3D Point Cloud

Weichun Xu; Yutian Gui  
Florida Institute of Technology

# Abstract

*A Point Cloud is a set of data points in some coordinate system. In a three-dimensional coordinate system, these points are usually defined by X, Y, and Z coordinates, and often are intended to represent the external surface of an object.*

*This document is on how to get a 3D Point Cloud of a 3D image. In the processing, we use knn (2-nn) to get the k-nearest neighbors of each point and use each group of knn points to get a plane. Then we calculate the angle between places and if the angle is less than threshold, we think of these 2 points are almost in the same plane and remove one of these 2 points.*

*Current algorithm to calculate knn and remove points on the same plane need to compare each 2 points of a 3D image, which will use 2 for-loop to go through all the data. However these data processing are independent, therefore using GPGPU parallel programming to reduce the computing time is possible. With the parallel computing, the time complexity of the algorithm will be changed from O(n2) to O(n).*

# INTRODUCTION

CUDA: Compute Unified Design Architecture, which is a parallel computing platform and provides the function of general purpose graphics processing unit (GPGPU) computing.

Point Cloud: A set of data points in some coordinate system

3D Point Cloud: A set of data points in a three-dimensional coordinate system

k-NN: k-Nearest Neighbors

3D Point Cloud is very useful in 3D image reconstruction, which can be used in visualization, animation, medical image, etc. With the developing requirement on the high quality image (animation) and the accuracy of 3D image (medical image), the workload on data processing of extracting 3D Point Cloud is getting more and more time consuming. This paper is focused on how to extract 3D Point Cloud from a 3D image with GPGPU to reduce computing time.

**Problem Statement.** Using CPU to obtain 3D Point Cloud is always time consuming, especially when the image is getting larger and larger.

To extract 3D Point Cloud, we need to use knn (k=2) to obtain 2 nearest points of each point of a 3D image and use each 3 points (the original point and its 2 nearest neighbors) to generate a triangle (or a plane). Then we need to compare the angle between these planes and remove some points if they are on the same plane.

To calculate knn (k=2), for each point (called original point thereafter), we need to go through all the other points and compare these points with original point, which needs 2 for-loop (time complexity: O(n2)) to realize this.

When comparing angles between planes to remove the points on the same plane, we need to compare each plane of a point with the planes of the subsequent points of the image, whose time complexity is also O(n2).

Therefore, the time complexity of using CPU sequential computing is O(n2) and normally 3D image is always very large, the time used in computing and extracting 3D Point Cloud will waste us a lot of time.

**Motivation.** Point Cloud can be used in a lot of areas in current society, including visualization, animation, rendering and [mass customization](http://en.wikipedia.org/wiki/Mass_customization) applications. 3D point cloud is very useful in [medical imaging](http://en.wikipedia.org/wiki/Medical_imaging) to represent volumetric data. It is also one of the sources to make [digital elevation model](http://en.wikipedia.org/wiki/Digital_elevation_model) of the terrain in [geographic information system](http://en.wikipedia.org/wiki/Geographic_information_system). It can also be applied in generating 3D model of urban environment.

**Proposed Solution.** The current algorithm, using 2 for-loop, to extract 3D Point Cloud with CPU needs to take a lot of time and the time complexity of computing is O(n2).

When we analyze the algorithm and data processing of 3D Point Cloud, we find the calculation can be run in parallel, so it’s possible that we can use GPGPU to shorten the computing time. Using GPGPU parallel computing, we can reduce the total time complexity to obtain 3D Point Cloud with GPGPU to O(n).

**Contributions.**

This project makes the following contributions:

* Use parallel programming to efficiently generate 3D Point Cloud
* Shorten the time in obtaining 3D Point Cloud

## Design Goals

3D Point Cloud is useful for 3D modelling and 3D image reconstruction. This project purpose is to use CUDA parallel computing to significantly shorten the time to get 3D Point Cloud from a large image.

