Java 2D&3D Graphics Project2

m5271506 Kiyohiro Murai

Development Environment

- Create a panel for each shape to display.

- Create a shape from vertex data and triangular mesh data
- Read vertex data and triangle mesh data from obj file.
- Create the code based on the code in the lecture materials.

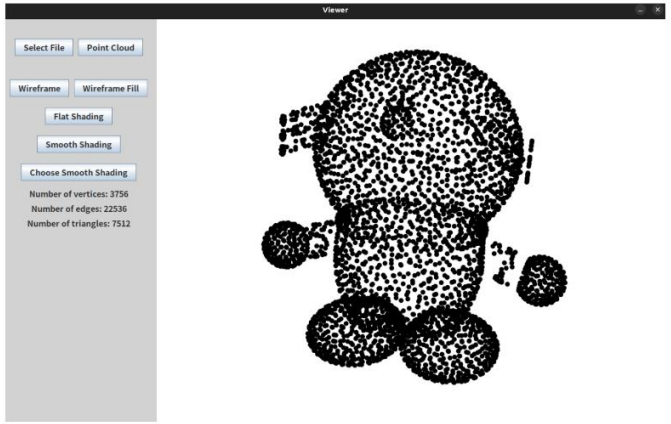

Figure 1: Screenshot of PointCloud

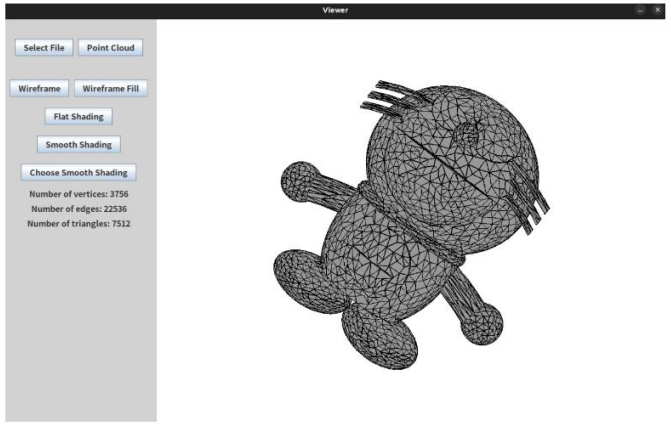

Figure 3: Screenshot of FilledWireFrame

Shading

- **FlatShading**
- Compute the vectors of the two edges using the triangle vertex coordinates. Compute the cross product of two edge vectors to obtain the normal vector. This normal vector is set as the normal vector of the mesh and the normal vector of each vertex of the triangle.

- **SmoothShading**
- In smooth sharding, I set by specifying a different normal vector for each vertex from a triangular mesh. The normal vector of each vertex is the average value of the normal vectors adjacent to the vertex.

```
public void calculateFacesNormal() {
    for (Face face : this.faces) {
        Vector v1 = vertices.get(face.getVIndex(0));
        Vector v2 = vertices.get(face.getVIndex(1));
        Vector v3 = vertices.get(face.getVIndex(2));

        Vector normal = calculateFaceNormal(v1, v2, v3);

        face.setNormal(normal);
    }
}

private Vector calculateFaceNormal(Vector v1, Vector v2, Vector v3) {
    Vector edge1 = new Vector(v2.x - v1.x, v2.y - v1.y, v2.z - v1.z, 1.0f);
    Vector edge2 = new Vector(v3.x - v1.x, v3.y - v1.y, v3.z - v1.z, 1.0f);

    Vector normal = edge1.cross(edge2);
    normal.normalize();

    return normal;
}
```

Geodesic Distance

- **For data with triangle mesh information**
- The graph is constructed based on the information of the edges of the triangular mesh. Starting from a user-selected vertex, the shortest path to all other vertices is calculated using Dijkstra's algorithm. The distances between vertices are calculated based on Euclidean distances.

- **For PointCloud with only vertex information**
- A graph was created based on the k-nearest neighbor algorithm. As the number of vertices increases, the amount of calculation increases and it takes more time. With the test data provided, it took a huge amount of time to run.

```
class Dijkstra {

    public static List<Integer> findShortestPath(Graph graph, int start, int end) {
        int startNode = start;
        int endNode = end;

        PriorityQueue<Edge> pq = new PriorityQueue<>();

        float[] dist = new float[graph.edges.size()];
        Arrays.fill(dist, Float.MAX_VALUE);
        dist[startNode] = 0;

        int[] prev = new int[graph.edges.size()];
        Arrays.fill(prev, -1);

        pq.add(new Edge(startNode, 0));

        while (!pq.isEmpty()) {
            Edge currentEdge = pq.poll();

            if (dist[currentEdge.to] < currentEdge.weight) {
                continue;
            }

            for (Edge edge : graph.edges.get(currentEdge.to)) {
                if (dist[edge.to] > dist[currentEdge.to] + edge.weight) {
                    dist[edge.to] = dist[currentEdge.to] + edge.weight;
                    prev[edge.to] = currentEdge.to;
                    pq.add(new Edge(edge.to, dist[edge.to]));
                }
            }
        }

        List<Integer> path = new ArrayList<>();
        for (int at = endNode; at != -1; at = prev[at]) {
            path.add(at);
            if (at == startNode)
                break;
        }
        Collections.reverse(path);

        return path;
    }
}
```

Demo

Problems

- **For PointCloud with only vertex information**
- The k-nearest neighbor algorithm took an enormous amount of time to run on the test data provided. The use of kd tree can be considered for improvement. But I couldn’t implement it.

- **Mouse translation problem**
- I wanted to use the mouse to move a shape in translation, but for some reason the behavior didn’t work properly. The officially stated default key (shift key) was assigned a rotation function, but the movement function did not work correctly.

- And the function to save figure data is also not implemented.