|  |
| --- |
| 大连理工大学软件学院 |
| 数字影视后期制作 |
| Progressive Mesh |
|  |
|  |
| **学号：201492317** |
| **姓名：李涵** |
| **17/6/25** |

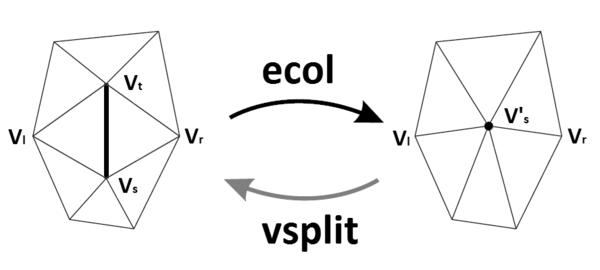
|  |
| --- |
|  |

1. 算法简介

渐进网格是动态细节水平（LOD）的技术之一。 该技术由Hugues Hoppe于1996年推出。该方法使用保存模型的结构 - 渐进式网格，可根据当前视图选择细节级别。 实际上，这意味着可以一次显示具有最低细节水平的整个模型，然后逐渐显示更多的细节。 其中的缺点是相当大的记忆消耗。 优点是它可以实时工作。 逐步网格也可以用于计算机技术的其他领域，例如通过互联网逐渐传输数据或进行压缩。

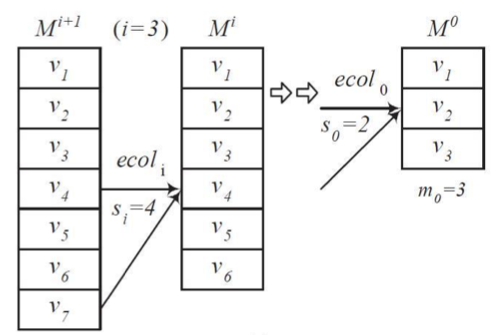
渐进网格是一种作为最佳质量的原始模型而创建的数据结构，简化了合适的抽取算法，该算法逐步删除模型中的一些边缘（边缘折叠操作）。在实际应用中，有必要进行尽可能多的简化，以实现最小的模型。 所得到的模型以完整的质量由最小模型和逆运算序列表示为简单（顶点拆分操作）。这形成了一种层次结构，有助于在所选择的细节层次上创建一个模型。

1.Edge Collapse



网格优化方法随机遍历可能的网格空间，ecol采用两个连接的顶点，并用单个顶点替换它们。 在这个操作过程中，由边缘连接的两个三角形{vs，vt，vl}和{vt，vs，vr}也被删除。

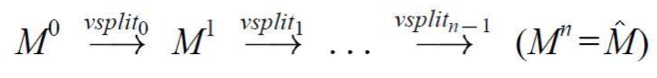




2. Vertex Split

(1)Edge Collapse变换可逆->Vertex Split。顶点分割（vsplit）是将顶点划分为两个新顶点的边缘折叠的反向操作。 因此，会出现新的边缘{vt，vs}和两个新的三角形{vs，vt，vl}和{vt，vs，vr}。

(2)更新转换附近的网格的属性。



3. Geomorphs

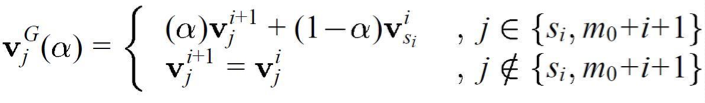
(1) 在两个网格之间创建平滑的视觉过渡



(2)获得混和参数α



(3)线性内插顶点位置

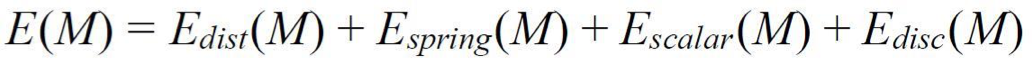


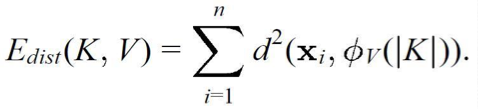
4. Mesh Optimization

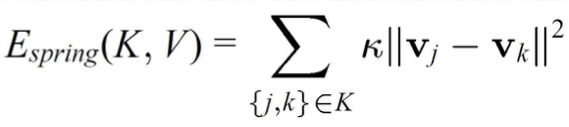
外循环: 算法对于K进行优化，优化了网格的连通性。

内循环: 对点的位置进行优化。

5. Energy Function







1. 核心代码

struct EdgeCollapseTarget {

    EdgeIter ei;

    double cost;

    double optimalCoord[3];

    int id;

    EdgeCollapseTarget(){}

    EdgeCollapseTarget(EdgeIter &ei\_in, double cost\_in, double \*optimalCoord\_in, int id\_in) {

        ei   = ei\_in;

        cost = cost\_in;

        for(int i = 0; i < 3; i++) optimalCoord[i] = optimalCoord\_in[i];

        id   = id\_in;

    }

};

struct VertexSplitTarget {

    EdgeIter ei;

    double   v1OrginalCoord[3];

    bool     v1OriginalIsBoundary;

    vector<CMyHalfEdge\*> halfedgesAroundV0;

    int id;