In the design, we will use knn (k=2) to get 2 nearest neighbors of each point of a 3D image. To realize this, we will use each thread to compare one point with all the other points one by one. So the 2 for-loop algorithm in CPU will be changed to 1 for-loop in GPGPU.

Then in generating planes between each 2 points (the 2 chosen points and these 2 points’ 2-nearest neighbor points will create 2 planes), calculating the angle between planes and removing one of the 2 points if they are on the same plane. We use CPU to go through all the points, and for each point (called original point thereafter), we use each thread of GPGPU to calculate the angle between each of the subsequent points (after the original point) with the original point and remove the point if the point is on the same plane as the original one. Here the reason why we use CPU to go through all the points is because we need to synchronize each time when an original point is compared and GPGPU synchronization can only be done in block while the size of threads of GPGPU computing will be much larger than the size of threads limited in a block.

## Projects Proposed Features and Functionality

* CUDA will be used to calculate which will significantly shorten the computing time
* Knn is used to obtain 2 nearest points of each point, which can be used to get a line or plane

# RELATED WORK

3D Point Cloud can get the set point set of 3D model, it means that by using 3D Point Cloud, people can represent a 3D model by a set of points. Then, associate with KNN algorithm, people can reduce the amount of a large set of data to describe a 3D image with only a smaller data set, which will reduce the requirement of disk space to save the data set and the execution time of 3D image reconstruction.

There are many ways to achieve the 3D Point Cloud with many kind of KNN algorithm. According to some paper, we learned some algorithms and made some comparison. By using simplest algorithm, the complexity is O(n2). However, by using some advanced algorithms, the complexity can be reduced to O(n\*logδ).(from reference [SSV06]). However, even the complexity reduced, it still costs a lot of time to execute the algorithm.

In our project, we try to use some other ways to reduce the complexity. There are many ways to simulate the algorithm, but as we introduced before, using traditional ways (executed by CPU) to calculate the result of 3D Point Cloud will cost a lot of time. This is because nowadays, mostly computers only have 1-8 cores, it has a low efficiency to run parallel task, and few tools use GPGPU to compress the calculation time while a lot work of 3D Point Cloud extraction can be run in parallel. What we want to enhance is to use GPGPU to obtain 3D Point Cloud, which will shorten the calculation time significantly. Extraction of 3D Point Cloud from a large 3D image with CPU may take tens of minutes to process. By this way, we can reduce the complexity much more than by optimizing the algorithm itself. As a result, we can reduce the complexity from O(n2) to O(n).

3D Point Cloud can be used to reduce the amount of a large set of data to describe a 3D image with only a smaller data set, which will reduce the requirement of disk space to save the data set and the execution time of 3D image reconstruction.

In the paper, we can see that in some application hundreds of seconds are used in calculating 3D Point Cloud. However we can reduce the computing time significantly by using GPGPU parallel computing. The only limit in GPGPU computing is that OS will terminate the kernel application if the kernel runs a long time. To solve the issue, it’s better to have a GPGPU computing server instead of using a GPGPU card which needs to run video display simultaneously.

# PROJECT PROPOSAL

In the project, we will use GPU to calculate knn of each point of a 3D image and remove some points if the points are almost on the same plane.

## Anticipated Approach

In the design, we will use GPGPU parallel computing to replace CPU sequential computing, we can change the 2 for-loops to 1 for-loop (the 1 for-loop can be run in CPU or in each thread of GPGPU), whose time complexity is O(1), and another for-loop can be realized in GPGPU parallel computing, which has a time complexity of O(1), so the total time complexity to obtain 3D Point Cloud with GPGPU will be reduced to O(n).

First, we will use GPU to calculate knn (k=2) to get 2 nearest neighbors of each point of a 3D image. In the process of calculating of the knn, the equation will be used to calculate the length of each point of a 3D image to all the other points, then pick up 2 points which have the shortest length.

