```cpp
#include <GL/glut.h>
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <assert.h>
#include <string.h>
#include<vector>
#include <fstream>
#include "arcball.h"

using namespace std;

#define PI 3.1415926536
int choose = 0;

typedef struct Vertex
{
    float x, y, z;
}Vertex;

typedef struct Face
{
    int num;
    int order[3];
}Face;

typedef struct HalfEdge//半边结构
{
    int origin;
    struct HalfEdge* next;
    struct HalfEdge* opposite;
    int IncFace;
}HalfEdge;

typedef struct Map
{
    int vs, ve, e;

}Map;

vector<Vertex> vertex;
vector<Face> face;
vector<HalfEdge*> edge;
int e_num;
int n_node, n_face, n_edge;
int width = 800;
int height = 800;

int readoff()
{
    char theoff[233] = "bunny.off";
    // cin >> theoff;
    ifstream fin(theoff);
    if (!fin.is_open()) {
        puts("error opening file.");
        exit(1);
    }
    string isoff;
    do {
        fin >> isoff;
    } while (isoff[0] == '#');
    if (isoff != "OFF") {
        puts("this is not a .off file.");
        exit(1);
    }
    fin >> n_node >> n_face >> n_edge;

    for (int i = 0; i < n_node; i++)
    {
        Vertex ver;
        fin >> ver.x >> ver.y >> ver.z;
        vertex.push_back(ver);
    }
    for (int i = 0; i < n_face; i++)
    {
        Face f;
        fin >> f.num >> f.order[0] >> f.order[1] >> f.order[2];
        face.push_back(f);
    }
}

void initEdge()//生成半边存入vector
{
    int** map = new int* [n_node];
    for (int i = 0; i < n_node; i++) {
        map[i] = new int[n_node];
    }
    for (int i = 0; i < n_node; i++)
    {
        for (int j = 0; j < n_node; j++)
        {
            map[i][j] = -1;
        }
    }
    e_num = 0;
    for (int i = 0; i < n_face; i++)
    {
        HalfEdge* edge1 = new HalfEdge();
        HalfEdge* edge2 = new HalfEdge();
        HalfEdge* edge3 = new HalfEdge();

        edge1->origin = face[i].order[0];
        edge2->origin = face[i].order[1];
        edge3->origin = face[i].order[2];

        edge1->next = edge2;
        edge2->next = edge3;
        edge3->next = edge1;

        HalfEdge* tmpe = new HalfEdge();
        if (map[face[i].order[1]][face[i].order[0]] != -1)
        {
            tmpe = edge[map[face[i].order[1]][face[i].order[0]]];
            edge1->opposite = tmpe;
            tmpe->opposite = edge1;
        }
        else
        {
            edge1->opposite = NULL;
            map[face[i].order[0]][face[i].order[1]] = e_num;
        }
        e_num++;
        if (map[face[i].order[2]][face[i].order[1]] != -1)
        {
            tmpe = edge[map[face[i].order[2]][face[i].order[1]]];
            edge2->opposite = tmpe;
            tmpe->opposite = edge2;
        }
        else
        {
            edge2->opposite = NULL;
            map[face[i].order[1]][face[i].order[2]] = e_num;
        }
        e_num++;
        if (map[face[i].order[0]][face[i].order[2]] != -1)
        {
            tmpe = edge[map[face[i].order[0]][face[i].order[2]]];
            edge3->opposite = tmpe;
            tmpe->opposite = edge3;
        }
        else
        {
            edge3->opposite = NULL;
            map[face[i].order[2]][face[i].order[0]] = e_num;
        }
        e_num++;

        edge1->IncFace = i;
        edge2->IncFace = i;
        edge3->IncFace = i;

        edge.push_back(edge1);
        edge.push_back(edge2);
        edge.push_back(edge3);
    }
    n_edge = edge.size();
}

```

```cpp
HalfEdge* findOriginEdge(int v)//找到从该定点出发的一条半边
{
    for (int k = 0; k < n_edge; k++)
    {
