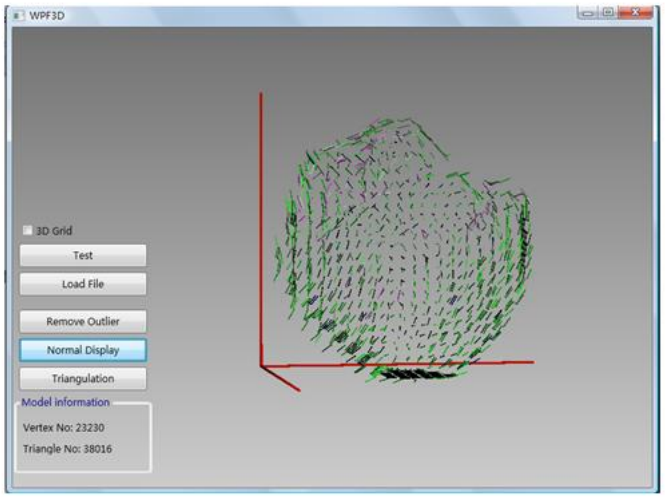


Figure 5.7: The normal computation function.

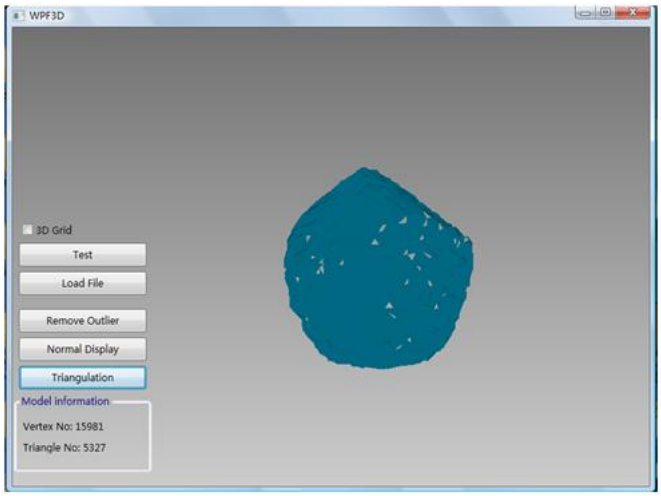


Figure 5.8: The greedy projection function.

the object, but the 3D mesh model still has some holes on it. This is because, currently there is not a function to find out and fix these holes and 3D reconstruction module needs further improvement.

5.3 The time performance of the 3D scanner system

Table 5.1 has offered the time consuming of the 3 tests above and as well the parameters setting. The testing of the time consuming of the 3D reconstruction module is performed by executing the

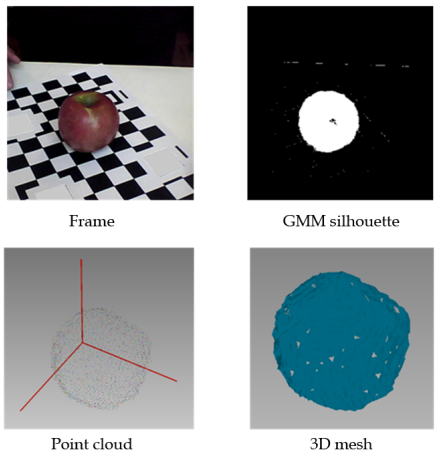


Figure 5.9: The result sequence of an apple.

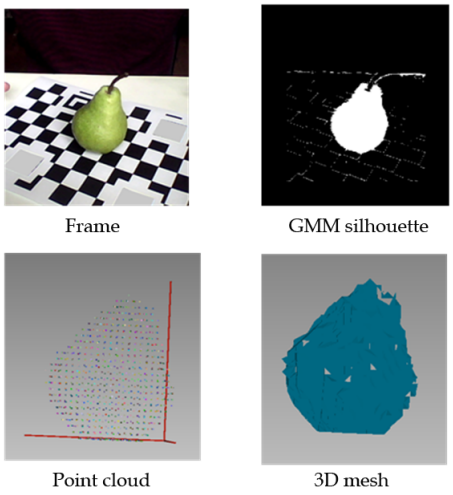


Figure 5.10: The result sequence of a pear.

denoising, the normal computation and the greedy projection functions continuously by program. The CPU of the computer is: Intel®Core(TM)2 Duo CPU T6400 @2.00GHz 2.00GHz, memory size is 2GB and the operating system is 32-bit Windows Vista system.

From the table, it can be known that the time consuming is not too much and can meet the practical requirement. By ajusting the voxel number, the frame number and the convergence threshold, better time performance can be obtained at the cost of a little lower result quality.