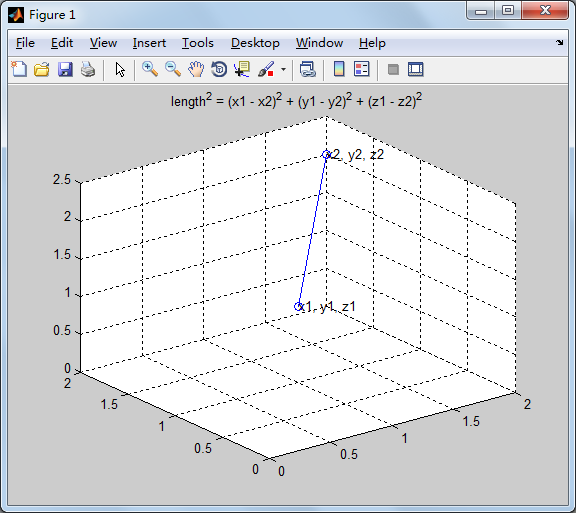


Figure 1: *equation to calculate the length*

When selecting the 2 points, we need to eliminate the situation that 3 points are on the same line. We need to make sure the 3 points can create a triangle. To achieve this, the following method is used to solve the problem: use these points to obtain vector, normalize the vectors and check if the normals of the vectors are the same. If they are the same, it means that the points are on the same line and we need to pick up another nearest point.

Then we use the equation to calculate the distance of each 2 points of the image.

If the distance is less than distance\_threshold (distance\_threshold is parsed through program argument), we use these 2 points and its 2-nn neighbor points to generate planes.

Then we use equation to calculate the angel between the planes generated by these 2 points and their 2-nn neighbor points. If the angel is less than angle\_threshold (angle\_threshold is parsed through program argument), either of these 2 points will be removed.

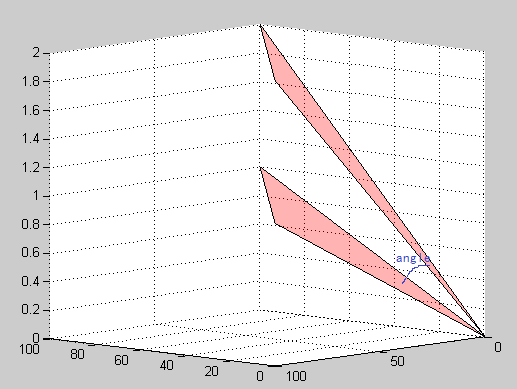


Figure 2: *calculate the angle between 2 planes*

## Target Platforms

CUDA will be used to calculate knn and angle between planes.

Visual Studio will be used to compile and run the code.

Nvidia GPU GT 525M is used in GPGPU computing.

Matlab is used to generate original 3D image used as simulation data set.

## Evaluation Criteria

Current algorithm to process 3D Point Cloud is based on CPU, so we will use CPU to realize the design and in the design we will print the time used in each step of the program. With our modified algorithm, GPGPU will be used in extracting 3D Point Cloud and the time used in allocating memory, copying memory and GPGPU computing will be shown in the program.

The information of time used in CPU program and GPGPU program has been recorded and compared in a diagram (Figure 3), which shows the difference between CPU based program and GPGPU based program to extract 3D Point Cloud.

