3D Reconstruction With Kinect

Li Pei, Yiming Zhang

lip@andrew.cmu.edu, yimingz2@andrew.cmu.edu

**Abstract.** 3D Model Reconstruction is of great meaning in many aspects of society. However, modern large scale 3D scanner are expensive, not portable and need to be controlled by professionals, which is not applicable for individuals and small team. In this project, a cheaper, user-friendly and mobile solution for 3D Model Reconstruction was implemented with Microsoft Kinect. Based on the depth and RGB camera of Kinect, we extract point cloud and corresponding RGB value of model and background. Then we use threshold method to delete background and get model. Using the foreground point cloud, we implement Delaunay Triangulation turning point cloud into a set of surface. Last step, after we get front and back of the model in previous process, we could align 3 corresponding point in the front and back to build up a whole 3D model.  
  
**Keywords**: Depth Image, Point Cloud, Triangulation, 3D Reconstruction

1 Introduction

3D Model Reconstruction is of great meaning in modern society, which build the foundation of several cutting-edge technology, like 3D printing. And with the development of 3D scanner, non-contact 3D Model Reconstruction technology is embracing its prosperity. However, modern large scale 3D scanners are expensive, lack mobility and need to be controlled by professionals, which inspire us to build a relatively low cost, mobile scanner with moderate accuracy.

In this project, a cheaper, user-friendly and mobile solution for 3D Model Reconstruction was implemented with Microsoft Kinect. Based on the depth and color camera equipped in Kinect for interacting with user, data of objects for reconstruction can be collected and output using Kinect’s SDK. With sensor-collected data, 3D Model Reconstruction is applicable because position and color of each pixel in 3D space can be obtained, and some post-processing could be applied for 3D Model Reconstruction.

The general approach of our project can be divided in several consecutive steps. The first is collecting depth and RGB value of object into a format as point cloud. Then we could extract the foreground from these points. With this foreground points we can do Delaunay Triangulation and turn them into surface. And if we get the front and back of a body, we could combine it by aligning three corresponding points in the front and back, and we can get the reconstructed model.

The equipment we use is the first generation of Kinect and we also take advantage of software. In this project, we use Microsoft Visual Studio for collecting data from Kinect, Matlab for 3-point-alignment and Delaunay Triangulation, Meshlab and ProEngineer for present the 3D reconstructed model.

The paper was organized as below. First we give the introduction and then is the literature review. The third part is the demonstration of how to implement our solution. We will present all the results in the 4th part and then give analysis on Part 5. Part 6 will be our brief conclusion and all the reference could be read in Part 7.

2 Literature Review

This project is originating from the area of surface reconstruction from dense range data. The basic two directions in this area are reconstruction from unorganized points, and reconstruction that exploit the underlying structure of the acquired data [1].

In past, there had been two main methods to get unorganized points, multi-view stereo and laser distance measurement. The first is the process taking two or more images and estimating the 3D model by finding matching pixels of image, which is widely used in photogrammetry [2]. The second method is widely used by different types of 3D scanners. Almost all of these scanners calculate the distance from object surface to the scanner by measuring the time difference between beaming the laser and detecting the reflection. This calculation is based on extremely accurate timer which is expensive, and has relatively low resolution and no texture [3].

In addition to the 3D scanner, some cameras came into exist these years which generate depth maps at video rate. There small size and ability is suitable in multiple application areas [4]. The principle of this category of camera is also to measure the time of difference, so most of them are accurate but expensive, except for Kinect. That is why we choose Kinect in this project for reconstruction.

3D reconstruction is a well-studied area. Article [5] introduces an accurate and practical calibration method of a depth and color camera pair. In [6] Khoshelham describes the calibration of Kinect with OpenNi drivers and intensively analysis the accuracy of Kinect. ICP (iterative closest point) algorithm [7] is used to estimate the relative camera pose between two models and RGB and depth combining version which is used in RGB-D sensors is described in [8]. Many software and open source library about computer vision brings a lot of convenience in study. The OpenCV [9] library offers the implementation of almost all algorithms in computer vision. The PCL [10] offers methods in point cloud management such as model registration and merging. The Geomagic software [11] offers easily handle to 3D models.

3 Approach Descriptions

3.1 Getting Depth and RGB image with Kinect

To reconstruct body of object such as body, it is recommended to start from collecting point cloud of the surface, and the quality of collection will affect the reconstruction. In this project, we collect cloud point with the first generation of Microsoft Kinect. Comparing with the modern large scale 3D scanner, it is far less expensive, more portable and easy to implement.

The first generation Kinect was equipped with a RGB camera, an infrared camera and infrared light source. With RGB camera we can get RGB image and with infrared camera and light source we can get the depth image.

It is easy to implement a Kinect with your personal computer. We will give a brief description here. First you must assure your computer support DirectX 11 for image processing and Visual Studio is highly recommended for communication with your Kinect and it is required for install Kinect Windows SDK. After a correct process of installation, you could run a sample provided by SDK which prove the correct function of your Kinect and computer.

After the initialization, we can collect data from camera with the API provided by SDK, here we provide a list of API relevant to our project,

**Table 3.1. Kinect API provided by Kinect for Windows SDK**

|  |  |
| --- | --- |
| API Name | Function |
| NuiGetSensorCount | Get the available number of sensor |
| NuiCreateSensorByIndex | Create an interface of sensor |
| INuiSensor::NuiStatus | Check if the sensor connected |
| CreateEvent | Initialize the depth data streaming out of sensor |
| INuiSensor::NuiImageStreamOpen | Open the depth stream for collecting depth data |
| INuiSensor::NuiImageStreamGetNextFrame | Get depth data of next frame |
| INuiSensor::NuiImageStreamReleaseFrame | Save frame of the released color data |
| INuiSensor::Release | Release the sensor |

In this project, we use both point cloud with color and without color, and we implement 2 different sets of API, further information is in the part of our citation and code.

3.2 Extract foreground in Kinect Depth Image Output

Compared with color image, the depth image is more important for 3D reconstruction. In this part, we will present our approach to processing the output data of Kinect depth image to extract the foreground object and delete the background.

The depth data collected by Kinect takes unit of millimeter (mm) and range from 800 to 4000 mm. However, the image pixel we get is 640\*800. If we do not process the depth data and use the raw data, the 3D reconstruction will be distortion because the data in depth is more in number and larger in value. Furthermore, if we want to apply the Delaunay Triangulation, the data point not belonging to the object will result in wrong connection of triangle and distort the surface.

From our perspective, to avoid the problem from the depth issue, we can delete the background and extract the foreground from the image. In this project, we use threshold to separate the object to be reconstructed out of the environment.

