CS117 Final Report

Python implementation program of image-based scan to 3d model

Developed by Kai Yan #24684490

Project Overview:

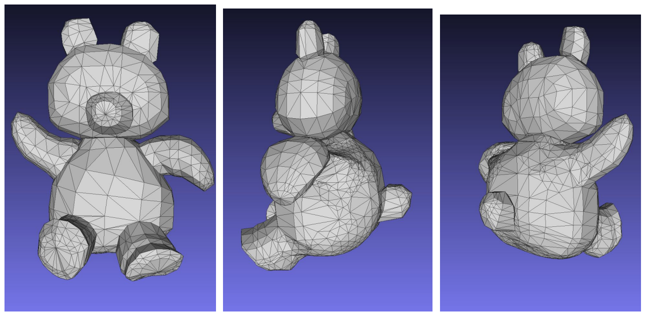
The program will perform features for reverse engineer focusing on transfer the 2D screenshot of either screen shot or real-life picture into 3d meshed models. The idea of the program is to do reverse engineer for object model and use different camera position for scanning and reconstruct the origin of the 3d model for the 2d image object. The goal of the program is tried to solve the problem for using a set of image to create a great quality 3d model in a small time complexity. The major problems to solve in this program are:

- Camera object: camera.py calibration.py
- Data visualization: uiconstruction.py pltload.py
- Image converting format and grayscale: purify.py
- Image reading and writing: load.py imagearray.py
- Image corner detection: ShiTomasi.py HarrisCorner.py
- Image denoise: imgDenoise.py
- Images points matching: SIFT.py
- Image stereo: stereo.py
- Images points distance estimation: distance_to_camera.py
- Point cloud generation for 2 images: threeDpoint.py
- Point cloud combining: objectmaker.py
- Obj file handling: readobj.py writeobj.py clearobj.py
- Delaunay meshing and edge optimization: Delaunay.py
- Filtering mesh cleaning using original image: readmask.py clearmesh.py
- Layer subdivision: laypoints.py
- Convex Hull algorithm: convexhull.py
- Remeshing and Smoothing: Using Maya and Meshlab

Data Sets:

Input:

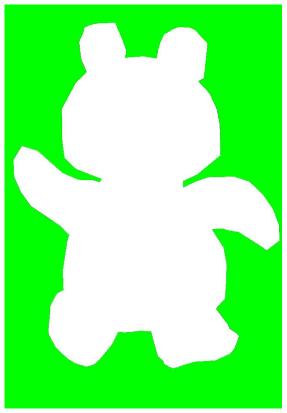
An image set that take on the Object. In the demo dataset, the program use a set of screenshot with different angle to a 3d model. The Image format is .png with 650*950 pixels located in Folder called ./teddydata



intermediate input:

Mask.jpg The Image format is .png with 650*950 pixels. This image is the input for the function to cleaning the mesh and noise in clearmesh.py

The green part of the image has a color of (0,255,0) which is easy to identify and distinguish the object and background



Algorithms:

Harris Corner Detection: extract corners and infer features of an image.

The Mathematic used is $[u\ v]\left(\sum\begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix}\right)\begin{bmatrix} u \\ v \end{bmatrix}$

Shi Tomasi: $R = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$ Also extract corners and infer features of an image. The project used both for finding N strongest corners without missing essential points.

SIFT: uses Difference of Gaussians to find feature matching point. (Referenced from 2004, D.Lowe, University of British Columbia) The function is implemented by using OpenCV.

Point Cloud location Identifier:
The single surface point cloud will rotate and transform to the correction position of the final point cloud by using the following rotation algorithm

```
for i in arrdata: #different for different axis
                x = i[2]*math.sin(j/id*math.pi) + i[0]*math.cos(j/idmath.pi)+w*math.cos(j/id*math.pi+w/4*math.pi)+w
                y = i[1]
                z = i[2]*math.cos(j/id*math.pi) - i[0]*math.sin(j/id*math.pi)+w*math.sin(j/id*math.pi)
                rotatedata.append([x, y, z])
```

Delaunay triangulation: An computational geometry method with given a set of points and triangulation make no point in the points set inside the circumcircle of any triangle. The program implement the DT with 3 dimention and make sure no point in P inside the circum-hypersphere. The time complexity is O(nlogn) However, DT can cause long edges and have to be filtered.

```
for (each triangle)
  {
    if (vertex is inside triangle's circumscribed circle)
    {
      store triangle's edges in edgebuffer
      remove triangle
    }
  }
  remove all double edges from edgebuffer,
    keeping only unique ones
  for (each edge in edgebuffer)
  {
    form a new triangle between edge and vertex
  }
```