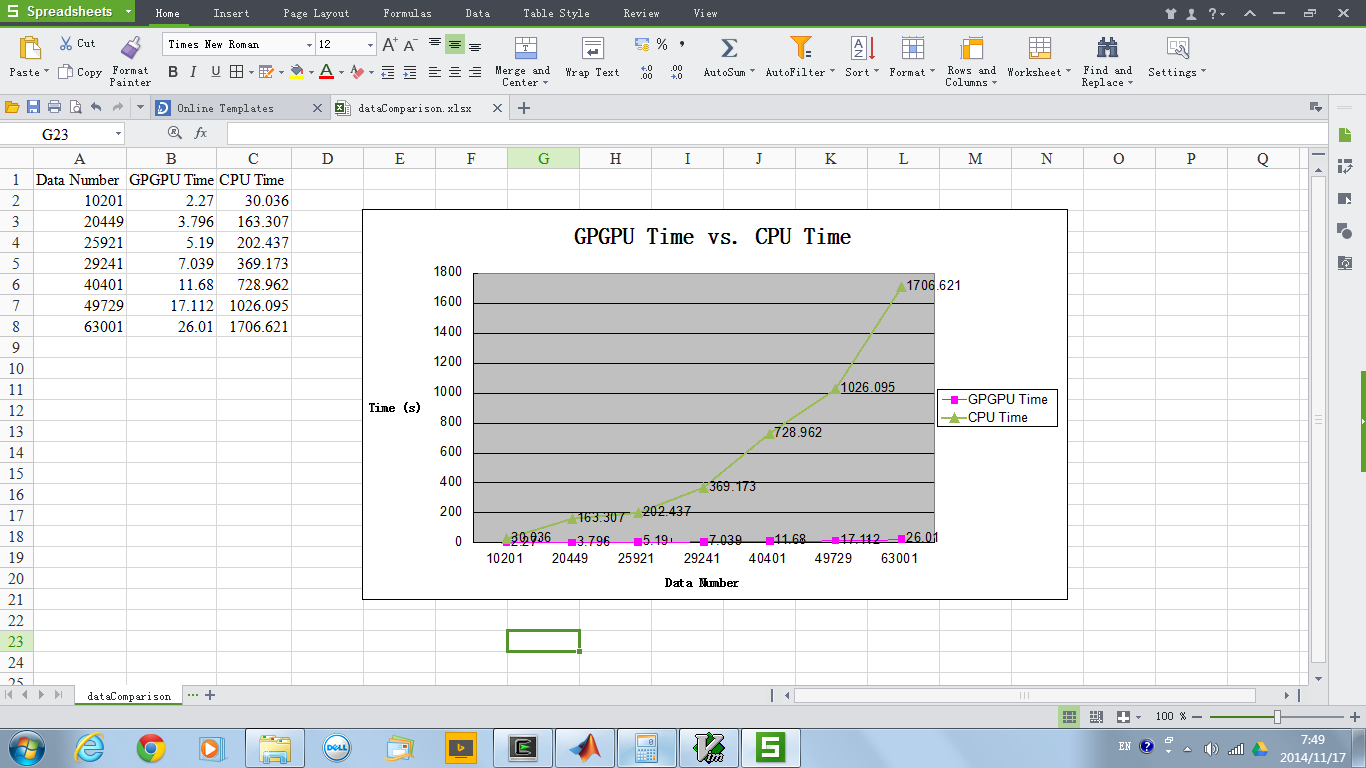


Figure 3: *execution time of CPU-based design vs. execution time of GPU-based design*

# RESEARCH TIMELINE

**Project Milestone Report (Alpha Version)**

* Search on the internet for 3D Point Cloud
* Read and understand the paper of 3D Point Cloud
* Prompt the realization of 3D Point Cloud
* Analysis and find a way to generate simulation data
* Analyze the time of the tasks and make the schedule

**Project Final Deliverables**

* Software Source Code
* Demo of the design
* Document of the design and the result of simulation

**Project Future Tasks**

* With current algorithm, there will have issues around the edge of the 3D image when extracting the Point Cloud (refer to Figure 4):

When comparing Point A and Point B, it is better to keep Point B rather than Point A, but current algorithm will remove either of Point A and Point B

If d2 < distance\_threshold, Point C may be removed when comparing Point C with Point B

d1

d2

Point A

Point B

Point C

Figure 4: *Issue of the algorithm*

Therefore, if more time is given, a better algorithm should be studied and used to solve the issue.

Based on the design requirement and the time available to do the project, a detailed schedule plan is made. Figure 15 is the GANT chart of the schedule plan of project.

# Method

To calculate the 3D Point Cloud of a 3D image, the data of the points of the image is read by the program. Each data of the points have 3 variables: x, y and z.

After reading the image data, we allocate memory space on GPU memory and copy the image data from host to device. Then we calculate the knn of each point of the 3D image and remove some points which are on the same plane. Finally we write the left points to a file.