To extract the foreground, we use the double-threshold method to get the foreground. Based on the fact that the body of human is in a reasonable range and we could get the points in this range and delete every other points. For example, if we preset and as the reasonable min and max depth value of the object, and we have raw data point, then we set all the points with the function

|  |  |
| --- | --- |
|  | (**3.1**) |

The extraction of double threshold is accurate because it did not distort the original depth relation. With this processed depth image, we could do further work on the 3D reconstruction.

3.3 Building 3D Coordinate with Depth Image

After obtaining the processed depth image, we can transform all the points into a point cloud. A plausible can guarantee the triangulation and reconstruction of human body.

Using Kinect we can get point with depth, however, the point can only carry one valid value as depth. To get complete 3D value, we have to build a 3D coordinates. There are 2 different ways for the building process.

3.3.1 Getting Image Coordinate

The first kind coordinate is image coordinate. Based on image, we can treat every row and column index as an (*x,y*), combined with the depth we get a 3D coordinate which represent the coordinate of the points.

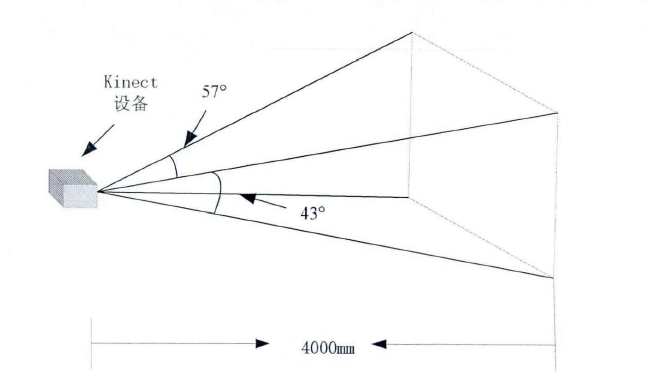
This is an easy way to reconstruct with relatively low accuracy, because it distort the relation and proportion of original object.

3.3.2 Getting Space Coordinate

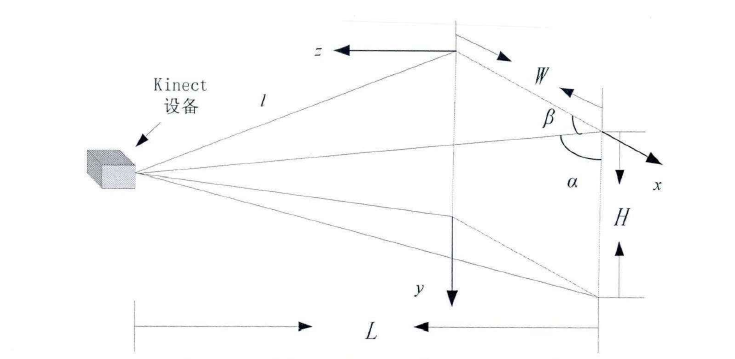
As mentioned in 3.3.1, the image coordinate will brought in distortion and lose part of the points relation. In order to get the correct proportion of points, we will build a space coordinate to represent the real physical coordinate.

To build the real coordinate, we have to build up the transformation method based the imaging principle of Kinect. The view field of Kinect is limited in a pyramid shown in Fig 3.1 The vertical view angle is 43 degree and horizontal view angle is 57 degree. The largest detect range is 4000mm.

To get the space coordinate, we could build the coordinate according to this view field. The parameter is preset in Fig 3.2.



**Fig3. 1** The view Field of Kinect



**Fig3. 2** Kinect Transform Parameter in View Field

In Fig 3.2, *L* is Kinect’s largest detect distance, the largest height and width of view is *H* and *W* and and are the corner angle for two side, then we can get

|  |  |
| --- | --- |
|  | (**3.2**) |

Assuming in the depth image, in height direction we get points and in width direction we get points. The actual height and width distance between two points are *h* and *w*, then we get

|  |  |
| --- | --- |
|  | (**3.3**) |

Combined with (3.2), we get

|  |  |
| --- | --- |
|  | (**3.3**) |

Assuming a point P is (*x,y*), and the corresponding point in image coordinate is (), and the coordinate in the field is (),

|  |  |
| --- | --- |
|  | (**3.4**) |

If we know the points original depth information, the space coordinate of this point is

|  |  |
| --- | --- |
|  | (**3.5**) |

With the known parameter, we get the space coordinate

|  |  |
| --- | --- |
|  | (**3.6**) |

This is an accurate coordinate, however the computation is complicated, in this project we use Matlab to solve this computation problem.

3.4 Present the Point Cloud

After all the process in previous section, we can get an accurate set of point cloud now. To present this cloud, we use the an CAD software called ProEngineer. If we output the point cloud with (*x,y,z*) format in .txt file, if we change the suffix to .pts, we can insert all the point into ProEngineer and get the point cloud figure.

3.5 Splice Point Cloud of Different Surfaces

In the previous process, the point cloud data we get is only one direction, like the front or back of body, to implement the 3D reconstruction we have to collage at least two images of front and back. In this project, we implement the three-points align method to implement splice process.

Three-points align is a simple splice method, which is a kind of mark point method. To reconstruct a model, we have to set three base point that can be seen in both front and back of it, and transform all the points according to these three points.

The general process of splice is get base point , and . Then another image with the same base point , and . Then we get vector and . Then we get

|  |  |
| --- | --- |
|  | (**3.7**) |

and

|  |  |
| --- | --- |
|  | (**3.8**) |

With the property of cross product, *V* and *W* compose two right-hand orthogonal coordinate. Then we unitization *V* and *W* to get unit coordinate *v* and *w*. For any points in [*v*] to [*w*] , we have

|  |  |
| --- | --- |
|  | (**3.9**) |

And the rotation matrix is

|  |  |
| --- | --- |
|  | (**3.10**) |

To get a special point to make and , we can get the translation matrix

|  |  |
| --- | --- |
|  | (**3.11**) |

And the final function is like

|  |  |
| --- | --- |
|  | (**3.12**) |

3.6 Triangulation

3.6.1 Triangulation for Points in One Direction

Using a set of triangle to approximate the surface of object is a common surface reconstruction method. To get this triangle, we have to triangulation the point cloud. With triangulation, we could build up the topology relation for all the points in point cloud. Triangulation could be described as a method to connect all the points to form a set of tetrahedron without overlapping and gap. In this project, we implement the Delaunay Triangulation.

Delaunay Triangulation is a special kind of triangulation. First we have to define a Delaunay Side, assume E has one side e with two end a and b, it is a Delaunay side if there exist a circle with a and b on it, and no other point of point cloud V in this circle. It is also called Empty Circle Property. And if a triangulation of point cloud V only contain Delaunay triangulation, this triangulation is Delaunay Triangulation.

To be conclude, there are two important principal of Delaunay Triangulation. The first is Empty Circle Property and the second is Maximization of Minimum Angle, which means the triangle out of Delaunay Triangulation is with largest inner angle against other triangulation method.