        if (edge[k]->origin == v)
            return edge[k];
    }
    return NULL;
}

void subdivide()
{
    vector<Vertex> vertex2;
    vector<Face> face2;
    vector<HalfEdge*> edge2;
    HalfEdge* he = new HalfEdge();
    int n;
    float p_sumx, p_sumy, p_sumz;
    float px, py, pz;
    float beta;
    cout << "细分开始" << endl;
    for (int i = 0; i < n_node; i++)//旧点更新
    {
        he = findOriginEdge(i);

        if (he != NULL)
        {
            n = 0;
            p_sumx = 0;
            p_sumy = 0;
            p_sumz = 0;
            HalfEdge* e = new HalfEdge();
            e = he->next;
            int p0 = e->origin;

            while (e->next->origin != p0)
            {
                n++;
                p_sumx += vertex[e->next->origin].x;
                p_sumy += vertex[e->next->origin].y;
                p_sumz += vertex[e->next->origin].z;
                HalfEdge* te = new HalfEdge();
                te = e->next->opposite;
                e = te->next;
            }
            n++;
            p_sumx += vertex[p0].x;
            p_sumy += vertex[p0].y;
            p_sumz += vertex[p0].z;
            beta = 1 / (double)n * (0.625 - pow(0.375 + 0.25 * cos(2 * PI / n),
              2));

            px = (1 - n * beta) * vertex[i].x + beta * p_sumx;
            py = (1 - n * beta) * vertex[i].y + beta * p_sumy;
            pz = (1 - n * beta) * vertex[i].z + beta * p_sumz;

            Vertex v;
            v.x = px;
            v.y = py;
            v.z = pz;
            vertex2.push_back(v);
        }
    }
    int** map1 = new int* [n_node];
    for (int i = 0; i < n_node; i++) {
        map1[i] = new int[n_node];
    }

    cout << "map1=" << sizeof(map1[0]) / sizeof(int) << endl;
    float qx, qy, qz;

    for (int i = 0; i < n_edge; i++)//新点生成
    {
        if (!map1[edge[i]->origin][edge[i]->next->origin])
        {
            int p = edge[i]->origin;
            int pi = edge[i]->next->origin;
            int pi1 = edge[i]->next->next->origin;
            int pi0 = edge[i]->opposite->next->next->origin;
            qx = 0.375 * (vertex[p].x + vertex[pi].x) + 0.125 * (vertex[pi1].x + vertex[pi0].x);
            qy = 0.375 * (vertex[p].y + vertex[pi].y) + 0.125 * (vertex[pi1].y + vertex[pi0].y);
            qz = 0.375 * (vertex[p].z + vertex[pi].z) + 0.125 * (vertex[pi1].z + vertex[pi0].z);

            Vertex v;
            v.x = qx;
            v.y = qy;
            v.z = qz;
            vertex2.push_back(v);

            map1[edge[i]->origin][edge[i]->next->origin] = vertex2.size() - 1;
            map1[edge[i]->next->origin][edge[i]->origin] = vertex2.size() - 1;
        }
    }
    /*
    cout<<"新点"<<endl;
    for(int i=0;i<vertex2.size();i++)
    {
        cout<<vertex2[i].x<<" "<<vertex2[i].y<<" "<<vertex2[i].z<<endl;
    }
    */
    for (int i = 0; i < n_face; i++)//新面
    {
        int a, b, c, d, e, f;
```

```
            a = face[i].order[0];
            b = face[i].order[1];
            c = face[i].order[2];
            d = map1[a][b];
            e = map1[b][c];
            f = map1[a][c];

            Face f2;
            f2.num = 3;

            f2.order[0] = a;
            f2.order[1] = d;
            f2.order[2] = f;
            face2.push_back(f2);

            f2.order[0] = d;
            f2.order[1] = b;
            f2.order[2] = e;
            face2.push_back(f2);

            f2.order[0] = d;
            f2.order[1] = e;
            f2.order[2] = f;
            face2.push_back(f2);