Close Vertices merging: Remove the close Vertices and clean the point cloud. The algorithm used Voronoi graph to find the closest point

Edge Length Filter: Remove the faces in mesh that contain excessed length of edges.

```
        3d points distance: math.sqrt((p1[0]-p2[0])**2+(p1[1]-p2[1])**2+(p1[2]-p2[2])**2)
        identify: if sqdist(np.array(vdata[i[0]]), np.array(vdata[i[1]])) > 100 or
sqdist(np.array(vdata[i[0]]), np.array(vdata[i[2]])) > 100 or sqdist(np.array(vdata[i[1]]),
np.array(vdata[i[2]])) > 100:
      deletelist.append(count)
```

Box Filter/Liang-Barsky line clipping algorithm: For detecting if two vertices has an edge that intersect or overlap an area. If there are intersection, remove the face contains the edge.

```
t = [-1,-1,-1,-1]
dx = x2-x1
dy = y2-y1
p = [-1,-1,-1,-1]
q = [-1,-1,-1,-1]
p[0]=-dx
p[1]=dx
p[2]=-dy
p[3]=dy
q[0]=x1-box[0]
q[1]=box[2]-x1
q[2]=y1-box[1]
q[3]=box[3]-y1
for i in range(0,4):
    if p[i] != 0:
        t[i]=q[i]/p[i]
    else:
        if p[i] != 0:
            t[i]=q[i]/p[i]
        else:
            if p[i]==0 and q[i] < 0:
                return False
            else:
                if p[i]==0 and q[i] >= 0:
                    return False
```

```
if t[0] > t[2]:
    t1=t[0]
else:
    t1=t[2]

if t[1] < t[3]:
    t2=t[1]
else:
    t2=t[3]
if t1 < t2:
    return True
else:
    return False
```

Masking Filter: visit each pixel or range in the masking image and use the Box filter to check if any edge or vertice pass through the area. If so, remove the face contain the vertice.

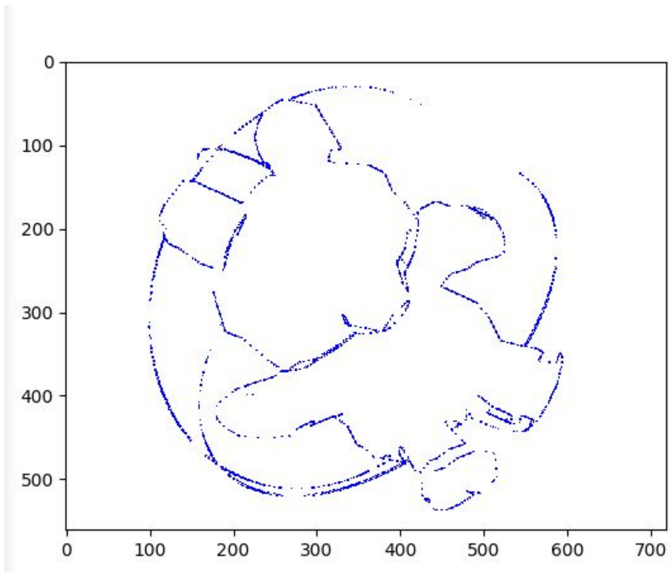
Layer converting:

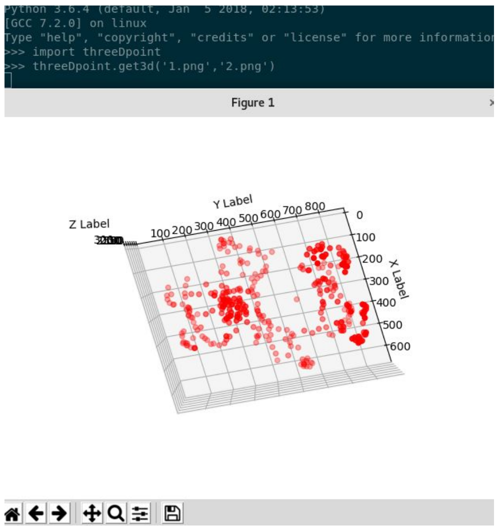
Convex Hull: the convex hull or convex envelope or convex closure of a set X of points in the Euclidean plane or in a Euclidean space is the smallest convex set that contains X. It is written in O(nlogn) algorithm to delete the internal points on the point cloud, and mesh, which allow the program start the surface meshing:

```
while far_point is not start:
        # get the first point (initial max) to use to compare with others
        p1 = None
        for p in points:
            if p is point:
                continue
            else:
                p1 = p
                break
        far_point = p1
        for p2 in points:
            # ensure we aren't comparing to self or pivot point
            if p2 is point or p2 is p1:
                continue
            else:
                direction = self._get_orientation(point, far_point, p2)
                if direction > 0:
                    far_point = p2
```