There are some property of this triangulation method. The triangulation will be the same no matter you start point or region. Every triangle will have the largest inner angle. If you add or delete points, it will only affect neighbor points, and it will output convex polygon crust.

In this project, we use Matlab delaunay(*x,y*) function and triplot(*tri,x,y*) function to generate and present Delaunay Triangulation. We use MyCrust and trisurf(tri,x,y,z) function to generate the final result. The delaunay(*x,y*) function is based on Delaunay algorithm. First we index every points and project them to (*x,y*), and then it operate triangulation according to the Delaunay’s algorithm and the triplot function connect all the points together according to their index. MyCrust is an oper-source function in Matlab which calculate the tetrahedronation of the figure and use trisurf to present the final result.

3.6.2 Triangulation for Points in Multiple Direction

In 3.6.1, we discussed how to triangulation point clouds in one direction. However, to reconstruct a 3D model, we have to do triangulate for multiple directions. Things are more complicated here because we cannot project points in (*x,y*) because every (*x,y*) will have two different depth value. To rebuild the surface for a collaged point cloud, we have to use different method.

In this project, we use Giaccari Luigi’s MyCrust algorithm based on Matlab to triangulate the point cloud. This algorithm can be only used in a closed form point cloud. It return a triangle surface of three input point, and it has good performance for point cloud with large density.

The process of MyCrust start with a random point in point cloud. Let it be . If we choose nearest to , then we get a side of triangle. Then we find the optimal vertex for this side. If we find it, we set ,.and as 1, representing they are used, and choose the point after as a new start for searching. After iteration process, we could get surface of the body.

Here we will specify how to choose the optimal vertex. Assume we will find the optimal vertex for edge E, we think the optimal vertex is the point with minimum distance sum to the both end of E, and the minimum inner angle of this triangle is larger than 35 degree, and the maximum inner angle of this triangle is smaller than 90 degree. The MyCrust algorithm using bounding box method. We choose the largest side of the triangle multiply 2.5 as side for a cube, and both end of this side are vertex of this cube. We find the optimal vertex for triangulation in this box and reduce the amount of computation.

