# Final Project Writeup

Yushan Zhou, Danni Shi

## 1. Project introduction

The goal of the project is to implement the Ball Pivoting Algorithm (BPA) in Python to reconstruct surfaces/3D models from point clouds. The principle of the BPA is very simple: Three points form a triangle if a ball of a user-specified radius touches them without containing any other point. Starting with a seed triangle, the ball pivots around an edge (i.e. it revolves around the edge while keeping in contact with the edge's endpoints) until it touches another point, forming another triangle. The process continues until all reachable edges have been tried, and then starts from another seed triangle, until all points have been considered [1].

## 2. Implementation

### 2.1 Input

The input includes a text file that contains all points data and a user-specified radius.
The text file contains points information in the following format:

`x y z normal_x normal_y normal_z`

Every point is represented by six numbers. The x, y and z indicate the point's coordinates, and normal_x, normal_y and normal_z are the point's normal vector's coordinates.

### 2.2 Output

The output is a .ply[2] file that displays a 3D model generated from BPA. This file contains a header that specifies elements of the polygon and their types, like vertices, edges, and facets, and a list of elements themselves.

```
ply
format ascii 1.0
element vertex 39100
property float32 x
property float32 y
property float32 z
element face 38864
property list uint8 int32 vertex_indices
end_header
-0.06325 0.0374781 0.0428449
-0.0635 0.0367289 0.0424662
-0.0625 0.0368247 0.0433543
-0.063 0.0367836 0.0429737
-0.06275 0.0360343 0.0425949
-0.06375 0.0374237 0.0423375
-0.064 0.0366195 0.0414512
-0.06275 0.0375189 0.0432255
-0.06325 0.0374781 0.0428449
-0.0635 0.0381592 0.0425894
-0.063 0.0381998 0.0429699
-0.06425 0.0373013 0.0411957
-0.064 0.0380781 0.0418282
-0.06375 0.0388138 0.0420802
-0.06225 0.0375597 0.0436061
-0.06325 0.0388809 0.0427144
-0.063 0.0395892 0.0427125
-0.0625 0.0382403 0.0433505
```

Figure 1. On the left: the .ply file displays a 3D bunny model generated by BPA. On the right: an example of the header and elements list in a .ply file.

## 2.3 Algorithm breakup

The algorithm takes in a set of oriented points and produces a mesh. Detailed steps for creating the mesh are listed as below:

**a) Read points**

First, read points data from an input .txt file and store them in a point list. To facilitate further operations, for each point, mark whether it is used and find its neighbors by calculating the distance between it and other points.

**b) Find the seed triangle**

Choose any free point from the point list, pick up two unused points from its neighbors and generate a triangle from these three points, check if the normal of the triangle is on the same direction with point normals to guarantee the triangle is on the surface of the object, calculate the radius of the incircle of this triangle and check if it is less than the user-specified radius.

**c) Add seed triangle to the mesh and edges to the front**

The mesh is represented as a list of triangles. When the seed triangle is created, add it into the list as the root of the mesh. The BPA will check the periphery of the mesh to expand triangles, so a list of edges, the front, is required to store edges on the periphery. The three edges of the seed triangle will be added into the front list. Later, whenever the mesh is updated, the front should be updated as well.

**d) Expand triangles**

Choose an edge from the front list, check if it has unused point neighbors. If so, generate a triangle from the edge and the point, same as the seed triangle, check if its normal is on the same direction with point normals and if its incircle radius is less than the user-specified radius. If the triangle satisfies these conditions, add it into the mesh and update the front. Repeat the step until it is unable to generate a triangle from any edge in the front.

**e) Find new seed triangles**

When triangles cannot be expanded, repeat Step b) to d) by finding new seed triangles from unused points and expanding it.

**f) Terminate the process**

When all the points are used up, or no new triangles can be generated, or the number of expansion reaches the user-specified limit (it's an optional input), the program will terminate.

## 2.4 Main data structures

- **Points:** Consists of a point's x, y, z coordinates, the normal, the cell it's in.
- **Edge**: Consists of 2 Point objects which represent the 2 vertices of an edge.
- **Cell**: In order to limit the search space when expanding, we construct this structure which represents a 2r x 2r x 2r cubic. Given that if the distance between 2 balls is larger than 2r the pivoting ball won't be settled between them, we only have to check the points in P's cell and all 8 cells adjacent to this cell when generating seed triangles or expanding triangles from Point P. Each cell is represented by an encoded vertex point.
- **Triangles**: Consists of 3 Point objects which represent the 3 vertices of a mesh triangle generated by BPA.

## 3. Tests

We get input files from the IPOL Journal[3], which provides 64 public archives of original point sets.