    VertexSplitTarget(){}

};

class Simplification {

    CMyMesh \*mesh;

priority\_queue <EdgeCollapseTarget, deque<EdgeCollapseTarget>, greater<EdgeCollapseTarget>> heap;

    list<EdgeCollapseTarget> suspendedEdgeCollapseTarget;

    stack<VertexSplitTarget>  vertexSplitTarget;

    stack<EdgeCollapseTarget> readdedEdgeCollapseTarget;

    int ect\_id\_base;

    int n\_active\_faces;

    void AssignInitialQ();

    void CumulateQ(VertexIter &vi, double \*normal, double d);

    void ComputeOptimalCoordAndCost(EdgeIter &ei);

    CMyHalfEdge\* FindBoundaryEdgeIncidentToVertexInCW(CMyHalfEdge \*baseHalfEdge);

    void FindNeighborHalfEdge(VertexIter &v1, vector<FaceIter> &facesOriginallyIncidentToV0OrV1);

    bool IsFinWillNotBeCreated(EdgeIter &ei);

    void RemoveEdge(EdgeIter &ei, double \*optimalCoord, bool isFirstCollapse);

public:

    Simplification(){ ect\_id\_base = 0; }

    void InitSimplification(CMyMesh \*mesh\_in);

    bool EdgeCollapse();

    void VertexSplit();

    void ControlLevelOfDetail(int step);

};

bool operator>(const EdgeCollapseTarget &a, const EdgeCollapseTarget &b){

    return a.cost > b.cost;

}

void Simplification::InitSimplification(CMyMesh \*mesh\_in)

{

    mesh = mesh\_in;

    n\_active\_faces = mesh->n\_faces;

    AssignInitialQ();

    for(EdgeIter ei = mesh->edges.begin(); ei != mesh->edges.end(); ei++)

        ComputeOptimalCoordAndCost(ei);

}

void Simplification::AssignInitialQ()

{

    for(VertexIter vi = mesh->vertices.begin(); vi != mesh->vertices.end(); vi++){

        for(int i = 0; i < 10; i++) vi->Q[i] = 0.0;

        CMyHalfEdge \*startHalfEdge, \*endHalfEdge;

        if(vi->isBoundary == false) startHalfEdge = vi->neighborHe;

        else startHalfEdge =FindBoundaryEdgeIncidentToVertexInCW(vi->neighborHe);

        CMyHalfEdge \*hep = startHalfEdge;

        do{

            CumulateQ(vi, hep->face->ormal, -DotProduct(hep->face->ormal, hep->face->halfedge[0].vertex->coord));

            if(vi->isBoundary && hep->prev->mate == NULL){

                endHalfEdge = hep->prev;

                break;

            }

            hep = hep->prev->mate;

        }while(hep != startHalfEdge && hep != NULL);

        if(vi->isBoundary){

double boundaryVector[3], pseudoNormal[3];

            for(int i = 0; i < 3; i++) boundaryVector[i] = startHalfEdge->next->vertex->coord[i] - startHalfEdge->vertex->coord[i];

            CrossProduct(boundaryVector, startHalfEdge->face->ormal, pseudoNormal);

            Normalize(pseudoNormal);

            CumulateQ(vi, pseudoNormal, -DotProduct(pseudoNormal, startHalfEdge->vertex->coord));

            for(int i = 0; i < 3; i++) boundaryVector[i] = endHalfEdge->next->vertex->coord[i] - endHalfEdge->vertex->coord[i];

            CrossProduct(boundaryVector, endHalfEdge->face->ormal, pseudoNormal);

            Normalize(pseudoNormal);

            CumulateQ(vi, pseudoNormal, -DotProduct(pseudoNormal, endHalfEdge->vertex->coord));

        }

    }

}

void Simplification::CumulateQ(VertexIter &vi, double \*normal, double d)

{

    double a = normal[0];

    double b = normal[1];

    double c = normal[2];

    vi->Q[0] += a\*a;

    vi->Q[1] += a\*b;

    vi->Q[2] += a\*c;

    vi->Q[3] += a\*d;

    vi->Q[4] += b\*b;

    vi->Q[5] += b\*c;

    vi->Q[6] += b\*d;

    vi->Q[7] += c\*c;

    vi->Q[8] += c\*d;

    vi->Q[9] += d\*d;

}

void Simplification::ComputeOptimalCoordAndCost(EdgeIter &ei)

{

    VertexIter v0 = ei->halfedge[0]->vertex;

    VertexIter v1 = ei->halfedge[0]->next->vertex;

    double newQ[4][4];

    newQ[0][0]              = v0->Q[0] + v1->Q[0];

    newQ[0][1] = newQ[1][0] = v0->Q[1] + v1->Q[1];

    newQ[0][2] = newQ[2][0] = v0->Q[2] + v1->Q[2];

    newQ[0][3] = newQ[3][0] = v0->Q[3] + v1->Q[3];

    newQ[1][1]              = v0->Q[4] + v1->Q[4];

    newQ[1][2] = newQ[2][1] = v0->Q[5] + v1->Q[5];

    newQ[1][3] = newQ[3][1] = v0->Q[6] + v1->Q[6];

    newQ[2][2]              = v0->Q[7] + v1->Q[7];