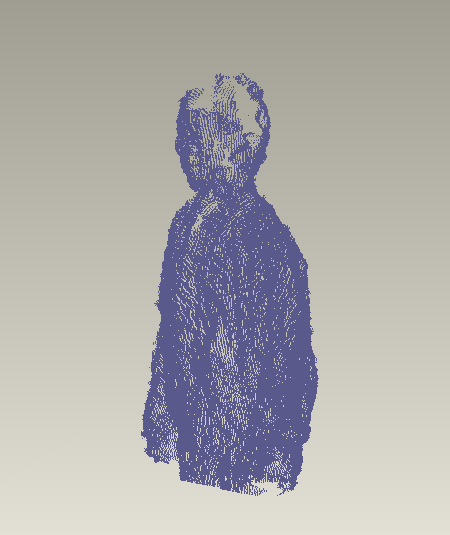
4. Results

4.1 Point Cloud Image of Front

|  |  |
| --- | --- |
|  |  |

**Fig. 4.1.** Point Cloud plot (In Pro/Engineer)

4.2 Spliced 3-D Point cloud



**Fig. 4.2.** Point Cloud and RGB overlay plot from the backside of the figure

4.3 Point Cloud RGB Image

|  |  |
| --- | --- |
|  |  |
|  |  |

**Fig. 4.3.** 3-D color Point Cloud front/back side and spliced

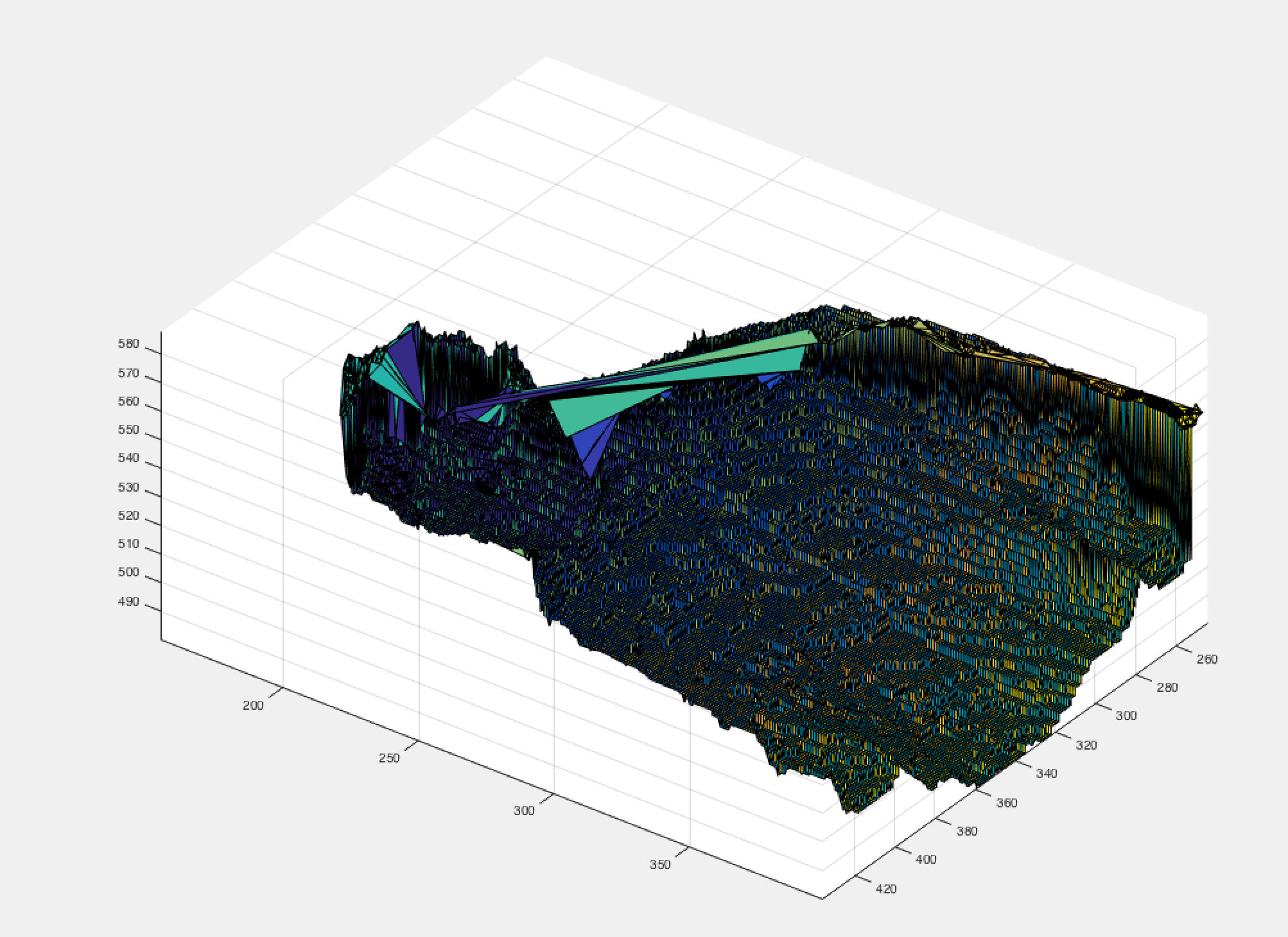
4.4 Triangulation with Delaunay

|  |  |
| --- | --- |
|  |  |
|  |  |

**Fig. 4.4.** Delaunay result of front/back point cloud

|  |  |
| --- | --- |
|  |  |
|  | |

**Fig. 4.5.** Delaunay Result of spliced color point cloud



**Fig. 4.6.** Delaunay Result of spliced point cloud

4.5 Triangulation with MyCrust

|  |  |
| --- | --- |
|  |  |

**Fig. 4.7.** MyCrust result of spliced 3-D point Cloud and details

4.6 Final Result

|  |  |
| --- | --- |
|  |  |
|  |  |

**Fig. 4.8.** Final result

5 Analysis

The keys of this project are collecting data from object, collage and triangulation. To collect the data of object, we find it best to choose somewhere open with less background. Because Kinect has a relative large detect depth, if you have too much background object in the range of 4000mm, it is difficult to extract foreground for post-processing, and the background left will eventually affect the triangulation process.

The approach of splice need a pre-configuration of parameter. At first you need to take several test for the corresponding point and do the collage to see whether it works well, after you assure the parameter, this process will allow a slight move of object, which means you do not have to stick in the same position. However, stay still at the configured position will bring in more accuracy.

The ordinary triangulation solution in MATLAB is delaunay function family, which could quickly triangulate the 2-D scattered points and build a surface based on it. In 3-D circumstances, we could sometimes use trisurf or trimesh to combine the 2-D triangulation result with the depth information to build a 3-D model. But after experiments, we find that this solution is not applicable to our circumstance. In our data, front and back are plotted on the same coordinates, therefore in 2-D triangulation, the two points on the same coordinates would be considered as one, which causes funny errors in the result. Even if we slide one of the point cloud a little bit to empty their intersection part, the triangulation would connect the front and back coordinates as a small triangle, which lead to front information leaking to the back result and vise versa. Therefore, we use the MyCrust algorithm here.

The triangulation result is limited by the unsatisfying resolution of the Kinect camera and the limited viewpoints. If you look at the front or the back surface of the object, everything works perfect, but if you look at the side, you will find the triangle is large and not precise. This is normal because we do not take any image in the side, and the side surface was correlated by the front and back point, which is less accurate than front and back. Furthermore, MyCrust works well for point cloud with larger density, which explained this situation.

6 Conclusions

In this project, we provide a low cost, portable and effective way for 3D reconstruction. It is applicable and handy even for non-professionals. The main process of our project is collecting data with Microsoft Kinect, extract foreground object point cloud, collage point cloud in different direction and triangulation. Our implementation is simple but the output is satisfactory. Future work may be done to solve the large size triangulation for point cloud with low density, and more adaptable algorithm for collage point cloud in different direction.

7 References

1. B. Curless and M. Levoy. “A volumetric method for building complex models from range images,” In Proceedings of the 23rd annual conference on Computer graphics and interactive techniques (SIGGRAPH). New York, NY, USA, pp.303-312, 1996.

2. Z. Zhang, “Digital Photogrammetry and Computer Vision,” Geomatics and Information Science of Wuhan University, issue 12, pp.1035-1039, 2004.

3. Nakagawa M. “Integrating high resolution air-borne linear CCD (TLS) imagery and LIDAR data,” 2nd GRSS/ISPRS Joint Workshop on Remote Sensing and Data Fusion over Urban Areas, pp. 236 – 240 2003.

4. M. Liao, Q. Zhang, H. Wang, R.Yang, M. Gong, “Modeling deformable objects from a single depth camera,” IEEE 12th International Conference on Computer Vision, pp.167-174, 2009.

5. C. Daniel Herrera, Juho Kannala, Janne Heikkila. Accurate and Practical Calibration of a Depth and Color Camera Pair[J]. Computer Analysis of Images and Patterns, vol.6855, pp.437-445, 2011.

6. K. Khoshelham. “Accuracy analysis of kinect depth data,” (ISPRS) http://www.isprs.org/proceedings/XXXVIII/5-W12/Papers/ls2011\_submission\_40.pdf, 2011.

7. P.J. Besl, H.D. McKay. “A Method for Registration of 3-D Shapes,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.14, issue.2, pp.239 - 256. 1992.

8. P. Henry, M. Krainin, E. Herbst, X. Ren, D. Fox. “RGB-D Mapping: Using Depth Cameras for Dense 3D Modeling of Indoor Environments,” International Symposium on Experimental Robotics. 2010.

9. Heikkila, J. “A four-step camera calibration procedure with implicit image correction,” In Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), pp.1106-1112, 1997.

10. C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In Proceedings of the International Conference on Computer Vision (ICCV), 1998.

11. Fischler, M. A. and Bolles, R. C. “Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography,” Communications of the ACM, vol.24, issue.6, pp.381–395. 1981.

Appendix: Source Code

1. Kinect v1.8 SDK sample code

2. extractfromBG.m

clear all;

clc;

load f\_colorpoint.txt %back point cloud with background

load b\_colorpoint.txt %front point cloud with background

% buffer for the front points

extracted\_f = zeros(1,6);

% buffer for the back points

extracted\_b = zeros(1,6);

% extract all the front body points

j = 1;

for i = 1:640\*480

if f\_colorpoint(i,3)>12000 && f\_colorpoint(i,3)<20000 ...

&& f\_colorpoint(i,1) > 50 && f\_colorpoint(i,1) < 400 ...

&& f\_colorpoint(i,2) > 200 && f\_colorpoint(i,2) < 500

extracted\_f(j,:) =f\_colorpoint(i,:);

j = j + 1;

end

end

% extract all the back body points

j = 1;

for i = 1:640\*480

if b\_colorpoint(i,3)>12000 && b\_colorpoint(i,3)<18000 ...