            f2.order[0] = f;
            f2.order[1] = e;
            f2.order[2] = c;
            face2.push_back(f2);
        }

        n_face = face2.size();
        n_node = vertex2.size();
        cout << n_node << " " << n_face << endl;


        int** map2 = new int* [n_node];
        for (int i = 0; i < n_node; i++) {
            map2[i] = new int[n_node];
        }
        for (int i = 0; i < n_node; i++)
        {

            for (int j = 0; j < n_node; j++)
            {
                map2[i][j] = -1;
            }
        }
        e_num = 0;
        for (int i = 0; i < n_face; i++)//新边
        {
            HalfEdge* edge4 = new HalfEdge();
            HalfEdge* edge5 = new HalfEdge();
```

```cpp
        HalfEdge* edge6 = new HalfEdge();

        edge4->origin = face2[i].order[0];
        edge5->origin = face2[i].order[1];
        edge6->origin = face2[i].order[2];

        edge4->next = edge5;
        edge5->next = edge6;
        edge6->next = edge4;

        HalfEdge* tmpe = new HalfEdge();
        if (map2[face2[i].order[1]][face2[i].order[0]] != -1)
        {
            tmpe = edge2[map2[face2[i].order[1]][face2[i].order[0]]];
            edge4->opposite = tmpe;
            tmpe->opposite = edge4;
        }
        else
        {
            edge4->opposite = NULL;
            map2[face2[i].order[0]][face2[i].order[1]] = e_num;
        }
        e_num++;
        if (map2[face2[i].order[2]][face2[i].order[1]] != -1)
        {
            tmpe = edge2[map2[face2[i].order[2]][face2[i].order[1]]];
            edge5->opposite = tmpe;
            tmpe->opposite = edge5;
        }
        else
        {
            edge5->opposite = NULL;
            map2[face2[i].order[1]][face2[i].order[2]] = e_num;
        }
        e_num++;
        if (map2[face2[i].order[0]][face2[i].order[2]] != -1)
        {
            tmpe = edge2[map2[face2[i].order[0]][face2[i].order[2]]];
            edge6->opposite = tmpe;
            tmpe->opposite = edge6;
        }
        else
        {
            edge6->opposite = NULL;
            map2[face2[i].order[2]][face2[i].order[0]] = e_num;
        }
        e_num++;

        edge4->IncFace = i;
        edge5->IncFace = i;
        edge6->IncFace = i;

        edge2.push_back(edge4);
        edge2.push_back(edge5);
        edge2.push_back(edge6);
    }
    n_edge = edge2.size();
    /*
    cout<<"新边"<<endl;
    for(int i=0;i<edge2.size();i++)
    {
        cout<<edge2[i]->origin<<" "<<edge2[i]->next->origin<<" "<<edge2[i]-
          >IncFace<<endl;
    }
    */
    vertex.assign(vertex2.begin(), vertex2.end());

    face.assign(face2.begin(), face2.end());

    edge.assign(edge2.begin(), edge2.end());

    cout << "完成一次细分" << endl;
    cout << n_node << " " << n_edge << " " << n_face << endl;











}

ArcBallT arcBall(600.0f, 400.0f);
ArcBallT* ArcBall = &arcBall;// new ArcBallT(600.0f,400.0f);//&arcBall;

void display()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    glLoadIdentity();
    gluLookAt(0.0, 0.0, 5.0,
        0.0, 0.0, 0.0,
        0.0, 1.0, 0.0);
    glScalef(1.0, 2.0, 1.0);

    //glPushMatrix();
    glTranslatef(ArcBall->rightPan, ArcBall->upPan, 0);//1. 移动
    glScalef(ArcBall->zoomRate, ArcBall->zoomRate, ArcBall->zoomRate);//2. 缩放
    glMultMatrixf(ArcBall->Transform.M);                              //3. 旋转

    glBegin(GL_TRIANGLES);
    for (int i = 0; i < n_face; i++)
    {
        glVertex3f(vertex[face[i].order[0]].x, vertex[face[i].order[0]].y, vertex
          [face[i].order[0]].z);