When calculating Knn, at the beginning, for each point (we call it the original point in the following), if the point index is larger than or equal to 2, we pick up point 0 and point 1 as the 2 nearest points, otherwise we pick up point 2 and point 3 as the 2 nearest points. After picking up the first 2 nearest points, we sort these 2 points and put the nearer one to variable knn\_data0 and the farther one to variable knn\_data1. Then we go through each of the point of the image (if the point is the same point as the original point, skip to the next point) and calculate the length between the point and the original point. This point will be assigned to variable knn\_data2 and will be compared with variable knn\_data0 and knn\_data1. After sorting variables knn\_data0, knn\_data1 and knn\_data2 from low to high, we check if the line created by point knn\_data0 and original point is on the same line as the one generated by point knn\_data0 and original point. If they are on the same line, check if the line combined by knn\_data0 and original point is on the same line as the one with point knn\_data2 and original point and if no, knn\_data1 will be replaced by knn\_data2.

But there is issue when running the GPU program: as the GPU code takes a long time, the GPU will send out a timeout signal and kill the program (refer to Figure 5). So we separate the GPU code and call the GPU function multiple times in CPU code (refer to Figure 6 and Figure 7).



Figure 5: *GPU timeout*

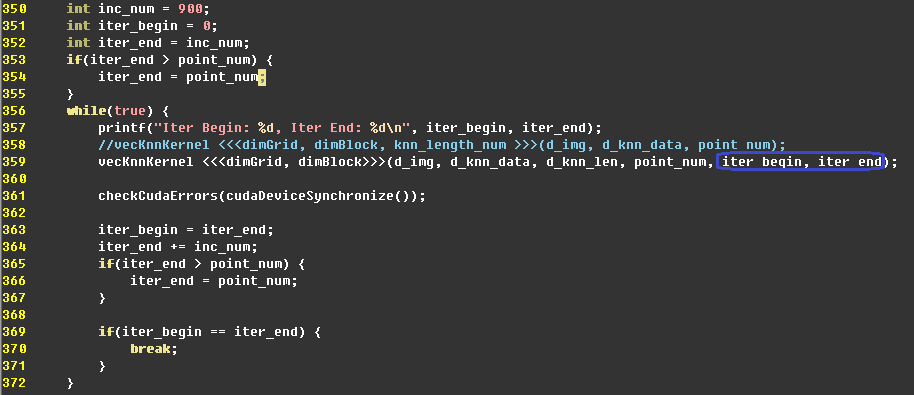


Figure 6: *call GPU function multiple times (CPU code)*

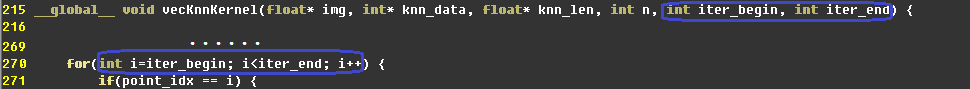


Figure 7: *call GPU function multiple times (GPU code)*

When removing points which are on the same plane, we go through all the points and for each point (if the point is not removed) we will calculate the points, which are the following points of the current point, to check if the length between these points and the current is less than or equal to distanceThreshold. If the length is less than or equal to distanceThreshold, we will check if these points (each of the point with its 2 nearest neighbors to generate a plane) are on the same plane (the cosine of the angle between these 2 planes is greater than angleThreshold) as the current point (the current point and its 2 nearest neighbors generate a plane). If any of the points is on the same plane, the point will be marked as removed.

However when removing the unused points, we need to synchronize each time when all the other points are compared with one point, and for GPU, the synchronization can only be done in block. So we realize the synchronization through the code on CPU by calling the GPU function cudaDeviceSynchronize() each time when we need to synchronize (refer to Figure 8).

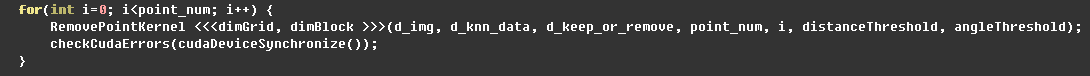


Figure 8: call GPU function multiple times (GPU code)

In the program, there are 2 methods used in calling the kernel function to remove the points on the same plane.