&& b\_colorpoint(i,1) > 0 && b\_colorpoint(i,1) < 400

extracted\_b(j,:) =b\_colorpoint(i,:);

j = j + 1;

end

end

% compress the Z axis

extracted\_f(:,3) = extracted\_f(:,3)/30;

extracted\_b(:,3) = extracted\_b(:,3)/30;

% slight move for triangulation

extracted\_f(:,2) = extracted\_f(:,2) + 0.001;

% get front point cloud

delete colorpoint\_f.txt;

save('colorpoint\_f.txt','extracted\_f','-ASCII','-append');

% get back point cloud

delete colorpoint\_b.txt;

save('colorpoint\_b.txt','extracted\_b','-ASCII','-append');

% 3-point alignment

extracted\_b(:,3) = 1000 - extracted\_b(:,3) + 35;

extracted\_b(:,1) = extracted\_b(:,1);

extracted\_b(:,2) = extracted\_b(:,2);

% get extracted point cloud

extracted = [extracted\_f;extracted\_b];

delete colorpoint.txt;

save('colorpoint.txt','extracted','-ASCII','-append');

3. TriangleRGBwithDepth.m

%get triangulation plot with RGB

clc

clear all

% triangulation for the back of body, de-comment it

% load colorpoint\_b.txt;

% myfile = colorpoint\_b;

% triangulation for the front of body, de-comment it

% load colorpoint\_f.txt;

% myfile = colorpoint\_f;

% triangulation for front and back of body

load colorpoint.txt

myfile = colorpoint;

%GET RGB Color and then ColorMap

RGB\_X = myfile(:,4);

RGB\_Y = myfile(:,5);

RGB\_Z = myfile(:,6);

RGB\_X=RGB\_X(:);

RGB\_Y=RGB\_Y(:);

RGB\_Z=RGB\_Z(:);

M=uint8([RGB\_X RGB\_Y RGB\_Z]);

C=double(M)/255;

colormap(C); %Run with this line gives the final result. Run without gives the triangulation result

%Delaunay

p = myfile(:,1:3);

%t=delaunay(p(:,1),p(:,2)); % Run with this line gives the Delaunay result

t = MyRobustCrust(p); % Run with this line gives the MyCrust Result

% plot the triangulation

trisurf(t,p(:,1),p(:,2),p(:,3),1:size(p,1),'edgecolor','none')

%trisurf(t,p(:,1),p(:,2),p(:,3),1:size(p,1),'facecolor','none')

axis equal;

shading interp;

**4. DUIQI.m**

load('point1.txt');

load('point2.txt');

b1=point1;

b2=point2;

save('b1.mat')

save('b2.mat')

figure(3);

hold on

axis equal

title('Points Cloud','fontsize',14)

plot3(b1(:,1),b1(:,2),b1(:,3),'g.')

figure(4);

hold on

axis equal

title('Points Cloud','fontsize',14)

plot3(b2(:,1),b2(:,2),b2(:,3),'g.')

P1=[1861.7 2203.9 2478

];

P2=[1858.8 2017 2485

];

P3=[2757.4 1860.6 2554

];

Q1=[1857 1947.1 2485

];

Q2=[1853.6 2132.6 2491

];

Q3=[2768.9 2206.4 2640

];

V1=P2-P1;

V11=P3-P1;

W1=Q2-Q1;

W11=Q3-Q1;

V3=cross(V1,V11);

W3=cross(W1,W11);

V2=cross(V3,V1);

W2=cross(W3,W1);

v1=V1/norm(V1);

v2=V2/norm(V2);

v3=V3/norm(V3);

w1=W1/norm(W1);

w2=W2/norm(W2);

w3=W3/norm(W3);

v=[v1

v2

v3];

w=[w1

w2

w3];

R=inv(v)\*w;

S=Q1-P1\*R;

load('point1.txt')

[m,n]=size(point1);

T=repmat(S,m,1)

P=point1\*R+T;

save('P.mat')

p=[P;point2];

save('p.mat')

figure(5);

hold on

axis equal

title('Points Cloud','fontsize',14)

plot3(p(:,1),p(:,2),p(:,3),'g.')

5. MyRobustCrust.m

Open-Source Code by Giaccari Luigi from Microsoft

%% MyRobustCrust

%

%Simple surface recostruction program based on Crust algorithm Given a set

% of 3D points returns a triangulated tight surface.

%

%%Version of my Crust that performs the manifold extraction in order to have

%a regular surface. This algorithm uses the ball pivoting method to extract

%the manifold and return outwards normals orientation.

%

%The more points there are the best the surface will be fitted, although

% you will have to wait more. For very large models an help memory errors

% may occurs. It is important even the point distribution, generally

% uniformly distributed points with denser zones in high curvature features

% give the best results.

%

% Remember crust algorithom needs a cloud representing a VOLUME so open

% surface may give inaccurate results. Surface with small holes are not

% considered open surface and generally are processed well.

%

%

% If any problems occurs in execution, or if you found a bug, have a

% suggestion or question just contact me at:

%

% giaccariluigi@msn.com

%

%

%

%

%Syntax:

%

%[t,tnorm]=MyCrust(p);

%

% figure(1)

% hold on title('Output Triangulation','fontsize',14) axis equal

% trisurf(t,p(:,1),p(:,2),p(:,3),'facecolor','c','edgecolor','b')

%

%Input:

% p is a Nx3 array containing the 3D set of points

%Output:

% t are points id contained in triangles, nx3 array .

%

% tnorm outwards normals of triangles, nx3 array .

%

% See also qhull, voronoin, convhulln, delaunay, delaunay3, tetramesh.

%

%Author:Giaccari Luigi

%Last Update: 03/1/2008

%Created: 10/10/2008

%

%Visit:

%

% http://giaccariluigi.altervista.org/blog/

%

%This work is free thanks to users gratitude, if you find it usefull

%consider making a donatin on my website. Thank you

function [t,tnorm]=MyRobustCrust(p)

%error check

if nargin>1

error('The only input must be the Nx3 array of points');

end

[n]=size(p,2);

if n ~=3

error('Input 3D points must be stored in a Nx3 array');

end

clear n

%% Main

starttime=clock;

%add points to the given ones, this is usefull

%to create outside tetraedroms

tic

[p,nshield]=AddShield(p);

fprintf('Added Shield: %4.4f s\n',toc)

tic

tetr=delaunayn(p);%creating tedraedron

tetr=int32(tetr);%use integer to save memory

fprintf('Delaunay Triangulation Time: %4.4f s\n',toc)

%connectivity data

%find triangles to tetraedrom and tetraedrom to triangles connectivity data

tic

[t2tetr,tetr2t,t]=Connectivity(tetr);

fprintf('Connectivity Time: %4.4f s\n',toc)

tic

[cc,r]=CC(p,tetr);%Circumcenters of tetraedroms

fprintf('Circumcenters Time: %4.4f s\n',toc)

clear n

tic

[tbound,Ifact]=Marking(p,tetr,tetr2t,t2tetr,cc,r,nshield);%Flagging tetraedroms as inside or outside

fprintf('Walking Time: %4.4f s\n',toc)

%recostructed raw surface

t=t(tbound,:);