    newQ[2][3] = newQ[3][2] = v0->Q[8] + v1->Q[8];

    newQ[3][3]              = v0->Q[9] + v1->Q[9];

    double matrix[4][4], rhs[4] = { 0.0, 0.0, 0.0, 1.0 }, solution[4];

    for(int i = 0; i < 3; i++) for(int j = 0; j < 4; j++) matrix[i][j] = newQ[i][j];

    matrix[3][0] = matrix[3][1] = matrix[3][2] = 0.0;

    matrix[3][3] = 1.0;

    double cost;

    double optimalCoord[3];

    if( SolveLinearSystem(matrix, rhs, solution) ){

        double temp[4];

        for(int i = 0; i < 4; i++) temp[i] = DotProduct4D(newQ[i], solution);

        cost = DotProduct4D(solution, temp);

        for(int i = 0; i < 3; i++) optimalCoord[i] = solution[i];

    }else{ // matrix is singular. solution is not unique.

        cost = 0.0;

        if(v0->isBoundary) for(int i = 0; i < 3; i++) optimalCoord[i] = v0->coord[i];

        else               for(int i = 0; i < 3; i++) optimalCoord[i] = v1->coord[i];

    }

    if(v0->isBoundary || v1->isBoundary) cost += BOUNDARY\_COST;

    heap.push( EdgeCollapseTarget(ei, cost, optimalCoord, ect\_id\_base) );

    ei->ect\_id = ect\_id\_base;

    ect\_id\_base++;

}

bool Simplification::EdgeCollapse()

{

    if(n\_active\_faces < 3) return false;

    if( readdedEdgeCollapseTarget.empty() == false ){

        RemoveEdge(readdedEdgeCollapseTarget.top().ei, readdedEdgeCollapseTarget.top().optimalCoord, false);

        readdedEdgeCollapseTarget.pop();

        return true;

    }

    list<EdgeCollapseTarget>::iterator ecti = suspendedEdgeCollapseTarget.begin();

    while( ecti != suspendedEdgeCollapseTarget.end() ){

        if(ecti->ei->isActive == false || ecti->id != ecti->ei->ect\_id){

            // obsolete. delete this

            ecti = suspendedEdgeCollapseTarget.erase(ecti);

        }else{

            if( IsFinWillNotBeCreated(ecti->ei) ){

                RemoveEdge(ecti->ei, ecti->optimalCoord, true);

                ecti = suspendedEdgeCollapseTarget.erase(ecti);

                return true;

            }else{

                ecti++;

            }

        }

    }

    EdgeCollapseTarget ect;

    while(heap.empty() == false){

        ect = heap.top();

        heap.pop();

        if(ect.ei->isActive == true && ect.id == ect.ei->ect\_id){

            if( IsFinWillNotBeCreated(ect.ei) ){

                RemoveEdge(ect.ei, ect.optimalCoord, true);

                return true;

            }

            else{

                suspendedEdgeCollapseTarget.push\_back(ect);

            }

        }

    }

    return false;

}

void Simplification::RemoveEdge(EdgeIter &ei, double \*optimalCoord, bool isFirstCollapse)

{

    CMyHalfEdge \*hepCollapse = ei->halfedge[0];

    VertexIter v0 = hepCollapse->vertex;

    VertexIter v1 = hepCollapse->next->vertex;

    hepCollapse->face->isActive = false;

    n\_active\_faces--;

    if(hepCollapse->mate != NULL){

        hepCollapse->mate->face->isActive = false;

        n\_active\_faces--;

    }

    vertexSplitTarget.push( VertexSplitTarget() );

    vertexSplitTarget.top().ei = ei;

    for(int i = 0; i < 3; i++) vertexSplitTarget.top().v1OrginalCoord[i] = v1->coord[i];

    vertexSplitTarget.top().v1OriginalIsBoundary = v1->isBoundary;

    CMyHalfEdge \*startHalfEdge;

    vector<FaceIter> facesOriginallyIncidentToV0OrV1;

    if(v0->isBoundary == false) startHalfEdge = hepCollapse;

    else                        startHalfEdge = FindBoundaryEdgeIncidentToVertexInCW(hepCollapse);

    CMyHalfEdge \*hep = startHalfEdge;

    do{

        facesOriginallyIncidentToV0OrV1.push\_back(hep->face);

        hep = hep->prev->mate;

    }while(hep != startHalfEdge && hep != NULL);

    if(v1->isBoundary == false) startHalfEdge = hepCollapse->next;

    else                        startHalfEdge = FindBoundaryEdgeIncidentToVertexInCW(hepCollapse->next);

    hep = startHalfEdge;

    do{

        facesOriginallyIncidentToV0OrV1.push\_back(hep->face);

        hep = hep->prev->mate;

    }while(hep != startHalfEdge && hep != NULL);

    if(v0->isBoundary == false) startHalfEdge = hepCollapse;

    else                        startHalfEdge = FindBoundaryEdgeIncidentToVertexInCW(hepCollapse);

    hep = startHalfEdge;

    do{

        if(hep->face->isActive){

            hep->vertex = v1;

            vertexSplitTarget.top().halfedgesAroundV0.push\_back(hep);

        }

        hep = hep->prev->mate;