One method (Figure 9) is that after calling the kernel function, data will be copied back to host and in the next loop if the point has already removed, there is no need to call the kernel function. This method will reduce the number of calling the kernel function but will increase the overhead in memory copy. Therefore the method is applicable when many points are on the same plane and many points can be removed in one time running of kernel function.

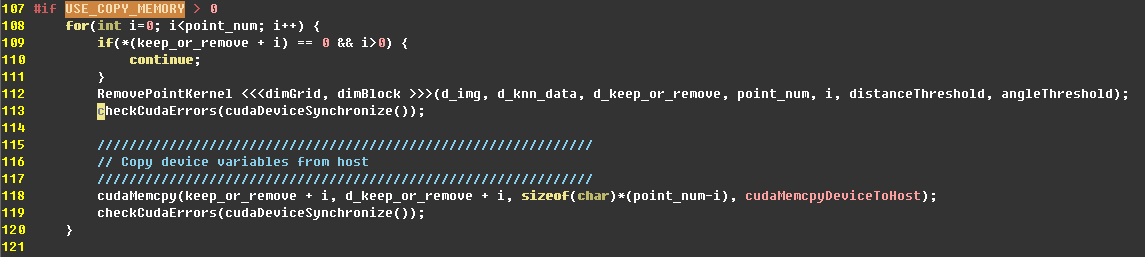


Figure 9: copy memory every time after calling kernel function

Another method (Figure 10 and Figure 11) is that the kernel function will be called no matter whether the point is removed or not and the data copy only happens when running kernel function is finished. In the way, there is low overhead in memory copy but this will cause the overhead in calling kernel function. Therefore this method is mainly used when only a few points are on the same plane and only few points can be removed in calling kernel function once.

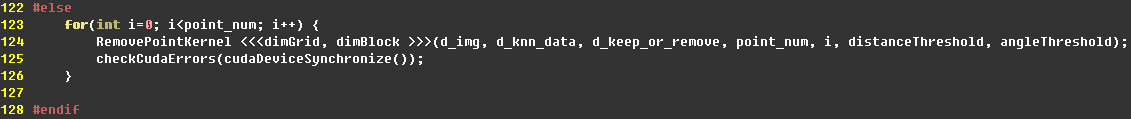


Figure 10: no memory copy each time after calling the kernel function

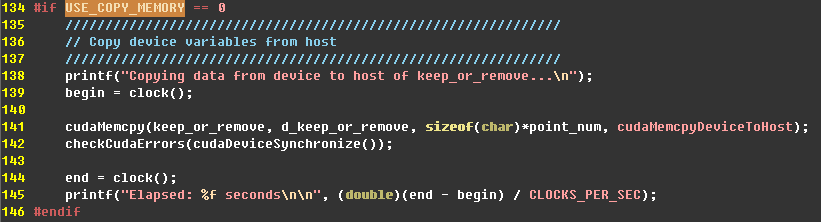


Figure 11: memory copy after all the calling kernel function is finished

# RESULTS

In the program, we also use CPU to extract 3D Point Cloud and compare the execution time between CPU and GPU. From Figure 12, Figure 13 and Figure 14, the execution time used in GPU computing is far less than the execution time used in CPU.

In Figure 12, time used in device memory allocation of d\_img, d\_knn\_data and d\_knn\_len is printed by the program. In Figure 14, it shows the time of memory allocation of device for keep\_or\_remove.

In Figure 12, the time of copying memory from host to device for a 3D image is displayed. In Figure 14, it shows the time of memory copy of knn\_data from device to host.

In Figure 13, the program prints the time used in running the kernel function of knn and the kernel function of removing point.

In Figure 14, it shows the time of CPU-based computing of 3D Point Cloud (running knn\_c and remove\_point\_c).

Therefore, from the figures (Figure 12, Figure 13 and Figure 14), we can see that the total time, including allocating device memory, memory copy and GPU execution, is only about 12 seconds while CPU takes more than 700 seconds to calculate the 3D Point Cloud.