Here is an example:

```
-0.038632 0.167867 0.002788 2.175860 1.105380 4.631200
-0.037523 0.124042 0.025944 0.255348 4.825558 3.145818
0.043233 0.089594 -0.006456 5.329182 0.855510 -2.414902
-0.026748 0.10975 0.038619 -1.235063 3.261980 5.210975
-0.063449 0.097112 -0.017938 -2.368708 0.514463 -5.785532
-0.025891 0.048891 0.049042 -1.501232 4.365808 3.313577
0.045346 0.082603 0.023076 5.826797 0.493719 1.763804
0.006619 0.047417 0.050356 1.237606 -3.614519 3.967856
-0.04071 0.050423 0.039629 -0.711702 0.563401 6.053964
-0.006309 0.051457 0.052597 -1.187559 -2.943299 5.109639
-0.064662 0.151974 0.036264 1.242156 4.225675 3.465758
-0.018605 0.049922 0.047107 -1.518097 2.022716 4.531967
-9.4e-05 0.052014 0.054278 -0.099453 -1.984873 5.065516
0.00941 0.060149 0.052895 2.331122 -1.814694 5.498609
0.000322 0.058135 0.054484 0.402913 -1.884302 5.863311
-0.085941 0.125175 0.049068 -3.749865 -0.192266 4.658874
-0.027746 0.064856 0.037498 -1.514918 -0.964871 4.400466
0.003858 0.062044 0.056777 1.130632 -1.419825 5.140898
-0.077835 0.145715 0.043808 -1.265911 3.926477 4.020892
-0.005881 0.062411 0.056478 -0.910388 -0.832178 5.824076
-0.047781 0.063839 0.037342 -3.513644 -1.680144 4.768870
0.011484 0.123198 0.03427 1.117812 4.033999 4.275233
-0.062211 0.147546 -0.005052 5.653147 0.592968 -1.929681
-0.012589 0.069082 0.054415 -1.549071 -0.979718 5.948740
```

```
ply
format ascii 1.0
element vertex 494
property float32 x
property float32 y
property float32 z
element face 585
property list uint8 int32 vertex_indices
end_header
-0.094871 0.122902 0.022189
-0.094007 0.129802 0.01818
-0.092111 0.128137 0.031104
-0.094871 0.122902 0.022189
-0.092635 0.116935 0.008842
-0.092423 0.113974 0.016676
-0.091672 0.117524 0.028069
-0.084462 0.111705 0.028857
-0.086549 0.109 0.015694
-0.08536 0.109807 0.006222
-0.086972 0.121397 -0.001898
-0.081065 0.111196 -0.000918
-0.089109 0.099057 0.008227
-0.089659 0.099833 0.017847
```

Figure 2. On the left: the input .txt file, each point is represented by six numbers, x y z coordinates and their normals. On the right: the output .ply file.

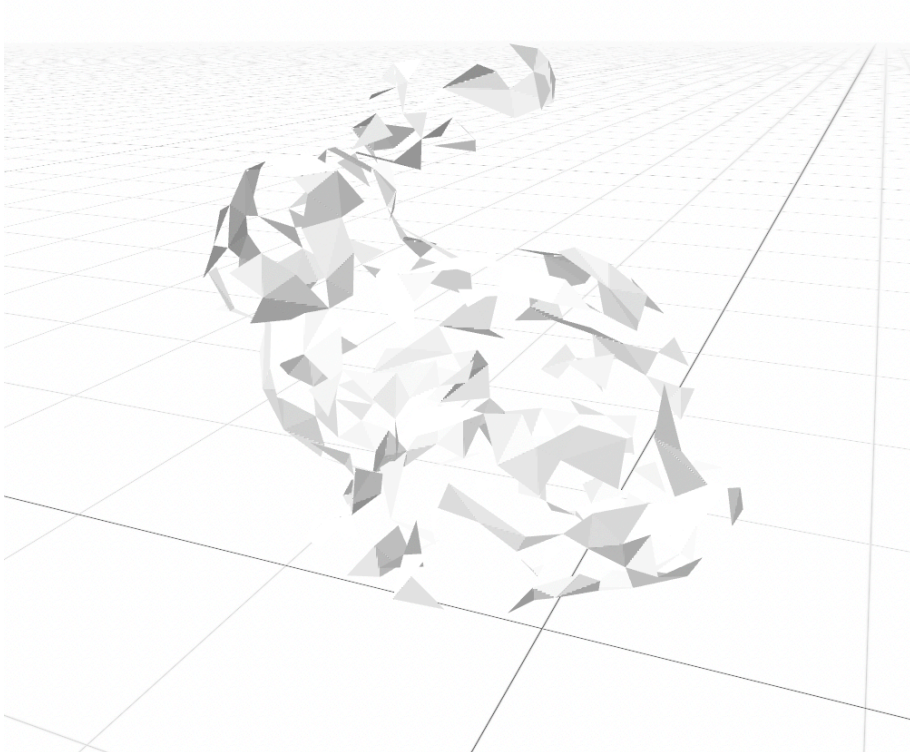

Figure 3. the output .ply file displays a 3D model of a bunny.

