**Final Project - Mesh Processing**

**学号：161810225姓名：王烨文 班级：1618104**

**分数：**

**一个月时间完成**

1．计算三角网格中每个三角形(face)的面积；10分

Scalar calc\_facet\_area(const FacetHandle& \_fh)

代码如下：

ExKernelT<ExItems>::calc\_facet\_area(const FacetHandle& \_fh){/////计算三角形的面积

///得到半边句柄

assert(\_fh.is\_valid());

assert(\_fh.idx() < facet\_size());

const HalfedgeHandle& hh = halfedge\_handle(\_fh);

const HalfedgeHandle& p\_hh = prev\_halfedge\_handle(hh);

const HalfedgeHandle& n\_hh = next\_halfedge\_handle(hh);

////由半边句柄得到边句柄

const EdgeHandle& e\_hh = edge\_handle(hh);

const EdgeHandle& e\_phh = edge\_handle(p\_hh);

const EdgeHandle& e\_nhh = edge\_handle(n\_hh);

////由边句柄得到各边长

double length0 = calc\_edge\_length(e\_hh);

double length1 = calc\_edge\_length(e\_phh);

double length2 = calc\_edge\_length(e\_nhh);

double p = (length0 + length1 + length2) / 2;

////利用海伦公式求面积s=sqrt(p(p-a)(p-b)(p-c))

Scalar area = 0.0;

area = sqrt(p \* (p - length0) \* (p - length1) \* (p - length2));

facet\_ref(\_fh).area\_ = area;

return area;

}

2．**计算三角网格中每个顶点(vertex) 的法向；20分**

Normal calc\_normal(const VertexHandle& \_vh)

代码如下：

ExKernelT<ExItems>::calc\_normal(const VertexHandle& \_vh) {////更新一个顶点的法向

assert( \_vh.is\_valid());

assert( \_vh.idx() < vertex\_size() );

Normal norm(0,0,0);///////用norm存储求得的法向值

//////////////////////////在此实现////////////////////////////////////

std::vector<FacetHandle> \_fhs;

HalfedgeHandle& hh = halfedge\_handle(\_vh);

VertexHandle vhs;

vhs = vertex\_handle(hh);

FacetHandle \_fh = facet\_handle(hh);

HalfedgeHandle& hhv = halfedge\_handle(vhs);

HalfedgeHandle cursor(hhv);

FacetHandle fh = facet\_handle(cursor);

\_fhs.push\_back(\_fh);//放入第一面

int j;

do {

FacetHandle fh = facet\_handle(cursor);

if (fh.is\_valid() && fh != \_fh) {

if (\_fhs.size() != 0) {

for (j = 0; j < \_fhs.size(); j++) {

if (fh.idx() == \_fhs[j].idx()) break;

}//end for

if (j >= \_fhs.size()) \_fhs.push\_back(fh);

}//end if

else \_fhs.push\_back(fh);

}//end if

cursor = cw\_rotated(cursor);

} while (hhv != cursor);//end for do while

double total\_area = 0.0;

for (int i = 0; i < \_fhs.size(); i++)

{

total\_area += calc\_facet\_area(\_fhs[i]);

}

for (int i = 0; i < \_fhs.size(); i++)

{

double weight = calc\_facet\_area(\_fhs[i]) / total\_area;

norm += calc\_normal(\_fhs[i]) \* weight;

}

/////////////////////////////////////////////////////////////////////

return norm.normalize();

}

3．**用OpenGL把三角网格中每个三角面片(face)的法向画出来（提示：在每个三角面片的重心处画）**；20分

bool ogl\_writer2(bool \_orient = true, bool \_smooth = false);

代码如下：

bool ReaderWriterT<Mesh>::ogl\_writer2(bool \_orient, bool \_smooth){////在里面把三角面片法向画出

HalfedgeHandle cshh;

Mesh::FacetIterator fit(mesh\_->facet\_begin());

//glShadeModel(GL\_FLAT);

glShadeModel(GL\_SMOOTH);

int orient = true;// (\_orient) ? 1 : -1;

mesh\_->update\_normals();

ogl\_writer(\_orient, \_smooth);

for (; fit != mesh\_->facet\_end(); ++fit) {

if ((\*fit).status\_.is\_deleted()) continue;

cshh = fit->halfedge\_handle\_;

FacetHandle fh = mesh\_->facet\_handle(cshh);