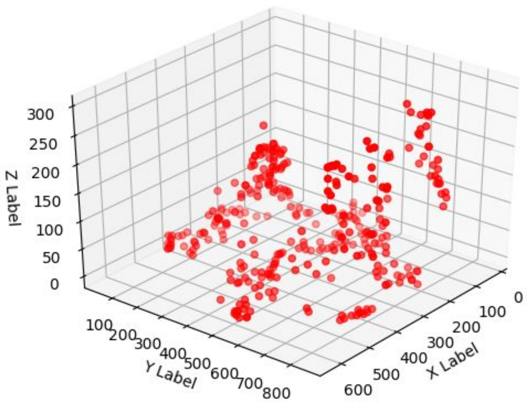
Results:

Line detection and distance measure for two image:

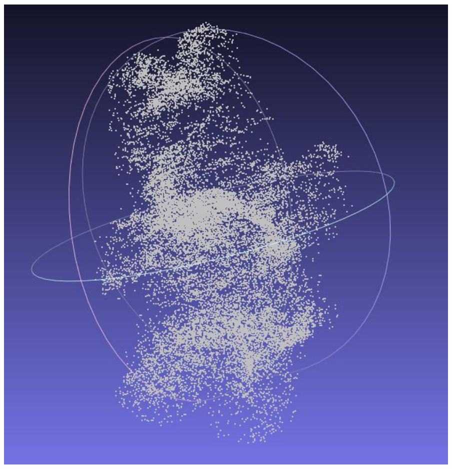
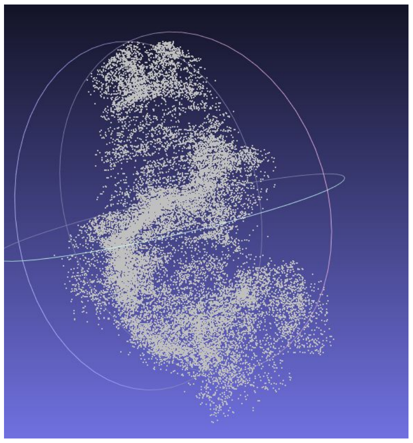
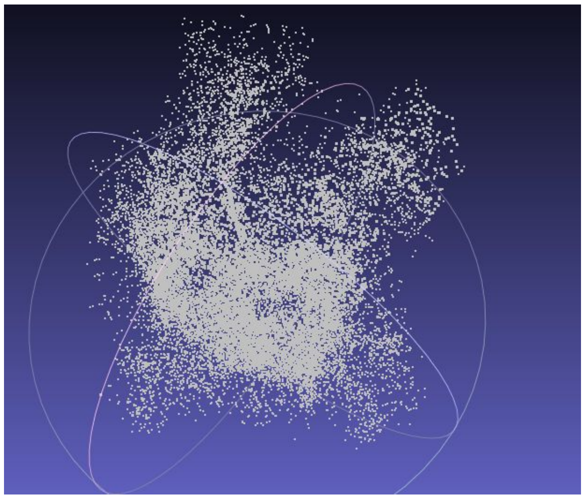
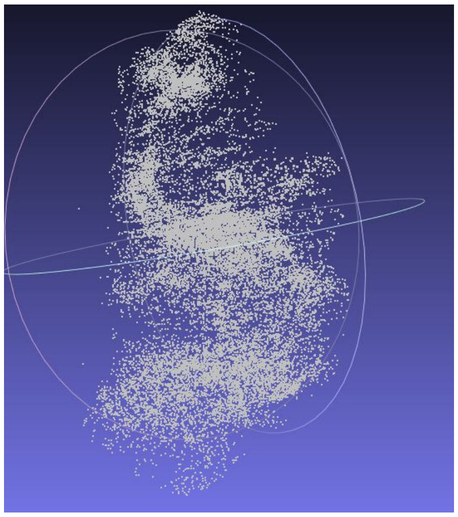