    }while(hep != startHalfEdge && hep != NULL);

    for(int i = 0; i < 3; i++)  v1->coord[i] = optimalCoord[i];

    if(isFirstCollapse){

        for(int i = 0; i < 10; i++) v1->Q[i] += v0->Q[i];

    }

    hepCollapse->edge->isActive = false;

    if(hepCollapse->next->mate != NULL) hepCollapse->next->mate->mate = hepCollapse->prev->mate;

    if(hepCollapse->prev->mate != NULL) hepCollapse->prev->mate->mate = hepCollapse->next->mate;

    hepCollapse->prev->edge->isActive = false;

    if(hepCollapse->next->edge->halfedge[0] == hepCollapse->next)

        hepCollapse->next->edge->halfedge[0] = hepCollapse->prev->mate;

    else hepCollapse->next->edge->halfedge[1] = hepCollapse->prev->mate;

    if(hepCollapse->next->edge->halfedge[0] == NULL){

        if(hepCollapse->next->edge->halfedge[1] != NULL){

            hepCollapse->next->edge->halfedge[0] = hepCollapse->next->edge->halfedge[1];

            hepCollapse->next->edge->halfedge[1] = NULL;

        }else{

            hepCollapse->next->edge->isActive = false;

        }

    }

    if(hepCollapse->prev->mate != NULL) hepCollapse->prev->mate->edge = hepCollapse->next->edge;

    if(hepCollapse->mate != NULL){

        if(hepCollapse->mate->next->mate != NULL) hepCollapse->mate->next->mate->mate = hepCollapse->mate->prev->mate;

        if(hepCollapse->mate->prev->mate != NULL) hepCollapse->mate->prev->mate->mate = hepCollapse->mate->next->mate;

        hepCollapse->mate->next->edge->isActive = false;

        if(hepCollapse->mate->prev->edge->halfedge[0] == hepCollapse->mate->prev)

hepCollapse->mate->prev->edge->halfedge[0] = hepCollapse->mate->next->mate;

        else

            hepCollapse->mate->prev->edge->halfedge[1] = hepCollapse->mate->next->mate;

        if(hepCollapse->mate->prev->edge->halfedge[0] == NULL){

            if(hepCollapse->mate->prev->edge->halfedge[1] != NULL){

                hepCollapse->mate->prev->edge->halfedge[0] = hepCollapse->mate->prev->edge->halfedge[1];

                hepCollapse->mate->prev->edge->halfedge[1] = NULL;

            }else{

                hepCollapse->mate->prev->edge->isActive = false;

            }

        }

        if(hepCollapse->mate->next->mate != NULL) hepCollapse->mate->next->mate->edge = hepCollapse->mate->prev->edge;

    }

    if( hepCollapse->next->edge->isActive == false &&

       (hepCollapse->mate == NULL || hepCollapse->mate->prev->edge->isActive == false) ){

        return;

    }

    if(v0->isBoundary) v1->isBoundary = true;

    FindNeighborHalfEdge(v1, facesOriginallyIncidentToV0OrV1);

    if(v1->isBoundary == false) startHalfEdge = v1->neighborHe;

    else                        startHalfEdge = FindBoundaryEdgeIncidentToVertexInCW(v1->neighborHe);

    hep = startHalfEdge;

    do{

        if(isFirstCollapse) ComputeOptimalCoordAndCost(hep->edge);

        mesh->AssignFaceNormal(hep->face);

        hep->next->vertex->neighborHe = hep->next;

        if(hep->prev->mate == NULL){

            if(isFirstCollapse) ComputeOptimalCoordAndCost(hep->prev->edge);

            hep->prev->vertex->neighborHe = hep->prev;

            break;

        }

        hep = hep->prev->mate;

    }while(hep != startHalfEdge && hep != NULL);

    mesh->AssignVertexNormal(v1);

    hep = startHalfEdge;

    do{

        mesh->AssignVertexNormal(hep->next->vertex);

        if(hep->prev->mate == NULL){

            mesh->AssignVertexNormal(hep->prev->vertex);

            break;

        }

        hep = hep->prev->mate;

    }while(hep != startHalfEdge);

}

CMyHalfEdge\* Simplification::FindBoundaryEdgeIncidentToVertexInCW(CMyHalfEdge \*baseHalfEdge)

{

    CMyHalfEdge \*hep = baseHalfEdge;

    do{

        if(hep->mate == NULL) return hep;

        hep = hep->mate->next;

    }while(hep != baseHalfEdge);

    return hep;

}

void Simplification::FindNeighborHalfEdge(VertexIter &v1, vector<FaceIter> &facesOriginallyIncidentToV0OrV1)

{

    for(unsigned int i = 0; i < facesOriginallyIncidentToV0OrV1.size(); i++){

        if(facesOriginallyIncidentToV0OrV1[i]->isActive == true){

            for(int j = 0; j < 3; j++){

                if(facesOriginallyIncidentToV0OrV1[i]->halfedge[j].vertex == v1) {

                    v1->neighborHe = &(facesOriginallyIncidentToV0OrV1[i]->halfedge[j]);

                    break;

                }

            }

            break;

        }

    }

}

bool Simplification::IsFinWillNotBeCreated(EdgeIter &ei)