Mesh::Normal norm = mesh\_->calc\_normal(fh);

const HalfedgeHandle& hh = mesh\_->halfedge\_handle(fh);

const HalfedgeHandle& p\_hh = mesh\_->prev\_halfedge\_handle(hh);

const HalfedgeHandle& n\_hh = mesh\_->next\_halfedge\_handle(hh);

const Coord& cd0 = mesh\_->coord(mesh\_->vertex\_handle(hh));

const Coord& cd1 = mesh\_->coord(mesh\_->vertex\_handle(p\_hh));

const Coord& cd2 = mesh\_->coord(mesh\_->vertex\_handle(n\_hh));

double gpx = (cd0.data\_[0] + cd1.data\_[0] + cd2.data\_[0]) / 3;

double gpy = (cd0.data\_[1] + cd1.data\_[1] + cd2.data\_[1]) / 3;

double gpz = (cd0.data\_[2] + cd1.data\_[2] + cd2.data\_[2]) / 3;

glBegin(GL\_LINES);

glVertex3f(gpx,gpy,gpz);

glVertex3f(gpx + 0.3 \* norm.x, gpy + 0.3 \* norm.y, gpz + 0.3 \* norm.z);

glVertex3f(gpx + 0.3 \* norm.x, gpy + 0.3 \* norm.y, gpz + 0.3 \* norm.z);

glVertex3f(gpx + 0.2 \* norm.x - 0.05 , gpy + 0.25 \* norm.y, gpz + 0.25 \* norm.z);

glVertex3f(gpx + 0.3 \* norm.x, gpy + 0.3 \* norm.y, gpz + 0.3 \* norm.z);

glVertex3f(gpx + 0.2 \* norm.x + 0.05, gpy + 0.25 \* norm.y, gpz + 0.25 \* norm.z);

glEnd();

}

return true;

}

效果如下（**注意：贴自己的实验结果！！！**）：

|  |
| --- |
|  |

4．**实现一种简单的三角网格去噪算法（例如拉普拉斯光顺）；30分**

void Laplacian\_Smoothing();

代码如下：

void ExKernelT<ExItems>::Laplacian\_Smoothing() {

/////请实现自己的去噪算法////////

int n;

std::cout << "请输入迭代次数:";

std::cin >> n;

double laplac = 0.0;

std::cout << "请输入拉普拉斯系数(越接近本身->1):";

std::cin >> laplac;

for (int i = 1; i <= n; i++)

{

std::vector<Coord> updateVertexPosition;

std::vector<VertexHandle> v\_near;

int vertex\_num = vertex\_size();

updateVertexPosition.resize(vertex\_num);

for (int i = 0; i < vertex\_num; i++) {

VertexHandle vh(i);

getNeighborRing(vh, 1, v\_near);

double xsum = 0.0, ysum = 0.0, zsum = 0.0, dsum = 0.0;

double d = 0.0;

for (int j = 1; j < v\_near.size(); j++)

{

d = sqrt((coord(v\_near[j]).x- coord(v\_near[0]).x) \* (coord(v\_near[j]).x - coord(v\_near[0]).x) + (coord(v\_near[j]).y - coord(v\_near[0]).y) \* (coord(v\_near[j]).y - coord(v\_near[0]).y) + (coord(v\_near[j]).z - coord(v\_near[0]).z) \* (coord(v\_near[j]).z - coord(v\_near[0]).z));

dsum += d;

xsum += coord(v\_near[j]).x\*d;

ysum += coord(v\_near[j]).y\*d;

zsum += coord(v\_near[j]).z\*d;

}

xsum = xsum / dsum;

ysum = ysum / dsum;

zsum = zsum / dsum;

Coord cd(xsum, ysum, zsum);

updateVertexPosition[i] = coord(v\_near[0]) \* laplac + cd \* (1 - laplac);

v\_near.clear();

}

for (int i = 0; i < vertex\_num; i++)

{

vertex\_ref(VertexHandle(i)).coord\_ = updateVertexPosition[i];

}

}

}

效果如下（**注意：贴自己的实验结果！！！**）：

|  |  |
| --- | --- |
| 去噪前 | 迭代5次，拉普拉斯系数0.8  去噪后 |

**5.（可选/可做可不做）实现一种基于三角网格的操作（例如，特征提取，三角网格分割，三角网格平面参数化，三角网格变形等等）；20分**

void mesh\_process();

代码如下：

void ExKernelT<ExItems>::mesh\_process(std::map<VertexHandle,double> &Gosk){///////(可选，在三角网格上实现一种操作，例如特征提取，网格分割，三角网格变形等等)