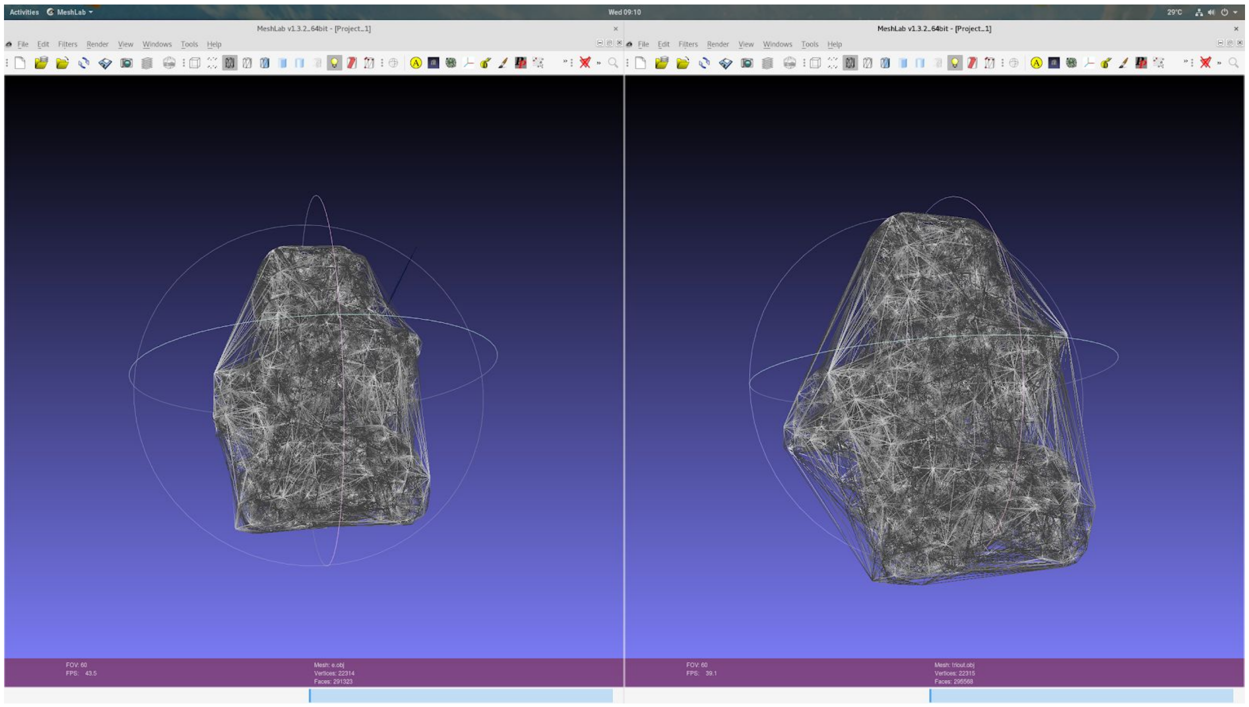
```
Python 3.6.4 (default, Jan  5 2018, 02:13:53)
[GCC 7.2.0] on linux
Type "help", "copyright", "credits" or "license" for more informatio
>>> import threeDpoint
>>> threeDpoint.get3d('1.png','2.png')
```



Point Cloud after scan the rest of 72 pictures:

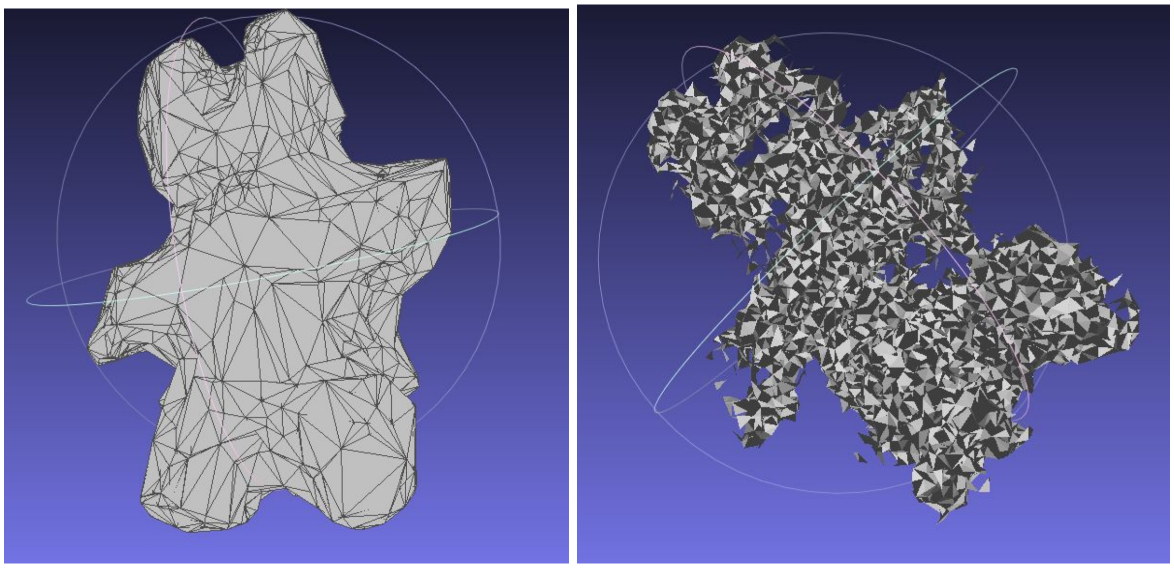
Boundary Filter using the mask picture


Activities
MeshLab
Wed 09:10
MeshLab v1.3.2_64bit - [Project_1]
File Edit Filters Render View Windows Tools Help
MeshLab v1.3.2_64bit - [Project_1]
File Edit Filters Render View Windows Tools Help

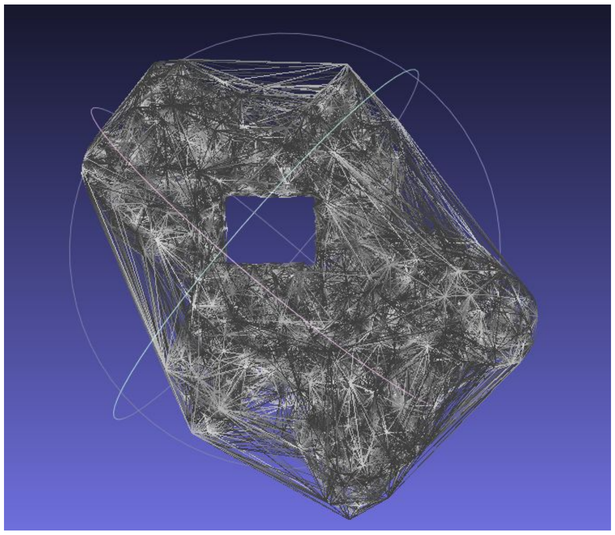
FOV: 60
FPS: 43.5
Mesh: e.obj
Vertices: 22314
Faces: 291323

FOV: 60
FPS: 39.1
Mesh: triout.obj
Vertices: 22315
Faces: 295568

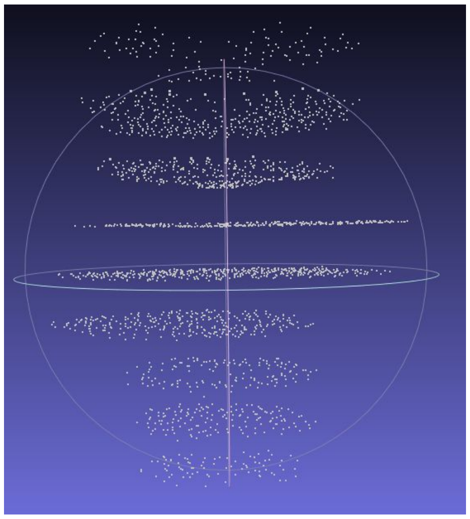
After Removing long edges with a edge removal function in Delaunay.py:



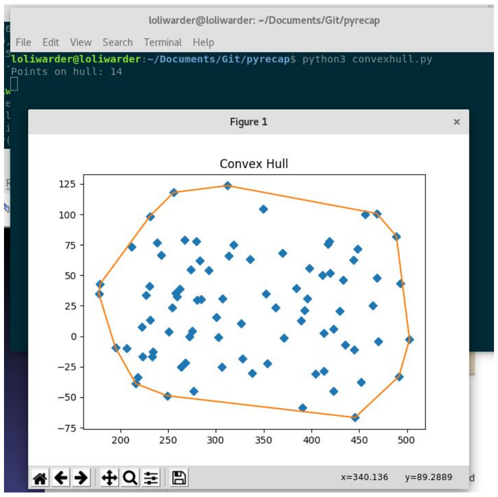
This is a more dramatic demonstration of the box filtering function(Liang-Barsky) can cause on the mesh:



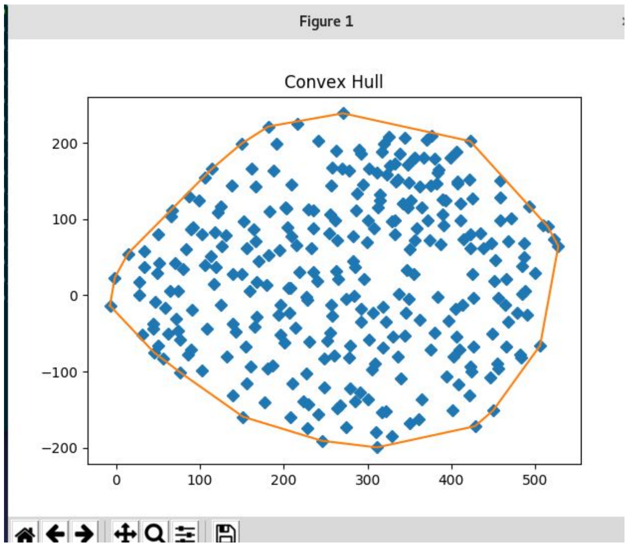
Finalize the program meshing and optimization the point: (Note that the sample use more severe value to create a obvious visual difference. In real production, there will be more layers can harder to see the difference obviously)

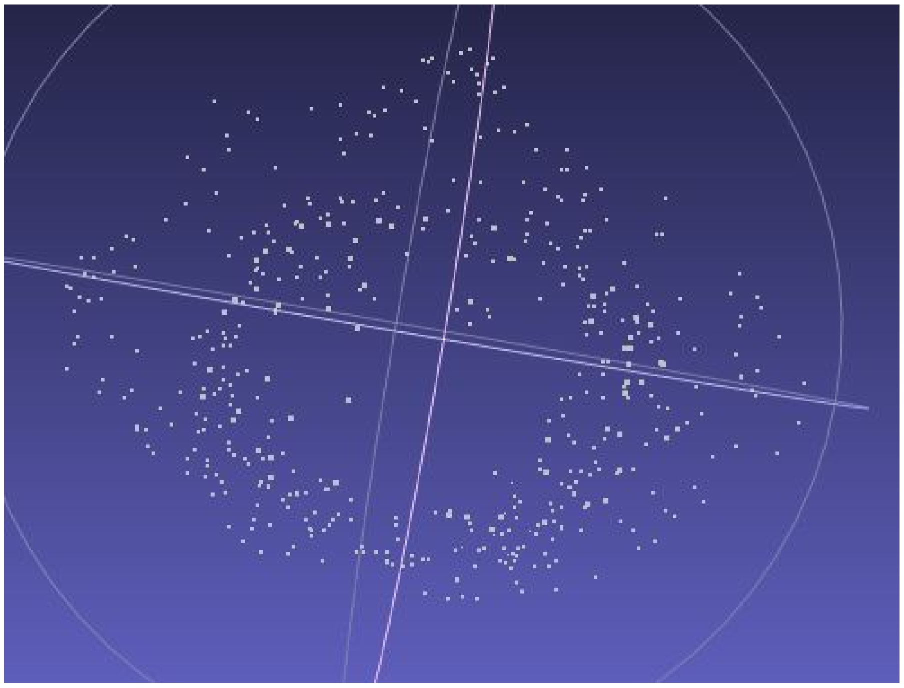


The internal point cloud that cont construct the surface can be removed using convexhull.py:

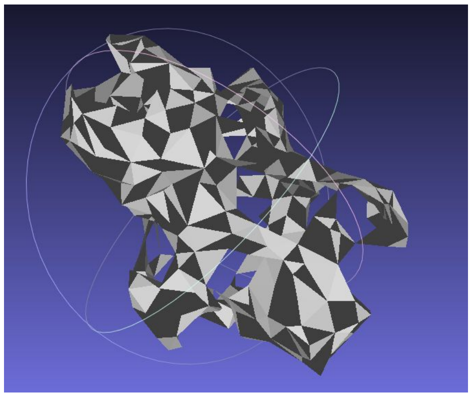
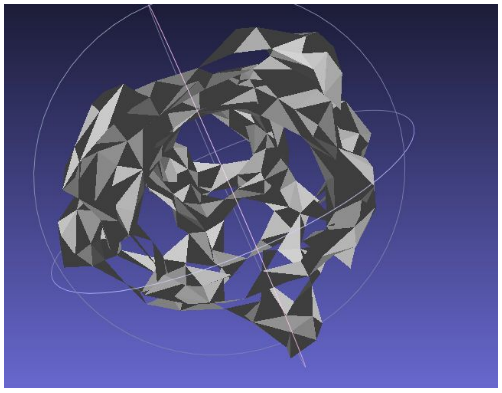
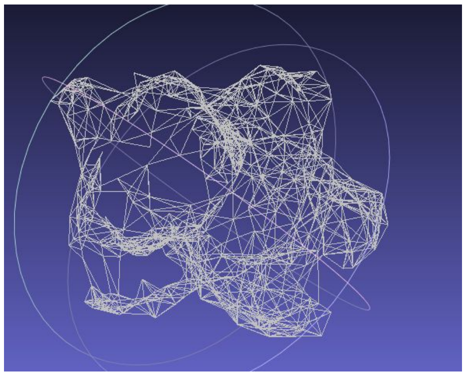

loliwarder@loliwarder: ~/Documents/Git/pyrecap
File Edit View Search Terminal Help
Figure 1
Convex Hull
x=340.136 y=89.2889

```
loliwarder@loliwarder:~/Documents/Git/pyrecap$ python3 convexhull.py
Points on hull: 14
```





Another meshing using DT can show now the mesh is a surface:

Assessment and Evaluation:

The majority of the algorithm used in the program take O(nlogn). The sample model has more than 300000 faces while each optimization and meshing can complete in less than ten seconds. Hence, the task is easy for computer to solve. The project successfully build the point cloud respect to the quantity of the image data. However, since there are more than 30000 vertices, the O(nlogn) time complexity on triangulation, mesh cleaning make sure the program can be complete in a less time. The masking technique also help the model to clean the noise and non-convex locations. Check the boundary of each image allow the program to identify non convex location can use the box fliter to delete the faces. The program worked on a low level programming that directly read and write the data of .obj file. The vertice stored in data structure of dictionary while the face value in .obj file is the key to access. Hence, finding the vertice that relate to a face only take O(1) time complexity.

Result of the final construction can be limit by non convex location and the background image. For the best scanning environment is using the black background with astral lamp light on the model. The non convex location on object can create noise can make the program failed to identify the boundary on the image. In fact, the multiple scan on different direction can decrease the effect of this situation. The detection of the object for 3d construction can be better using canny edge detection instead of the corner detection algorithm. Although I wrote the edge detection function in the project, the shorten of time doesn’t allow me to use it. The project success mostly in the optimization of algorithm. The construction of the final mesh can be construct along the convex hull on each xz-plane layer and relocation each vertice to constance distance with each other. Due to the hole in the mesh, the convex hull should also include interpolation for predicting the possible point that missed while modeling.

Appendix: Software

- Camera object: camera.py calibration.py (python version of matlab function written before)
- Data visualization: uiconstruction.py pltload.py (self scratched function)
- Image converting format and grayscale: purify.py (self scratched function)
- Image reading and writing: load.py imagearray.py (self scratched function)
- Image corner detection: ShiTomasi.py HarrisCorner.py (Implement library from OpenCv)
- Image denoise: imgDenoise.py (self scratched function)
- Images points matching: SIFT.py (implement from openCv library for point matching, self scratched for distance and corner detection)
- Image stereo: stereo.py (implementation from OpenCv)
- Images points distance estimation: distance_to_camera.py (self scratched function)
- Point cloud generation for 2 images: threeDpoint.py (self scratched function)
- Point cloud combining: objectmaker.py (self scratched function)
- Obj file handling: readobj.py writeobj.py clearobj.py (self scratched function)
- Delaunay meshing and edge optimization: Delaunay.py (self scratched function)
- Filtering mesh cleaning using original image: readmask.py clearmesh.py (self scratched function)
- Layer subdivision: laypoints.py (self scratched function)
- Convex Hull algorithm: convexhull.py (self scratched function)
- Remeshing and Smoothing: Using Maya and Meshlab (Tools)