{

    CMyHalfEdge \*hepCollapse = ei->halfedge[0];

    VertexIter v0 = hepCollapse->vertex;

    VertexIter v1 = hepCollapse->next->vertex;

    CMyHalfEdge \*startHalfEdgeV0, \*startHalfEdgeV1;

    if(v0->isBoundary == false) startHalfEdgeV0 = hepCollapse;

    else                        startHalfEdgeV0 = FindBoundaryEdgeIncidentToVertexInCW(hepCollapse);

    if(v1->isBoundary == false) startHalfEdgeV1 = hepCollapse->next;

    else                        startHalfEdgeV1 = FindBoundaryEdgeIncidentToVertexInCW(hepCollapse->next);

    CMyHalfEdge \*hepV0 = startHalfEdgeV0;

    do{

        CMyHalfEdge \*hepV1 = startHalfEdgeV1;

        do{

            if(hepV0->next->vertex == hepV1->next->vertex ||

               (hepV1->prev->mate == NULL && hepV0->next->vertex == hepV1->prev->vertex) ){

                VertexIter commonVertex = hepV0->next->vertex;

                if( commonVertex != hepCollapse->prev->vertex &&

                   (hepCollapse->mate != NULL && commonVertex != hepCollapse->mate->prev->vertex) ){

                    return false;

                }

            }

            if(hepV0->prev->mate == NULL){

                if(hepV0->prev->vertex == hepV1->next->vertex ||

                   (hepV1->prev->mate == NULL && hepV0->prev->vertex == hepV1->prev->vertex) ){

                    VertexIter commonVertex = hepV0->prev->vertex;

                    if( commonVertex != hepCollapse->prev->vertex &&

                       (hepCollapse->mate != NULL && commonVertex != hepCollapse->mate->prev->vertex) ){

                        return false;

                    }

                }

            }

            hepV1 = hepV1->prev->mate;

        }while(hepV1 != startHalfEdgeV1 && hepV1 != NULL);

        hepV0 = hepV0->prev->mate;

    }while(hepV0 != startHalfEdgeV0 && hepV0 != NULL);

    return true;

}

void Simplification::VertexSplit()

{

    if(vertexSplitTarget.empty() == false){

        CMyHalfEdge \*hepCollapsed = vertexSplitTarget.top().ei->halfedge[0];

        VertexIter v0 = hepCollapsed->vertex;

        VertexIter v1 = hepCollapsed->next->vertex;

        readdedEdgeCollapseTarget.push( EdgeCollapseTarget(vertexSplitTarget.top().ei, -1, v1->coord, -1) );

        for(int i = 0; i < 3; i++) v1->coord[i] = vertexSplitTarget.top().v1OrginalCoord[i];

        v1->isBoundary = vertexSplitTarget.top().v1OriginalIsBoundary;

        hepCollapsed->edge->isActive = true;

        FaceIter f0 = hepCollapsed->face;

        f0->isActive = true;

        n\_active\_faces++;

        if(hepCollapsed->next->mate != NULL) hepCollapsed->next->mate->mate = hepCollapsed->next;

        if(hepCollapsed->prev->mate != NULL) hepCollapsed->prev->mate->mate = hepCollapsed->prev;

        hepCollapsed->prev->edge->isActive = true;

        if(hepCollapsed->next->edge->halfedge[0] == hepCollapsed->next->mate)

            hepCollapsed->next->edge->halfedge[1] = hepCollapsed->next;

        else

            hepCollapsed->next->edge->halfedge[0] = hepCollapsed->next;

        for(int i = 0; i < 3; i++){

            f0->halfedge[i].vertex->neighborHe = &(f0->halfedge[i]);

        }

        if(hepCollapsed->mate != NULL){

            FaceIter f1 = hepCollapsed->mate->face;

            f1->isActive = true;

            n\_active\_faces++;

            if(hepCollapsed->mate->next->mate != NULL) hepCollapsed->mate->next->mate->mate = hepCollapsed->mate->next;

            if(hepCollapsed->mate->prev->mate != NULL) hepCollapsed->mate->prev->mate->mate = hepCollapsed->mate->prev;

            hepCollapsed->mate->prev->edge->isActive = true;

            if(hepCollapsed->mate->prev->edge->halfedge[0] == hepCollapsed->mate->prev->mate)

                hepCollapsed->mate->prev->edge->halfedge[1] = hepCollapsed->mate->prev;

            else

                hepCollapsed->mate->prev->edge->halfedge[0] = hepCollapsed->mate->prev;

            for(int i = 0; i < 3; i++){

                f1->halfedge[i].vertex->neighborHe = &(f1->halfedge[i]);

            }

        }

        for(unsigned int i = 0; i < vertexSplitTarget.top().halfedgesAroundV0.size(); i++){

            CMyHalfEdge \*hep = vertexSplitTarget.top().halfedgesAroundV0[i];

            hep->vertex = v0;

        }

        for(int i = 0; i < 2; i++){

            VertexIter v\_target;

            if(i == 0) v\_target = v0;

            else       v\_target = v1;

            CMyHalfEdge \*startHalfEdge;

            if(v\_target->isBoundary == false) startHalfEdge = v\_target->neighborHe;

            else                              startHalfEdge = FindBoundaryEdgeIncidentToVertexInCW(v\_target->neighborHe);