VertexIterator vi = vertex\_begin();

for (; vi != vertex\_end(); ++vi) {

if (vi->status\_.is\_deleted()) continue;

assert(vi->halfedge\_handle\_.is\_valid());

VertexHandle& vh = vertex\_handle(vi->halfedge\_handle\_);

std::vector<FacetHandle> \_fhs;

HalfedgeHandle& hh = halfedge\_handle(vh);

VertexHandle vhs;

vhs = vertex\_handle(hh);

FacetHandle \_fh = facet\_handle(hh);

HalfedgeHandle& hhv = halfedge\_handle(vhs);

HalfedgeHandle cursor(hhv);

FacetHandle fh = facet\_handle(cursor);

\_fhs.push\_back(\_fh);//放入第一面

int j;

do {

FacetHandle fh = facet\_handle(cursor);

if (fh.is\_valid() && fh != \_fh) {

if (\_fhs.size() != 0) {

for (j = 0; j < \_fhs.size(); j++) {

if (fh.idx() == \_fhs[j].idx()) break;

}//end for

if (j >= \_fhs.size()) \_fhs.push\_back(fh);

}//end if

else \_fhs.push\_back(fh);

}//end if

cursor = cw\_rotated(cursor);

} while (hhv != cursor);//end for do while

double total\_area = 0.0;

double area = 0.0;

double Angle = 2 \* 3.1415926535898;

for (int i = 0; i < \_fhs.size(); i++) {

const HalfedgeHandle& hh = halfedge\_handle(\_fhs[i]);

const HalfedgeHandle& p\_hh = prev\_halfedge\_handle(hh);

const HalfedgeHandle& n\_hh = next\_halfedge\_handle(hh);

Coord cd0 = coord(vertex\_handle(hh));

Coord cd1 = coord(vertex\_handle(p\_hh));

Coord cd2 = coord(vertex\_handle(n\_hh));

Coord cd\_ori = coord(vh);

if (cd\_ori == cd0) {

;

}

else if (cd\_ori == cd1) {

cd1 = coord(vertex\_handle(hh));

cd0 = coord(vh);

}

else if (cd\_ori == cd2) {

cd2 = coord(vertex\_handle(hh));

cd0 = coord(vh);

}

double a = sqrt(pow(cd0.data\_[0] - cd1.data\_[0], 2) + pow(cd0.data\_[1] - cd1.data\_[1], 2) + pow(cd0.data\_[2] - cd1.data\_[2], 2));

double b = sqrt(pow(cd0.data\_[0] - cd2.data\_[0], 2) + pow(cd0.data\_[1] - cd2.data\_[1], 2) + pow(cd0.data\_[2] - cd2.data\_[2], 2));

double c = sqrt(pow(cd2.data\_[0] - cd1.data\_[0], 2) + pow(cd2.data\_[1] - cd1.data\_[1], 2) + pow(cd2.data\_[2] - cd1.data\_[2], 2));

int tri = 0;//0-锐角,1-钝角,2-直角

if (a \* a + b \* b > c \* c && a \* a + c \* c > b \* b && b \* b + c \* c > a \* a) {

tri = 0;

}

else if (a \* a + b \* b == c \* c || a \* a + c \* c == b \* b || b \* b + c \* c == a \* a) {

tri = 2;

}

else {

tri = 1;

}

double angle\_ori = acos((a \* a + b \* b - c \* c) / (2 \* a \* b));

Angle -= angle\_ori;

double anglej = acos((a \* a + c \* c - b \* b) / (2 \* a \* c));

double anglej1 = acos((b \* b + c \* c - a \* a) / (2 \* b \* c));

if (tri == 0) {

double temp = 1 / tan(anglej);

double temp2 = 1 / tan(anglej1);

area = (0.125) \* (b\*b\*temp+c\*c\*temp2);

}

else if (tri == 1) {

area = (calc\_facet\_area(\_fhs[i])) - (1 / 8) \* (c \* c \* tan(anglej)) - (1 / 8) \* (a \* a \* tan(anglej1));

}

else if (tri == 2) {

area = (1 / 8) \* ((b \* b) \* (1 / tan(anglej)) + (c \* c) \* (1 / tan(anglej1)));

}

total\_area += area;

}

double GossK = Angle / total\_area;

Gosk.insert(std::pair<VertexHandle,double>{ vh,GossK });

}

}

bool ReaderWriterT<Mesh>::ogl\_writer3(bool \_orient, bool \_smooth) {////在里面把三角面片法向画出