## 4. Discussion

**How to choose the radius**

Different user-specified radius could influence the result. A large radius may result in a shorter running time but vague result, and a smaller radius may take more time but have better performance in displaying the model. However, there existing a threshold that the performance cannot be improved anymore with the decreasing of the radius.

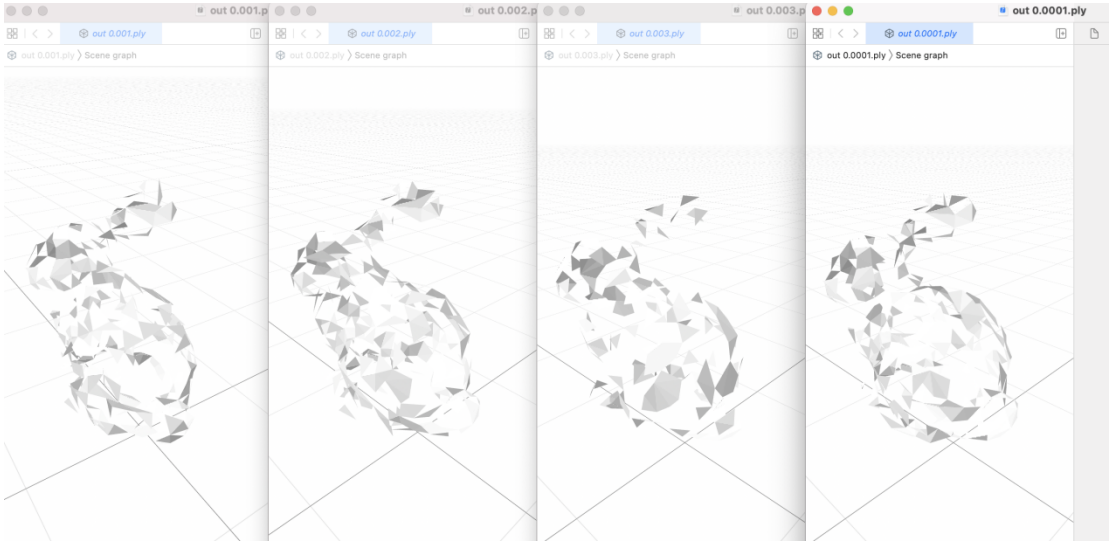

Figure 4. output .ply files with different user-specified radius
from left to right: radius = 0.001, 0.002, 0.003, and 0.0001

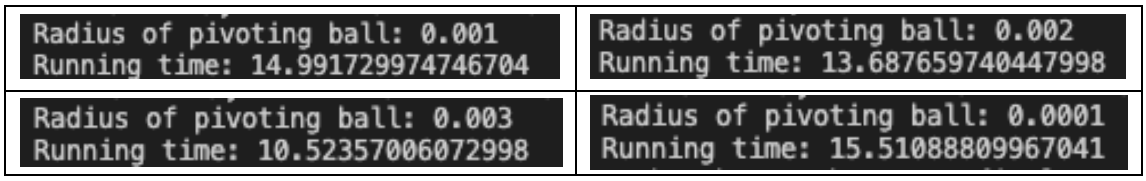


Figure 5. running time of different user-specified radius

As figure 4, 5 shows, when radius=0.003, the running time is the shortest, but in the output file, the triangles are big and details get lost. When radius decreases from 0.003 to 0.001, the size of triangles is decreasing and we could see more details. However, from 0.001 to 0.0001, the display does not improve that much.

**How to determine neighbors**

We calculate the distance between every two points. If the distance is less than 2 * radius, we consider these two points are neighbors.

**Holes**

The mesh generated from our BPA still contains lots of holes on it. One possible reason might be that the sampling density of the input dataset is too low. Some of the edges fail to be generated so some holes are left.

## References:


1. F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin, *The ball-pivoting algorithm for surface reconstruction*, IEEE Transactions on Visualization and Computer Graphics, 5 (1999), pp. 349–359. http://dx.doi.org/10.1109/2945.817351
2. PLY (file format) https://en.wikipedia.org/wiki/PLY_(file_format)
3. J. Digne, *An Analysis and Implementation of a Parallel Ball Pivoting Algorithm*, Image Processing On Line, 4 (2014), pp.149–168. https://doi.org/10.5201/ipol.2014.81