            CMyHalfEdge \*hep = startHalfEdge;

            do{

                mesh->AssignFaceNormal(hep->face);

                hep = hep->prev->mate;

            }while(hep != startHalfEdge && hep != NULL);

            mesh->AssignVertexNormal(v\_target);

            hep = startHalfEdge;

            do{

                mesh->AssignVertexNormal(hep->next->vertex);

                if(hep->prev->mate == NULL){

                    mesh->AssignVertexNormal(hep->prev->vertex);

                    break;

                }

                hep = hep->prev->mate;

            }while(hep != startHalfEdge);

        }

        vertexSplitTarget.pop();

    }

}

void Simplification::ControlLevelOfDetail(int step)

{

    int n\_target\_faces = mesh->n\_faces\*pow(0.9, step);

    cerr << "step " << step << " " << n\_target\_faces << endl;

    if(n\_target\_faces < n\_active\_faces){

        while(n\_target\_faces < n\_active\_faces) if(EdgeCollapse() == false) break;

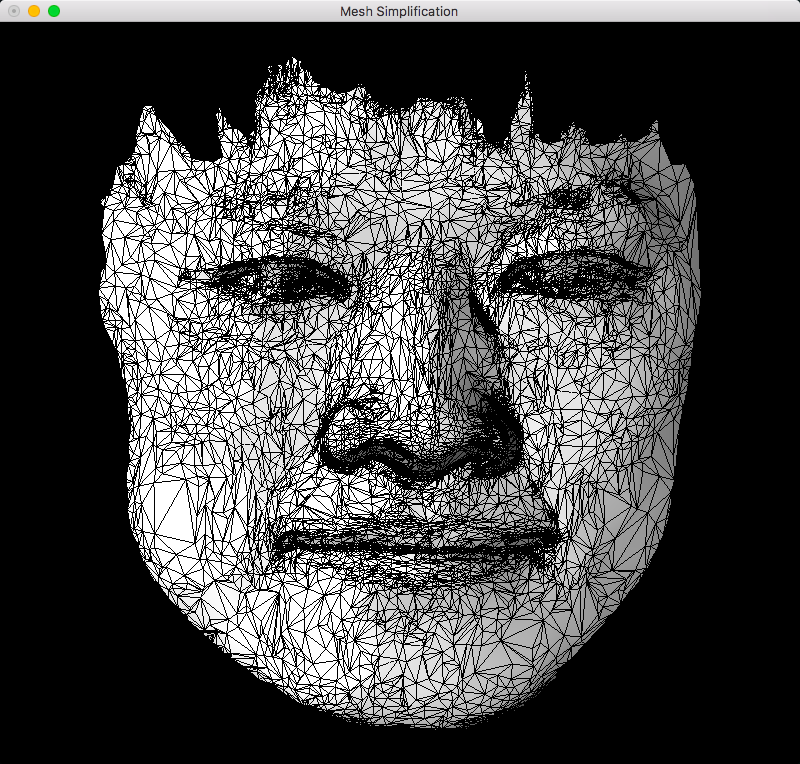
    }else if(n\_target\_faces > n\_active\_faces){

        while(n\_target\_faces > n\_active\_faces) VertexSplit();