% Ifact=Ifact(tbound);

clear tetr tetr2t t2tetr

%m

tic

[t,tnorm]=ManifoldExtraction(t,p);

fprintf('Manifold extraction Time: %4.4f s\n',toc)

time=etime(clock,starttime);

fprintf('Total Time: %4.4f s\n',time)

end

%% Circumcenters

function [cc,r]=CC(p,tetr)

%finds circumcenters from a set of tetraedroms

%batch the code to save memory

ntetr=size(tetr,1);

cutsize=25000;

i1=1;i2=cutsize;

r=zeros(ntetr,1);

cc=zeros(ntetr,1);

if i2>ntetr

i2=ntetr;%to trigeer the terminate criterion

end

while 1

%points of tetraedrom

p1=(p(tetr(i1:i2,1),:));

p2=(p(tetr(i1:i2,2),:));

p3=(p(tetr(i1:i2,3),:));

p4=(p(tetr(i1:i2,4),:));

%vectors of tetraedrom edges

v21=p1-p2;

v31=p3-p1;

v41=p4-p1;

%Solve the system using cramer method

d1=sum(v41.\*(p1+p4)\*.5,2);

d2=sum(v21.\*(p1+p2)\*.5,2);

d3=sum(v31.\*(p1+p3)\*.5,2);

det23=(v21(:,2).\*v31(:,3))-(v21(:,3).\*v31(:,2));

det13=(v21(:,3).\*v31(:,1))-(v21(:,1).\*v31(:,3));

det12=(v21(:,1).\*v31(:,2))-(v21(:,2).\*v31(:,1));

Det=v41(:,1).\*det23+v41(:,2).\*det13+v41(:,3).\*det12;

detx=d1.\*det23+...

v41(:,2).\*(-(d2.\*v31(:,3))+(v21(:,3).\*d3))+...

v41(:,3).\*((d2.\*v31(:,2))-(v21(:,2).\*d3));

dety=v41(:,1).\*((d2.\*v31(:,3))-(v21(:,3).\*d3))+...

d1.\*det13+...

v41(:,3).\*((d3.\*v21(:,1))-(v31(:,1).\*d2));

detz=v41(:,1).\*((v21(:,2).\*d3)-(d2.\*v31(:,2)))...

+v41(:,2).\*(d2.\*v31(:,1)-v21(:,1).\*d3)...

+d1.\*(det12);

%Circumcenters

cc(i1:i2,1)=detx./Det;

cc(i1:i2,2)=dety./Det;

cc(i1:i2,3)=detz./Det;

%Circumradius

r(i1:i2)=realsqrt((sum((p2-cc(i1:i2,:)).^2,2)));%ciecum radius

if i2==ntetr

break;%terminate criterion

end

i1=i1+cutsize;

i2=i2+cutsize;

if i2>ntetr

i2=ntetr;%to trigeer the terminate criterion

end

end

end

%% Connectivity

function [t2tetr,tetr2t,t]=Connectivity(tetr)

%Gets conectivity relantionships among tetraedroms

numt = size(tetr,1);

vect = 1:numt;

t = [tetr(:,[1,2,3]); tetr(:,[2,3,4]); tetr(:,[1,3,4]);tetr(:,[1,2,4])];%triangles not unique

[t,j,j] = unique(sort(t,2),'rows');%triangles

t2tetr = [j(vect), j(vect+numt), j(vect+2\*numt),j(vect+3\*numt)];%each tetraedrom has 4 triangles

% triang-to-tetr connectivity

nume = size(t,1);

tetr2t = zeros(nume,2,'int32');

count= ones(nume,1,'int8');

for k = 1:numt

for j=1:4

ce = t2tetr(k,j);

tetr2t(ce,count(ce)) = k;

count(ce)=count(ce)+1;

end

end

end % connectivity()

%% Marking

function [tbound,Ifact]=Marking(p,tetr,tetr2t,t2tetr,cc,r,nshield)

%The more important routine to flag tetredroms as outside or inside

%costants for the algorithm

TOLLDIFF=.01;%tollerance decrease at each iteration

% (the higher the value the more robust but slower is the algorithm. It is also required

% a higher MAXLEVEL value to rach the end of iterations. );

INITTOLL=.99;%starting tollerance

MAXLEVEL=10/TOLLDIFF;%maximum reachable level

BRUTELEVEL=MAXLEVEL-50;%level to start brute continuation

%preallocation

np=size(p,1)-nshield;%nshield = number of shield points put at the end of array

numtetr=size(tetr,1);

nt=size(tetr2t,1);

% deleted=true(numtetr,1);%deleted tetraedroms

% checked=false(numtetr,1);%checked tetraedroms

onfront=false(nt,1);%tetraedroms that need to be checked

% countchecked=0;%counter of checked tetraedroms

%First flag as outside tetraedroms with Shield points

%unvectorized

% for i=1:numtetr

% for j=1:4

% if tetr(i,j)>np;

% deleted(i)=true;

% checked(i)=true;

% onfront(t2tetr(i,:))=true;

% countchecked=countchecked+1;

% break

% end

% end

% end

%vectorized

deleted=any(tetr>np,2);%deleted tetraedroms

checked=deleted;%checked tetraedroms

onfront(t2tetr(checked,:))=true;

countchecked=sum(checked);%counter of checked tetraedroms

%tollerances to mark as in or out

toll=zeros(nt,1)+INITTOLL;

level=0;

%intersection factor

%it is computed from radius of the tetraedroms circumscribed sphere

% and the distance between their center

Ifact=IntersectionFactor(tetr2t,cc,r);

clear cc r

% Now we scan all tetraedroms. When one is scanned puts on front is

% neighbor. This means that now the neighobor can be checked too.

% At the begining only tetraedroms with shield points are on front,

% because we are sure the are out. Tetraedrom with high

% intersection factor will be marked as equal else different. When

% I say high i mean under a set tollerance that becames lower as

% the algorithm progresses. This Aims to avoid errors propagation

% when a tetraedrom is wrong marked.