        glVertex3f(vertex[face[i].order[1]].x, vertex[face[i].order[1]].y, vertex
          [face[i].order[1]].z);
        glVertex3f(vertex[face[i].order[2]].x, vertex[face[i].order[2]].y, vertex
          [face[i].order[2]].z);
    }
    glEnd();



    float a, b, c;
    a = vertex[choose].x;
    b = vertex[choose].y;
    c = vertex[choose].z;

    //设置点的大小
    glPointSize(7);
    //进行平滑处理
    glEnable(GL_POINT_SMOOTH);
    glHint(GL_POINT_SMOOTH, GL_NICEST);

    glBegin(GL_POINTS);
    //指定的点，换成绿色
    glColor3f(0, 255, 0);
    glVertex3f(a, b, c);
    glEnd();

    HalfEdge* edge = findOriginEdge(choose);

    do {
        glBegin(GL_POINTS);
        //相邻的点，换成红色
        glColor3f(255, 0, 0);
        glVertex3f(vertex[edge->origin].x, vertex[edge->origin].y, vertex[edge-
          >origin].z);
        glEnd();

        glBegin(GL_LINE_LOOP);
        //glLineWidth(3.0f);
        //相邻的边，换成蓝色
        glColor4ub(0, 0, 255, 255);
        glColor3f(0, 0, 255);
        glVertex3f(a, b, c);
        glVertex3f(vertex[edge->opposite->origin].x, vertex[edge->opposite-
          >origin].y, vertex[edge->opposite->origin].z);
        glEnd();

        edge = edge->opposite->next;
    } while (edge != findOriginEdge(choose));

    glPopMatrix();
    glutSwapBuffers();
}

void keyboard(unsigned char key, int x, int y)
{
    switch (key)
    {
    case '1':
        glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
        break;
    case '2':
        glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
        break;
    case '3':
        glPolygonMode(GL_FRONT_AND_BACK, GL_POINT);
        break;
    case 'w':
        subdivide();
        break;
    }
    glutPostRedisplay();
}

void reshape(int w, int h) {
    //定义视口大小
    glViewport(0, 0, (GLsizei)w, (GLsizei)h);
    //投影显示
    glMatrixMode(GL_PROJECTION);
    //坐标原点在屏幕中心
    glLoadIdentity();
    //操作模型视景
    gluPerspective(60.0, (GLfloat)w / (GLfloat)h, 1.0, 20.0);
    glMatrixMode(GL_MODELVIEW);
}

//移动
void move(int x, int y)
{
    ArcBall->MousePt.s.X = x;
    ArcBall->MousePt.s.Y = y;
    ArcBall->upstate();
    glutPostRedisplay();
}
//点击
void mouse(int button, int state, int x, int y)
{
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
        ArcBall->isClicked = true;
        move(x, y);
    }
    else if (button == GLUT_LEFT_BUTTON && state == GLUT_UP)
        ArcBall->isClicked = false;
    else if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) {
        ArcBall->isRClicked = true;
        move(x, y);
    }
    else if (button == GLUT_RIGHT_BUTTON && state == GLUT_UP)
        ArcBall->isRClicked = false;
    else if (button == GLUT_RIGHT_BUTTON && state == GLUT_UP)
        ArcBall->isRClicked = false;
    else if (button == GLUT_MIDDLE_BUTTON && state == GLUT_DOWN) {
        ArcBall->isMClicked = true;
        move(x, y);
    }
    else if (button == GLUT_MIDDLE_BUTTON && state == GLUT_UP)
        ArcBall->isMClicked = false;
    ArcBall->upstate();
    glutPostRedisplay();
}

int main(int argc, char** argv)
{
    cout << "给出顶点的索引（第几个点），将该点变为绿色，该点连接的边变为蓝色，与
      该点相连的顶点变为红色：" << endl;
    cin >> choose;
    readoff();
    initEdge();

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(width, height);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("loop");
    glutReshapeFunc(reshape);
    glutDisplayFunc(display);
    glutIdleFunc(display);
    glutMouseFunc(mouse);
    glutMotionFunc(move);
    glutKeyboardFunc(keyboard);
    glutMainLoop();
    return 0;
}
```