Group Member

Yasheng Sun 117020910076

Qichang Sun 117020910077

PS

My code is in https://github.com/sunyasheng/Computer-Graphics/. Core function code is listed in Appendix for convenience.

Problem Restatement

Methodology

Vertices Selection and Offsetting

The procedure is described in Fig.1. Basing on the region of interest, we find the entity of model included in the rectangle that user chooses. Also, we make up the code of rendering a rectangle which is listed in Appendix. Once the vertices are determined, we are capable of moving those vertices while holding the remaining vertices. Code accomplishing this function has been given, which basically adds an offset distance to the vertices in rectangle.

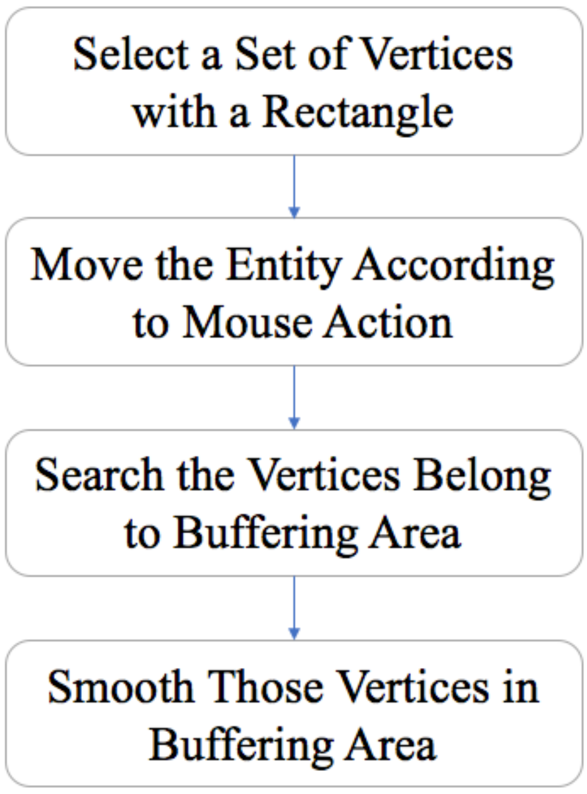

Select a Set of Vertices with a Rectangle
Move the Entity According to Mouse Action
Search the Vertices Belong to Buffering Area
Smooth Those Vertices in Buffering Area

Fig. 1. Workflow of Moving an Entity According to Choosing Area

Vertices Searching with BFS and Smoothing

There is some difficulty in buffer area definition. Intuitively, the contour which connect the moved entity with remaining entity is buffer area. However, it is impossible to eliminate those long and narrow triangles formed by stretching if we only apply smoothing to those vertices of triangles. Therefore, we first search the vertices surrounding those vertices according to distance using BFS. In second segment of code, the buffer size indicates how many steps we allow BFS to search. We just use the traditional data structure, queue, to mark the vertices required to be smoothed for later use.