%

ids=1:nt;

queue=ids(onfront);

nt=length(queue);

while countchecked<numtetr && level<MAXLEVEL

level=level+1;%level of scan reached

for i=1:nt%loop trough triangles <-----better is check only unchecked

id=queue(i);

tetr1=tetr2t(id,1);tetr2=tetr2t(id,2);%tetraedroms linked to triangle under analysis

if tetr2==0 %do not check boundary triangles

onfront(id)=false;

continue

elseif (checked(tetr1) && checked(tetr2)) %tetraedroms are already checked

onfront(id)=false;

continue

end

if Ifact(id)>=toll(id) %flag as equal

if checked(tetr1)%find the checked one between the two

deleted(tetr2)=deleted(tetr1) ;%flag as equal

checked(tetr2)=true;%check

countchecked=countchecked+1;

onfront(t2tetr(tetr2,:))=true;%put on front all tetreadrom triangles

else

deleted(tetr1)=deleted(tetr2) ;%flag as equal

checked(tetr1)=true;%check

countchecked=countchecked+1;

onfront(t2tetr(tetr1,:))=true;%put on front all tetreadrom triangles

end

onfront(id)=false;%remove from front

elseif Ifact(id)<-toll(id)%flag as different

if checked(tetr1)%find the checked one between the two

deleted(tetr2)=~(deleted(tetr1)) ;%flag as different

checked(tetr2)=true;%check

countchecked=countchecked+1;

onfront(t2tetr(tetr2,:))=true;%put on front all tetreadrom triangles

else

deleted(tetr1)=~(deleted(tetr2)) ;%flag as different

checked(tetr1)=true;%check

countchecked=countchecked+1;

onfront(t2tetr(tetr1,:))=true;%put on front all tetreadrom triangles

end

onfront(id)=false;%remove from front

else

toll(id)=toll(id)-TOLLDIFF;%tolleraces were too high next time will be lower

end

end

if level==BRUTELEVEL %brute continuation(this may appens when there are almost null volume tetraedroms)

beep

warning('Brute continuation necessary')

onfront(t2tetr(~(checked),:))=true;%force onfront collocation

end

%update the queue

queue=ids(onfront);

nt=length(queue);

end

%extract boundary triangles

tbound=BoundTriangles(tetr2t,deleted);

% this is the raw surface and needsimprovements to be used in CAD systems.

% Maybe in my next revision I will add surface post treatments. Anyway for

% grafical purpose this should be good.

%Output Data

numchecked=countchecked/numtetr;

if level==MAXLEVEL

warning([num2str(level),' th level was reached\n'])

else

fprintf('%4.0f th level was reached\n',level)

end

fprintf('%4.4f %% of Tetraedroms were checked\n',numchecked\*100)

end

%% AddShield

function [pnew,nshield]=AddShield(p)

%adds outside points to the given cloud forming outside tetraedroms

%shield points are very good in detectinf outside tetraedroms. Unfortunatly

%delunany triangulation with these points can be even of 50% slower.

%find the bounding box

maxx=max(p(:,1));

maxy=max(p(:,2));

maxz=max(p(:,3));

minx=min(p(:,1));

miny=min(p(:,2));

minz=min(p(:,3));

%give offset to the bounding box

step=max(abs([maxx-minx,maxy-miny,maxz-minz]));

maxx=maxx+step;

maxy=maxy+step;

maxz=maxz+step;

minx=minx-step;

miny=miny-step;

minz=minz-step;

N=10;%number of points of the shield edge

step=step/(N\*N);%decrease step, avoids not unique points

nshield=N\*N\*6;

%creating a grid lying on the bounding box

vx=linspace(minx,maxx,N);

vy=linspace(miny,maxy,N);

vz=linspace(minz,maxz,N);

[x,y]=meshgrid(vx,vy);

facez1=[x(:),y(:),ones(N\*N,1)\*maxz];

facez2=[x(:),y(:),ones(N\*N,1)\*minz];

[x,y]=meshgrid(vy,vz-step);

facex1=[ones(N\*N,1)\*maxx,x(:),y(:)];

facex2=[ones(N\*N,1)\*minx,x(:),y(:)];

[x,y]=meshgrid(vx-step,vz);

facey1=[x(:),ones(N\*N,1)\*maxy,y(:)];

facey2=[x(:),ones(N\*N,1)\*miny,y(:)];

%add points to the p array

pnew=[p;

facex1;

facex2;

facey1;

facey2;

facez1;

facez2];

% figure(4)

% plot3(pnew(:,1),pnew(:,2),pnew(:,3),'.g')

end

%% BoundTriangles

function tbound=BoundTriangles(tetr2t,deleted)

%extracts boundary triangles from a set tetr2t connectivity and form the

%deleted vector which tells tetraedroms that are marked as out

nt=size(tetr2t,1);%number of totals triangles

tbound=true(nt,2);%inizilize to keep shape in next operation

ind=tetr2t>0;%avoid null index

tbound(ind)=deleted(tetr2t(ind));%mark 1 for deleted 0 for kept tetraedroms

tbound=sum(tbound,2)==1;%bounary triangles only have one tetraedrom

end

%% Intersection factor

function Ifact=IntersectionFactor(tetr2t,cc,r)

nt=size(tetr2t,1);

Ifact=zeros(nt,1);%intersection factor

%it is computed from radius of the tetraedroms circumscribed sphere

% and the distance between their center

i=tetr2t(:,2)>0;

distcc=sum((cc(tetr2t(i,1),:)-cc(tetr2t(i,2),:)).^2,2);%distance between circumcenters

Ifact(i)=(-distcc+r(tetr2t(i,1)).^2+r(tetr2t(i,2)).^2)./(2\*r(tetr2t(i,1)).\*r(tetr2t(i,2)));

%unvectorized

% for i=1:nt

% if tetr2t(i,2)>0 %jump boundary tetraedrom

% distcc=sum((cc(tetr2t(i,1),:)-cc(tetr2t(i,2),:)).^2,2);%distance between circumcenters

% %intersection factor

% Ifact(i)=(-distcc+r(tetr2t(i,1))^2+r(tetr2t(i,2))^2)/(2\*r(tetr2t(i,1))\*r(tetr2t(i,2)));

% end

% end

end

%% Manifold Extraction

function [t,tnorm]=ManifoldExtraction(t,p)

%Given a set of trianlges,

%Buils a manifolds surface with the ball pivoting method.

% building the etmap

numt = size(t,1);

vect = 1:numt; % Triangle indices

e = [t(:,[1,2]); t(:,[2,3]); t(:,[3,1])]; % Edges - not unique

[e,j,j] = unique(sort(e,2),'rows'); % Unique edges

te = [j(vect), j(vect+numt), j(vect+2\*numt)];

nume = size(e,1);

e2t = zeros(nume,2,'int32');

clear vect j

ne=size(e,1);

np=size(p,1);

count=zeros(ne,1,'int32');%numero di triangoli candidati per edge

etmapc=zeros(ne,4,'int32');

for i=1:numt

i1=te(i,1);

i2=te(i,2);

i3=te(i,3);

etmapc(i1,1+count(i1))=i;

etmapc(i2,1+count(i2))=i;

etmapc(i3,1+count(i3))=i;

count(i1)=count(i1)+1;

count(i2)=count(i2)+1;

count(i3)=count(i3)+1;

end

etmap=cell(ne,1);

for i=1:ne

etmap{i,1}=etmapc(i,1:count(i));

end

clear etmapc

tkeep=false(numt,1);%all'inizio nessun trinagolo selezionato

%Start the front

%building the queue to store edges on front that need to be studied

efront=zeros(nume,1,'int32');%exstimate length of the queue

%Intilize the front

tnorm=Tnorm(p,t);%get traingles normals

%find the highest triangle

[foo,t1]=max( (p(t(:,1),3)+p(t(:,2),3)+p(t(:,3),3))/3);

if tnorm(t1,3)<0

tnorm(t1,:)=-tnorm(t1,:);%punta verso l'alto

end

%aggiungere il ray tracing per verificare se il triangolo punta

%veramente in alto.