//Mesh::FacetIterator fit(mesh\_->facet\_begin());

std::map<VertexHandle,double> Gosk;

mesh\_->mesh\_process(Gosk);

HalfedgeHandle cshh;

Mesh::FacetIterator fit(mesh\_->facet\_begin());

double minn = 60000, maxn = -60000;

for (std::map<VertexHandle, double>::iterator it = Gosk.begin(); it != Gosk.end(); it++)

{

if (it->second < minn)

minn = it->second;

if (it->second > maxn)

maxn = it->second;

}

double dx = maxn - minn;

double inter = dx / 5;

//glShadeModel(GL\_FLAT);

glShadeModel(GL\_SMOOTH);

int orient = true;// (\_orient) ? 1 : -1;

mesh\_->update\_normals();

for (; fit != mesh\_->facet\_end(); ++fit) {

if ((\*fit).status\_.is\_deleted()) continue;

cshh = fit->halfedge\_handle\_;

FacetHandle fh = mesh\_->facet\_handle(cshh);

const VertexHandle& vh0 = mesh\_->vertex\_handle(cshh);

double in = 80.0 / 20.0;

glBegin(GL\_TRIANGLES);

do {

const VertexHandle& vh = mesh\_->vertex\_handle(cshh);

for (int i = 0; i < 19; i++)

{

if (Gosk[vh] > 40.0)

glColor3f(1.0, 0.3, 0.0);

else if (Gosk[vh] < -40.0)

glColor3f(0.0, 0.3, 1.0);

else if (Gosk[vh] < 40 - in \* i && Gosk[vh]>40 - in \* (i + 1))

glColor3f(1.0 - (i + 1) \* 0.05, 0.35, 0.0 + (i + 1) \* 0.05);

}

glNormal3fv(mesh\_->normal(fh) \* orient);

glVertex3fv(mesh\_->coord(vh));

cshh = mesh\_->next\_halfedge\_handle(cshh);

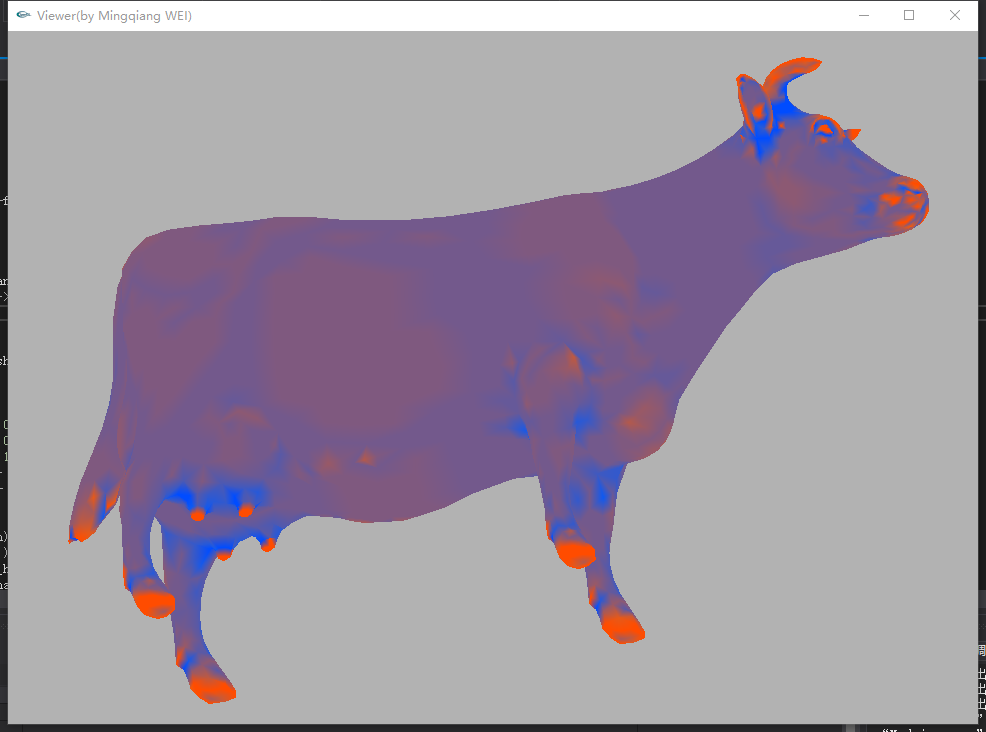
} while (cshh != fit->halfedge\_handle\_);

glEnd();

}

return true;

}

效果如下：

Tips.显红处至显蓝处高斯曲率数值逐渐下降（凹凸性）