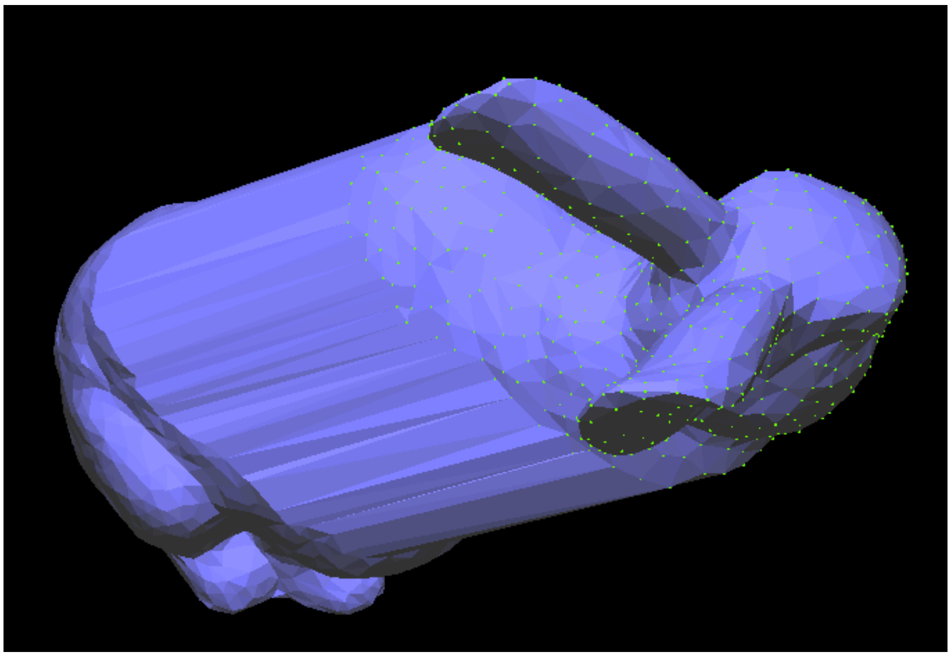


Fig. 2. Display in Flat Shaded Mode without Smoothing

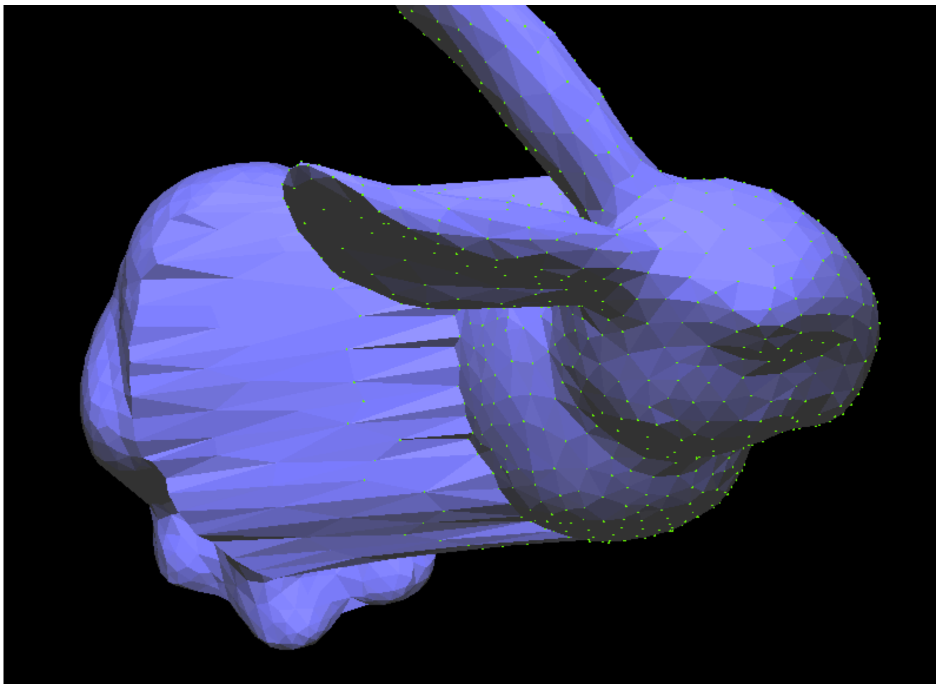


Fig. 3. Display in Flat Shaded Mode with Smoothing

About Laplacian Smoothing, a class is built to finish the Laplacian Smoothing. The code for construct Laplacian Matrix is listed in third segment of Appendix. In brief, we just add relevant vertices which is marked by a flag in the BFS process to Laplacian Matrix.

To present the smoothing result clearly, we show the results in flat shaded mode. The result in Fig. 2 shows that the buffer area is composed of a loop of narrow triangles before smoothing. After smoothing, the buffer area consists of many loops of triangles instead of only one loop as is shown in Fig. 3. This is because we smooth vertices surrounding the buffer area instead of just vertices in one loop of triangles. For the completeness of report, results in smooth shaded model are shown in Fig. 4.