%Gli altri triangoli possono essere trovati sapendo che se un

%triangolo ha il baricentro pi˘ alto sicuramente contiene il punto

%pi˘ alto. Vanno analizzati tutto i traingoli contenenti questo

%punto

tkeep(t1)=true;%primo triangolo selezionato

efront(1:3)=te(t1,1:3);

e2t(te(t1,1:3),1)=t1;

nf=3;%efront iterato

while nf>0

k=efront(nf);%id edge on front

if e2t(k,2)>0 || e2t(k,1)<1 || count(k)<2 %edge is no more on front or it has no candidates triangles

nf=nf-1;

continue %skip

end

%candidate triangles

idtcandidate=etmap{k,1};

t1=e2t(k,1);%triangle we come from

%get data structure

% p1

% / | \

% t1 p3 e1 p4 t2(idt)

% \ | /

% p2

alphamin=inf;%inizilizza

ttemp=t(t1,:);

etemp=e(k,:);

p1=etemp(1);

p2=etemp(2);

p3=ttemp(ttemp~=p1 & ttemp~=p2);%terzo id punto

%plot for debug purpose

% close all

% figure(1)

% axis equal

% hold on

%

% fs=100;

%

% cc1=(p(t(t1,1),:)+p(t(t1,2),:)+p(t(t1,3),:))/3;

%

% trisurf(t(t1,:),p(:,1),p(:,2),p(:,3))

% quiver3(cc1(1),cc1(2),cc1(3),tnorm(t1,1)/fs,tnorm(t1,2)/fs,tnorm(t1,3)/fs,'b');

%

for i=1:length(idtcandidate)

t2=idtcandidate(i);

if t2==t1;continue;end;

%debug

% cc2=(p(t(t2,1),:)+p(t(t2,2),:)+p(t(t2,3),:))/3;

%

% trisurf(t(t2,:),p(:,1),p(:,2),p(:,3))

% quiver3(cc2(1),cc2(2),cc2(3),tnorm(t2,1)/fs,tnorm(t2,2)/fs,tnorm(t2,3)/fs,'r');

%

%

ttemp=t(t2,:);

p4=ttemp(ttemp~=p1 & ttemp~=p2);%terzo id punto

%calcola l'angolo fra i triangoli e prendi il minimo

[alpha,tnorm2]=TriAngle(p(p1,:),p(p2,:),p(p3,:),p(p4,:),tnorm(t1,:));

if alpha<alphamin

alphamin=alpha;

idt=t2;

tnorm(t2,:)=tnorm2;%ripristina orientazione

%debug

% quiver3(cc2(1),cc2(2),cc2(3),tnorm(t2,1)/fs,tnorm(t2,2)/fs,tnorm(t2,3)/fs,'c');

end

%in futuro considerare di scartare i trianoli con angoli troppi bassi che

%possono essere degeneri

end

%update front according to idttriangle

tkeep(idt)=true;

for j=1:3

ide=te(idt,j);

if e2t(ide,1)<1% %Is it the first triangle for the current edge?

efront(nf)=ide;

nf=nf+1;

e2t(ide,1)=idt;

else %no, it is the second one

efront(nf)=ide;

nf=nf+1;

e2t(ide,2)=idt;

end

end

nf=nf-1;%per evitare di scappare avanti nella coda e trovare uno zero

end

t=t(tkeep,:);

tnorm=tnorm(tkeep,:);

end

%% TriAngle

function [alpha,tnorm2]=TriAngle(p1,p2,p3,p4,planenorm)

%per prima cosa vediamo se il p4 sta sopra o sotto il piano identificato

%dalla normale planenorm e il punto p3

test=sum(planenorm.\*p4-planenorm.\*p3);

%Computes angle between two triangles

v21=p1-p2;

v31=p3-p1;

tnorm1(1)=v21(2)\*v31(3)-v21(3)\*v31(2);%normali ai triangoli

tnorm1(2)=v21(3)\*v31(1)-v21(1)\*v31(3);

tnorm1(3)=v21(1)\*v31(2)-v21(2)\*v31(1);

tnorm1=tnorm1./norm(tnorm1);

v41=p4-p1;

tnorm2(1)=v21(2)\*v41(3)-v21(3)\*v41(2);%normali ai triangoli

tnorm2(2)=v21(3)\*v41(1)-v21(1)\*v41(3);

tnorm2(3)=v21(1)\*v41(2)-v21(2)\*v41(1);

tnorm2=tnorm2./norm(tnorm2);

alpha=tnorm1\*tnorm2';%coseno dell'angolo

%il coseno considera l'angolo fra i sempipiani e non i traigoli, ci dice

%che i piani sono a 180 se alpha=-1 sono concordi se alpha=1, a 90∞

alpha=acos(alpha);%trova l'angolo

%Se p4 sta sopra il piano l'angolo Ë quello giusto altrimenti va maggiorato

%di 2\*(180-alpha);

if test<0%p4 sta sotto maggioriamo

alpha=alpha+2\*(pi-alpha);

end

% fs=100;

% cc2=(p1+p2+p3)/3;

% quiver3(cc2(1),cc2(2),cc2(3),tnorm1(1)/fs,tnorm1(2)/fs,tnorm1(3)/fs,'m');

% cc2=(p1+p2+p4)/3;

% quiver3(cc2(1),cc2(2),cc2(3),tnorm2(1)/fs,tnorm2(2)/fs,tnorm2(3)/fs,'m');

%vediamo se dobbiamo cambiare l'orientazione del secondo triangolo

%per come le abbiamo calcolate ora tnorm1 t tnorm2 non rispettano

%l'orientamento

testor=sum(planenorm.\*tnorm1);

if testor>0

tnorm2=-tnorm2;

end

end

%% Tnorm

function tnorm1=Tnorm(p,t)

%Computes normalized normals of triangles

v21=p(t(:,1),:)-p(t(:,2),:);

v31=p(t(:,3),:)-p(t(:,1),:);

tnorm1(:,1)=v21(:,2).\*v31(:,3)-v21(:,3).\*v31(:,2);%normali ai triangoli

tnorm1(:,2)=v21(:,3).\*v31(:,1)-v21(:,1).\*v31(:,3);

tnorm1(:,3)=v21(:,1).\*v31(:,2)-v21(:,2).\*v31(:,1);

L=sqrt(sum(tnorm1.^2,2));

tnorm1(:,1)=tnorm1(:,1)./L;

tnorm1(:,2)=tnorm1(:,2)./L;

tnorm1(:,3)=tnorm1(:,3)./L;

end