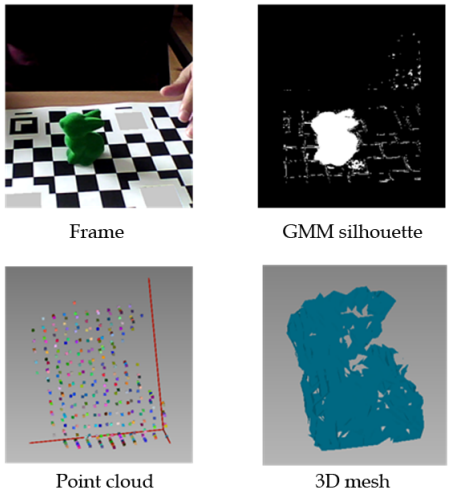


Figure 5.11: The result sequence of a bunny.

	Voxel number	frame number	convergence threshold	ponit cloud generating time	3D reconstruction time	Total time
apple	2406104	12	0.3%	9m 57s 587ms	1m 33s 823ms	11m 31s 410ms
pear	2406104	12	0.3%	7m 1s 251ms	31s 927ms	7m 33s 178ms
bunny	2406104	12	0.3%	8m 50s 621ms	54s 79ms	9m 44s 700ms

Table 5.1: The consumed time and parameters setting of the three results.

Chapter 6

Conclusion

This project is the continuation of 9th semester's project "Low cost 3D scanner" [7] and focuses on the completion of the acquisition module and the integration of the whole system. The basic tasks have been achieved, but still some optimizations are needed to make this 3D scanner system more practical.

6.1 Achieved work

6.1.1 The iterative algorithm

The core part of the work achieved in this report is the iterative algorithm which combines **GMM** and **space-carving-like method** and can obtain the silhouettes and the visual hull simultaneously. The training of GMM is achieved by utilizing the *gmdistribution* class of MATLAB. The space-carving-like method is performed to a voxel cube which is placed in the center of the world coordinate system and whether to carve a voxel is based on whether the voxel's projection on the image is inside the background or not. The space-carving-like method also can extract out the surface points of the point cloud. To assist the space-carving-like method, a **perspective projection algorithm** is implemented. It uses 3 transformations (pinhole camera transformation, image plane transformation and image transformation, cf. Figure 4.4) transforming the world coordinates into image coordinates. Later, to perfectly utilizing Aforge.net's glyph estimation result, the perspective algorithm is modified by making the pinhole camera transformation use Aforge.net's glyph transformation matrix directly. Also the old tracker is modified to better support the space-carving-like algorithm by increasing the glyphs' kinds from 1 to 4 and a chessboard pattern is attached to the tracker to help shadow eliminating and background subtraction.

The results in Section 4.2.4 have proved that the iterative algorithm is effective to extract the silhouettes and generate the visual hull point cloud simultaneously. But still it has the defect that cannot deal with the color possessed both by the foreground and the background properly. So optimizations are needed and are discussed later in Section 6.2.

6.1.2 Coordinate systems involved

In order to perform the space-carving-like method and the perspective projection algorithm, several coordinate systems are involved. The pinhole cameras and the voxel cube are all contained in the *world coordinate system* (cf. Section 4.2.1) and the space-carving-like method is performed inside this coordinate system. To perform the perspective projection, the *pinhole camera coordinate system* is introduced (cf. Section 2.3.1). Aforge.net's coordinate system is also a pinhole camera coordinate system. The pinhole camera has an image plane which has an *image plane coordinate system*, and the images displayed on this plane also have their own *image coordinate system*. The *world coordinate system* and the *pinhole camera coordinate system* are 3D coordinate systems while the *image plane coordinate system* and the *image coordinate system* are 2D coordinate systems. The goal of the perspective projection is to transform the world coordinates to the image coordinates through the pinhole camera coordinates and then the image plane coordinates.

6.1.3 C# and MATLAB integration

For the benefit of coping with GMM effectively, MATLAB programming is integrated into the C# project. Section 2.2 has introduced 3 ways of integrating MATLAB into C#. In this project, the first way and the second way are both used. During the developing phase, the first way is used to debug both the C# code and the MATLAB code. In the release phase, the second way is used to obtain a more easily deployed version.

6.1.4 The integration of the 3D scanner system

The accomplished acquisition module and the 3D reconstruction module are both integrated into CIR's software suit. The two modules both obey the MVC architecture and are well integrated into CIR's software suit's MVC architecture. The two modules use PCD file as intermedia file format and the final output of the whole system is a STL file. The operation procedures of the 3D scanner are demonstrated in Section 5.1.

6.2 Future work

6.2.1 The optimization of the iterative algorithm

The iterative algorithm has the defect of wrongly classifying colors into background colors currently. There are several ways may be useful in solving this problem.

First, the tracker's colors can be changed. Some uncommon colors can be found to make the combination of the tracker's colors. Several combinations can be prepared for different objects and the user can try to choose the tracker that has the most different colors with the scanned object.

Another idea is to add a function to recognize the black and white chessborad pattern. Then the tracker can be totally deleted as default background and be not used to compute the silhouette, so its colors will not affect the foreground. Based on this consideration, the tracker can also be modified into pure chessboard pattern.