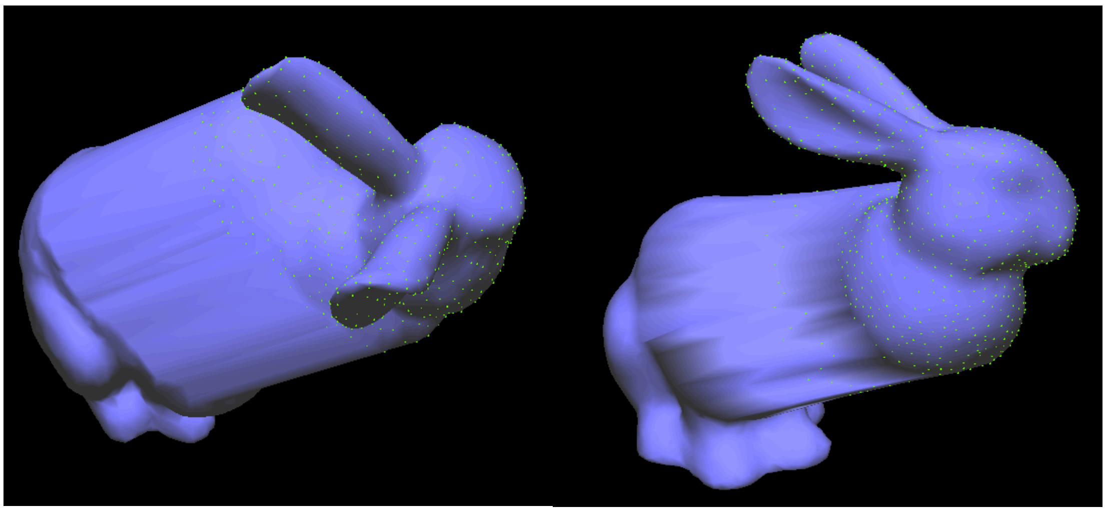


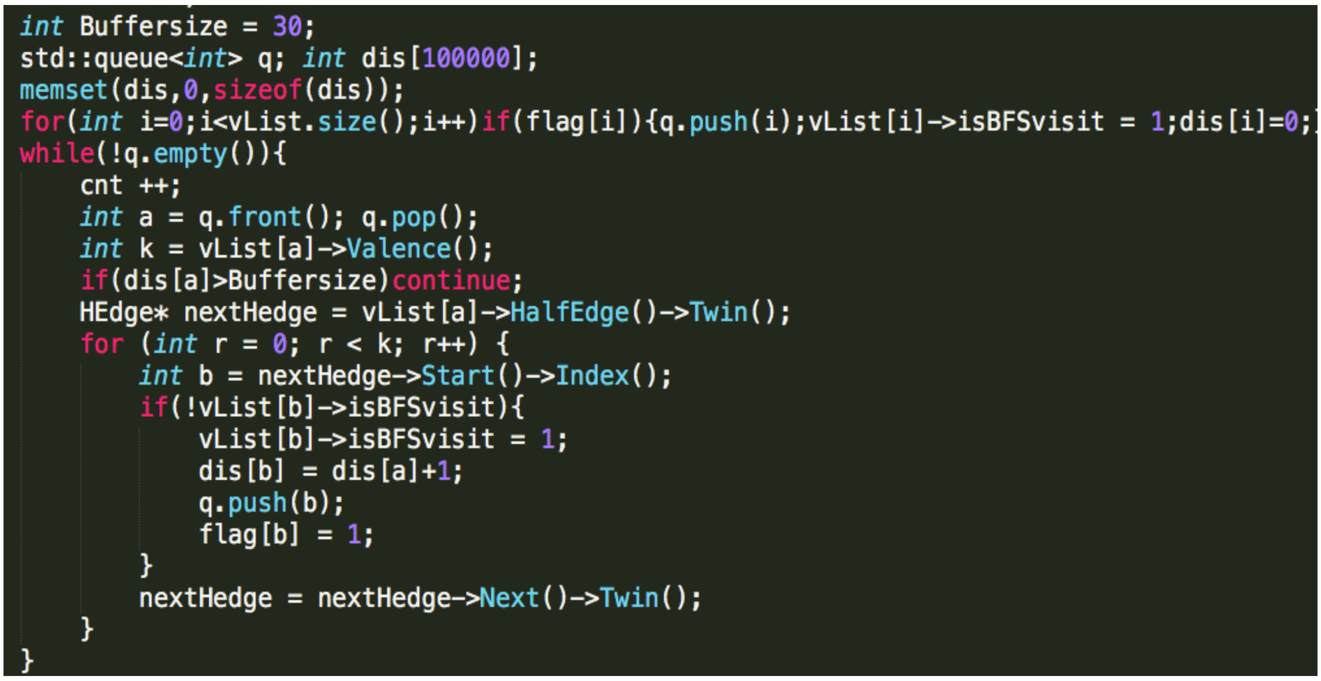
Fig. 4. Left is the result without smoothing, right is the result with smoothing.