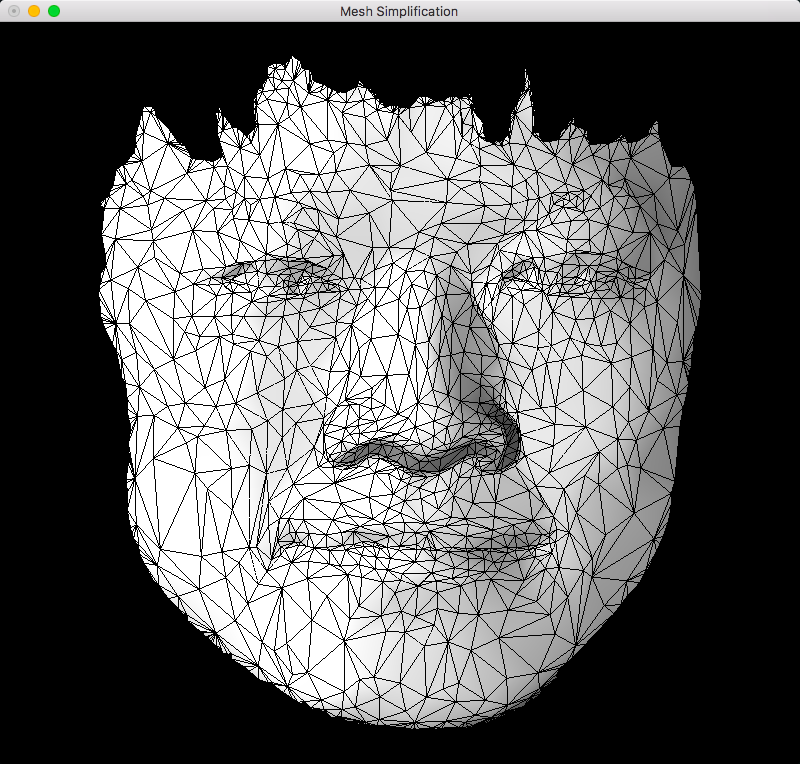
    }

}

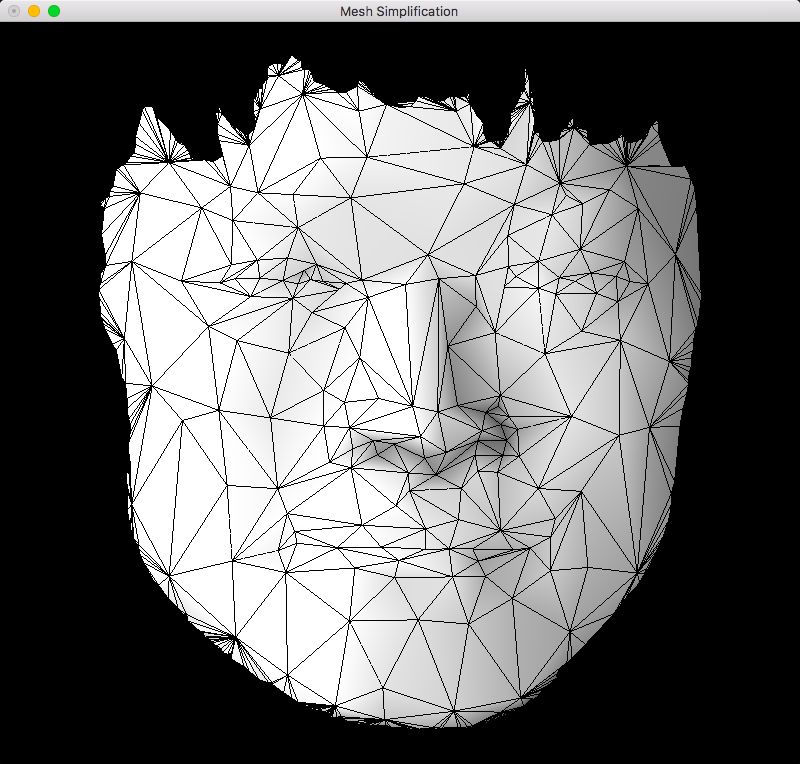
1. 运行结果

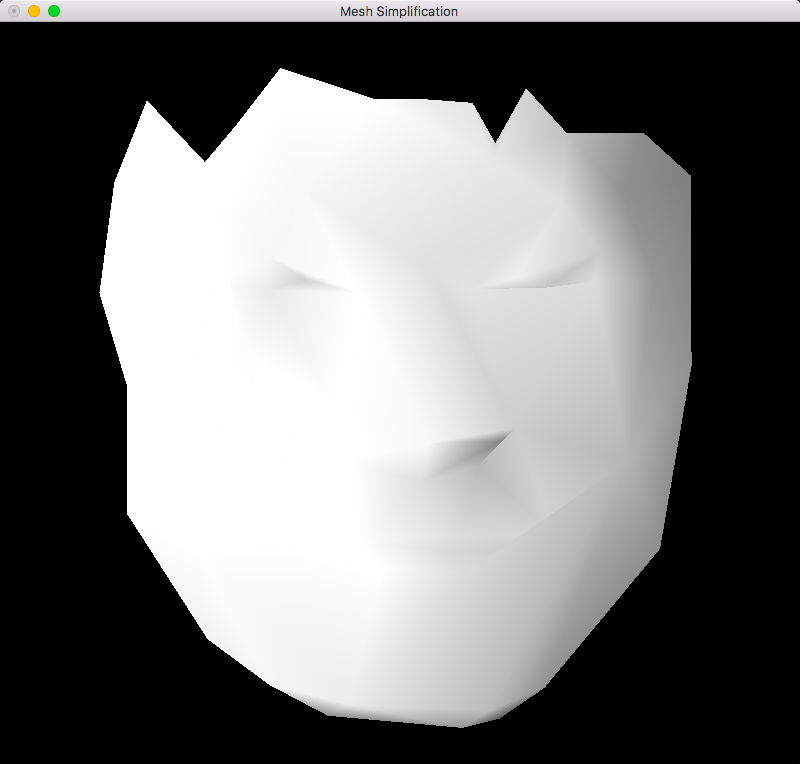
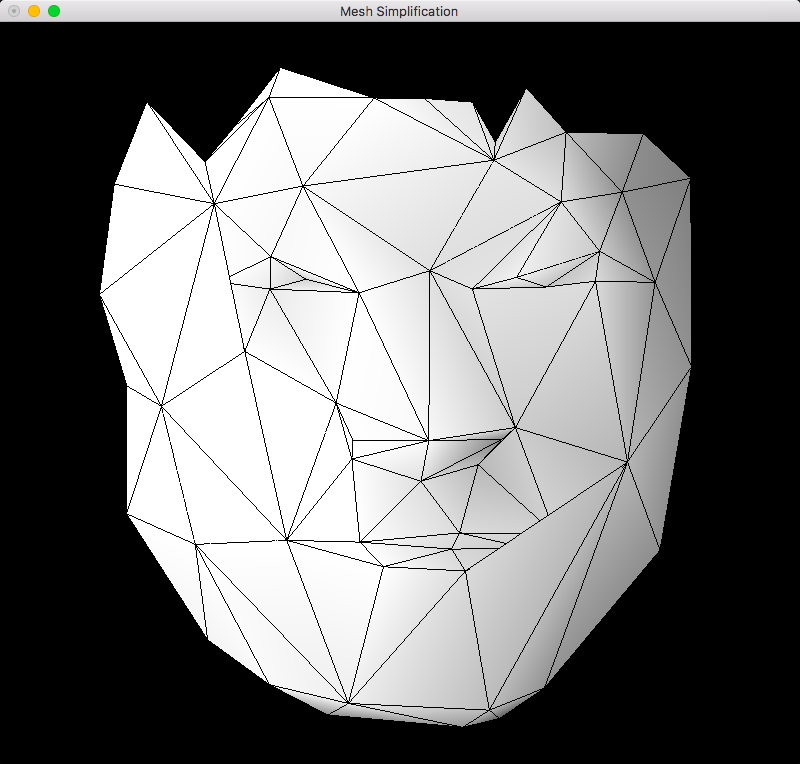
原始脸部数据