A third idea is to utilize the region or structure information of the foreground. As the scanned objects are most simple convex objects, the foreground pixels are necessarily connected to each other, so they will form a region and the edge of the region is usually smooth. Using some parameters describing the region relationship between the foreground pixels and the smoothness of the edge

curve will benefit the segmentation of the foreground. The energy function used in the graph-cuts method just right contains one part to describe the smoothness between the pixel points (cf. Section 2.1.3) and can be considered to implement in the future.

6.2.2 Another possible way for tranigulation

Some possible improvements about the 3D reconstruction have already been presented in Section 6.4.1 of [7]. Relating to the point cloud generated by the space-carving-like method, a new way of triangulation may be possibly implemented. As the point cloud is carved from a set of tidily arranged voxels, it is still formed up by relatively tidy points. These points are located next to each other at a certain interval, and the neighbor points' positions change smoothly. Each point is actually a voxel which is a unit cube and has 6 facets. Each facet is formed up by 2 triangles. So the 3D mesh can be obtained by figuring out which facets of the voxels can be seen from outside the object, then collect these facets and make them into triangles.

Bibliography

[1] Aforge.net, *Aforge.net framework*, http://www.aforgenet.com/.

[2] ———, *Glyphs' recognition*, http://www.aforgenet.com/articles/glyph_recognition/.

[3] Yuri Boykov and Vladimir Kolmogorov, *An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision*, IEEE Transactions on Pattern Analysis and Machine Intelligence **26** (2001), 359–374.

[4] Yuri Y. Boykov and Marie-Pierre Jolly, *Interactive graph cuts for optimal boundary&region segmentation of objects in n-d images*, 2001.

[5] Neill Campbell, George Vogiatzis, Carlos Hernndez, and Roberto Cipolla, *Automatic 3d object segmentation in multiple views using volumetric graph-cuts*, In British Machine Vision Conference, 2007.

[6] Ian Davis, *Parallel matrix multiplication with the task parallel library (tpl)*, http://innovatian.com/2010/03/parallel-matrix-multiplication-with-the-task-parallel-library-tpl/.

[7] Emannuel Dhiver, Romain Herry, Vincent Fouillard, and Yi Li, *Low cost 3d scanner*, Tech. report, Department of Computer Science, Aalborg University, January 2012.

[8] Kiriakos N. Kutulakos and Steven M. Seitz, *A theory of shape by space carving*, International Journal of Computer Vision **38** (2000), 307–314.

[9] Aldo Laurentini, *Visual hull concept for silhouette-based image understanding*, IEEE transactions on Pattern Analysis and Machine Intelligence (1994).

[10] Wonwoo Lee, Woontack Woo, and Edmond Boyer, *Silhouette segmentation in multiple views*, IEEE Transactions on Pattern Analysis and Machine Intelligence **33** (2011), 1429 – 1441.

[11] Math.net, *Math.net*, http://www.mathdotnet.com/.

[12] Mathworks, *Gaussian mixture model curve*, http://www.mathworks.com/matlabcentral/fx_files/24867/2/gaussian_mixture_model.png.

[13] ———, *Matlab builder ne for microsoft .net framework*, http://www.mathworks.se/products/netbuilder/.

[14] Microsoft, *Task parallel library (tpl)*, http://msdn.microsoft.com/en-us/library/dd460717.aspx.

[15] PCL, *Point cloud library*, http://pointclouds.org/.

[16] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake, *"grabcut": Interactive foreground extraction using iterated graph cuts*, ACM TRANSACTIONS ON GRAPHICS **23** (2004), 309–314.

[17] Dan Snow, Paul Viola, and Ramin Zabih, *Exact voxel occupancy with graph cuts*, In IEEE Conference on Computer Vision and Pattern Recognition, 2000, pp. 345–352.

[18] Wikipedia, *The application area of 3d scanner*, http://en.wikipedia.org/wiki/3d_scanner.

[19] ———, *Automatically tuned linear algebra software*, http://en.wikipedia.org/wiki/Automatically_Tuned_Linear_Algebra_Software.

[20] ———, *Component object model*, http://en.wikipedia.org/wiki/Component_Object_Model.

[21] ———, *Extrinsic parameters of the pinhole camera*, http://en.wikipedia.org/wiki/Camera_resectioning#Extrinsic_parameters.

[22] ———, *Math kernel library*, http://en.wikipedia.org/wiki/Math_Kernel_Library.

[23] ———, *Normal distribution*, http://en.wikipedia.org/wiki/Normal_distribution.

[24] ———, *Perspectiveprojection*, http://en.wikipedia.org/wiki/3D_projection#Perspective_projection.

[25] ———, *Simd*, http://en.wikipedia.org/wiki/SIMD.

[26] Math works, *gmdistribution.fit*, http://www.mathworks.se/help/toolbox/stats/gmdistribution.fit.html.

[27] Gang Zeng and Long Quan, *Silhouette extraction from multiple images of an unknown background*, In Proceedings of the Asian Conference of Computer Vision, 2004.

[28] Chiyuan Zhang, *Clustering (3): Gaussian mixture model*, http://blog.pluskid.org/?p=39.