Appendix

Function of rendering a rectangle.

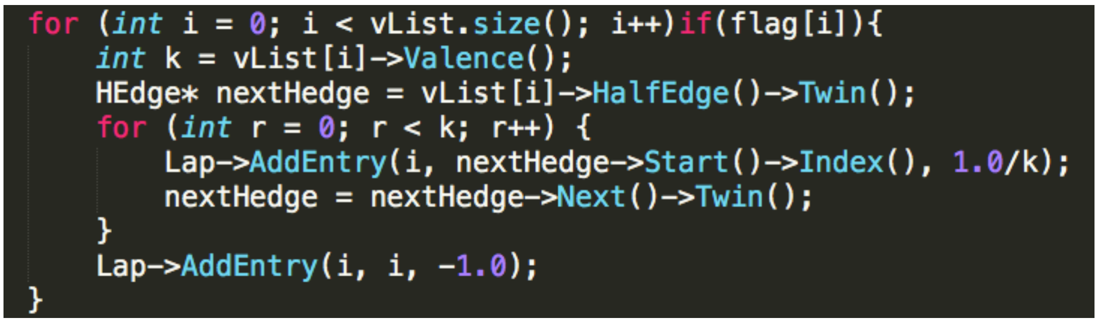
```
void DrawSelectionRect() {
    int x1 = downX, x2 = lastX, y1 = winHeight - downY, y2 = winHeight - lastY;
    GLProjector projector1;
    Eigen::Vector3d v1 = projector1.UnProject(x1, y1, zNear);
    Eigen::Vector3d v2 = projector1.UnProject(x2, y1, zNear);
    Eigen::Vector3d v3 = projector1.UnProject(x2, y2, zNear);
    Eigen::Vector3d v4 = projector1.UnProject(x1, y2, zNear);
    glColor3f(1.0, 0, 0);
    glBegin(GL_LINE_STRIP);
    glVertex3dv(v1.data());
    glVertex3dv(v2.data());
    glVertex3dv(v3.data());
    glVertex3dv(v4.data());
    glVertex3dv(v1.data());
    glEnd();
}
```

Function of searching vertices belong to buffering area.



```
int Buffersize = 30;
std::queue<int> q; int dis[100000];
memset(dis,0,sizeof(dis));
for(int i=0;i<vList.size();i++)if(flag[i]){q.push(i);vList[i]->isBFSvisit = 1;dis[i]=0;
while(!q.empty()){
    cnt ++;
    int a = q.front(); q.pop();
    int k = vList[a]->Valence();
    if(dis[a]>Buffersize)continue;
    HEdge* nextHedge = vList[a]->HalfEdge()->Twin();
    for (int r = 0; r < k; r++) {
        int b = nextHedge->Start()->Index();
        if(!vList[b]->isBFSvisit){
            vList[b]->isBFSvisit = 1;
            dis[b] = dis[a]+1;
            q.push(b);
            flag[b] = 1;
        }
        nextHedge = nextHedge->Next()->Twin();
    }
}
```

Code for Laplacian Smoothing



```
for (int i = 0; i < vList.size(); i++)if(flag[i]){
    int k = vList[i]->Valence();
    HEdge* nextHedge = vList[i]->HalfEdge()->Twin();
    for (int r = 0; r < k; r++) {
        Lap->AddEntry(i, nextHedge->Start()->Index(), 1.0/k);
        nextHedge = nextHedge->Next()->Twin();
    }
    Lap->AddEntry(i, i, -1.0);
}
```