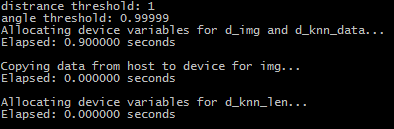


Figure 12: *execution time (1)*

When using GPGPU to calculate 2 nearest neighbors of each point of an image, as the image is very large and it takes a long time to run the kernel program, there is a timeout error of kernel. Therefore we separate the whole data set into several segments of data set and call the kernel function for each segment sequentially. In Figure 13, the message printed by the program shows the starting point and ending point of each segment when calling kernel function. By default, the size of a segment in the program is 900 points and it can be changed in the source code. In the future, if the variable needs to be changed frequently, we can bring this variable to the argument of the program.

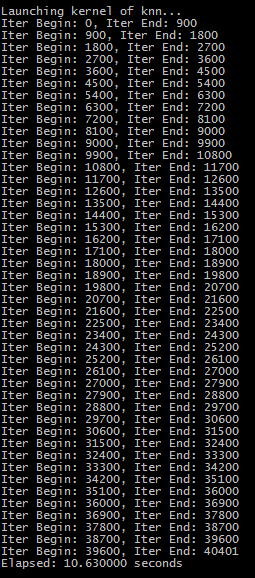


Figure 13: *execution time (2)*

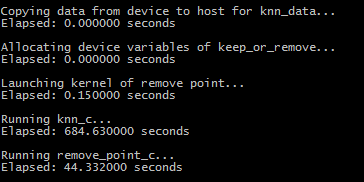


Figure 14: *execution time (3)*

# CONCLUSIONS and FUTURE WORK

From the algorithm and the simulation result, we can find the execution time using GPGPU parallel computing is reduced significantly compared to CPU computing. Therefore, using GPGPU to extract 3D Point Cloud from a 3D image is a feasible way and it will save a lot of computing time.

Currently, due to the limit time, we don’t have time to optimize the algorithm to solve the issue that happens at the edge of the image. In the future, a better method should be applied to find the edge of a 3D image to solve the issue.

**APPENDIX**

1. **Optional Appendix**

Build the program:

Change directory to project and use visual studio to open the project.vcxproj. Change the “Additional Include Directories” of C/C++ and CUDA C/C++ of the project properties if the directory of the installed CUDA header file is not the same as the one in the project. After updating the include directory, build the project and the binary program “project.exe” will be copied to directory simulation.

Run the program:

In the command window, change directory to simulation and run the command: .\project.exe [-d distance] [-a angle] [-f dataset]

[-d distance]: parse distance\_threshold through program argument, by default the value is 1

[-a angle]: parse angle\_threshold through program argument, by default the value is 0.99999

[-f dataset]: parse dataset file through program argument, by default the value is “dataset”

# References

|  |  |
| --- | --- |
| [GWM02] | Stefan Gumhold, XinlongWang, Rob MacLeod: Feature Extraction from Point Clouds. |
| [DHOS07] | Joel Daniels II, Linh K. Ha, Tilo Ochotta, Cl´audio T. Silva: Robust Smooth Feature Extraction from Point Cloud. |
| [WHH10] | Christopher Weber, Stefanie Hahmann, Hans Hagen: Sharp Feature Detection in Point Clouds. IEEE INTERNATIONAL CONFERENCE ON SHAPE MODELING AND APPLICATIONS (SMI) 2010. |
| [MD03] | Carsten Moenning, Neil A. Dodgson: A new point cloud simplification algorithm. Presented at 3rd IASTED International Conference on Visualization, Imaging, and Image Processing (VIIP 2003) 8-10 Sep 2003. |
| [SSV06] | Jagan Sankaranarayanan, Hanan Samet and Amitabh Varshney: A Fast k-Neighborhood Algorithm for Large Point-Clouds. Presented at Eurographics Symposium on Point-Based Graphics (2006). |

|  |
| --- |
|  |

Figure 15: schedule plan