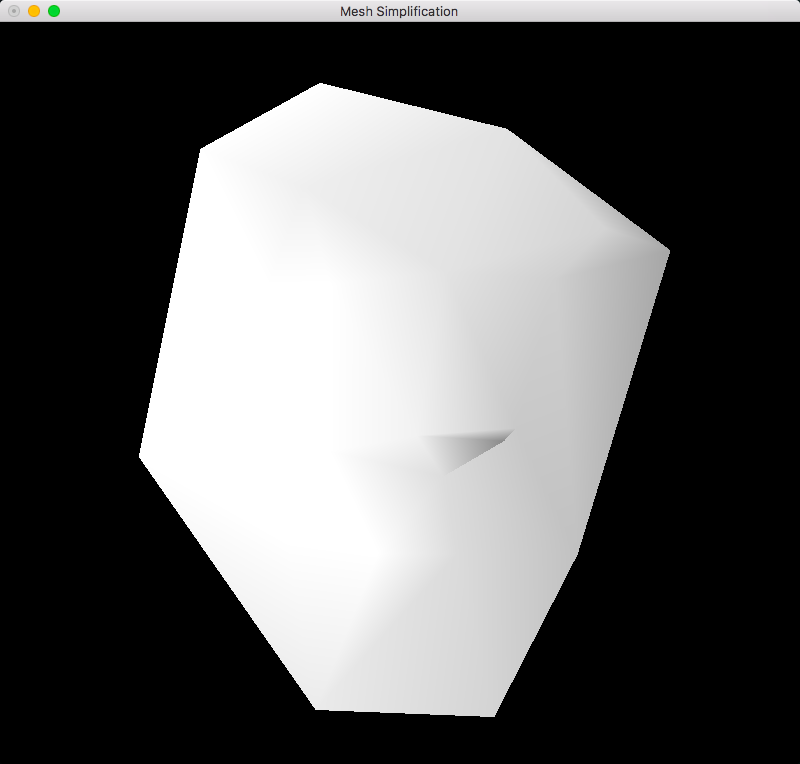
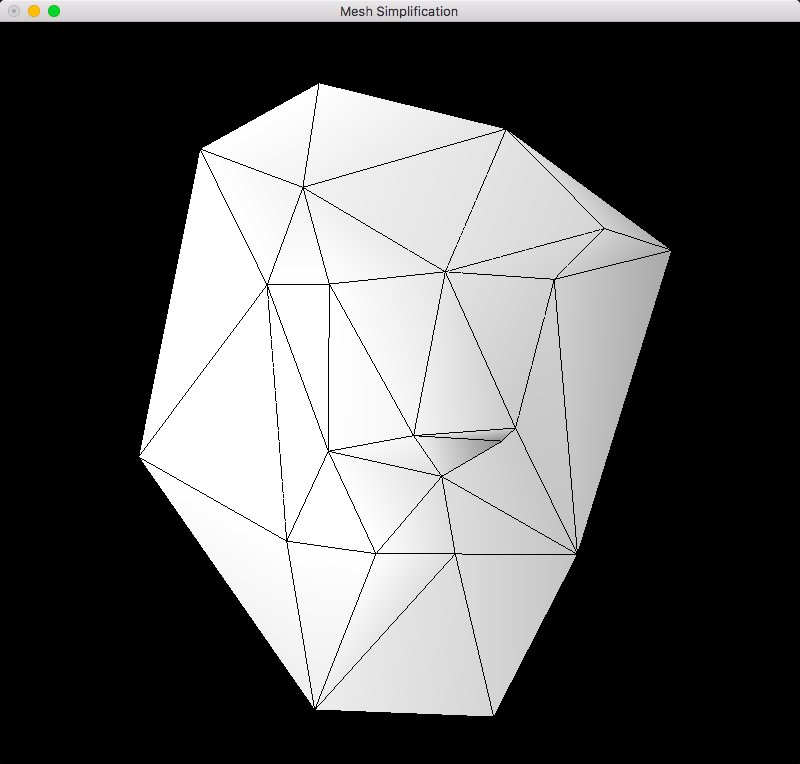
简化20次

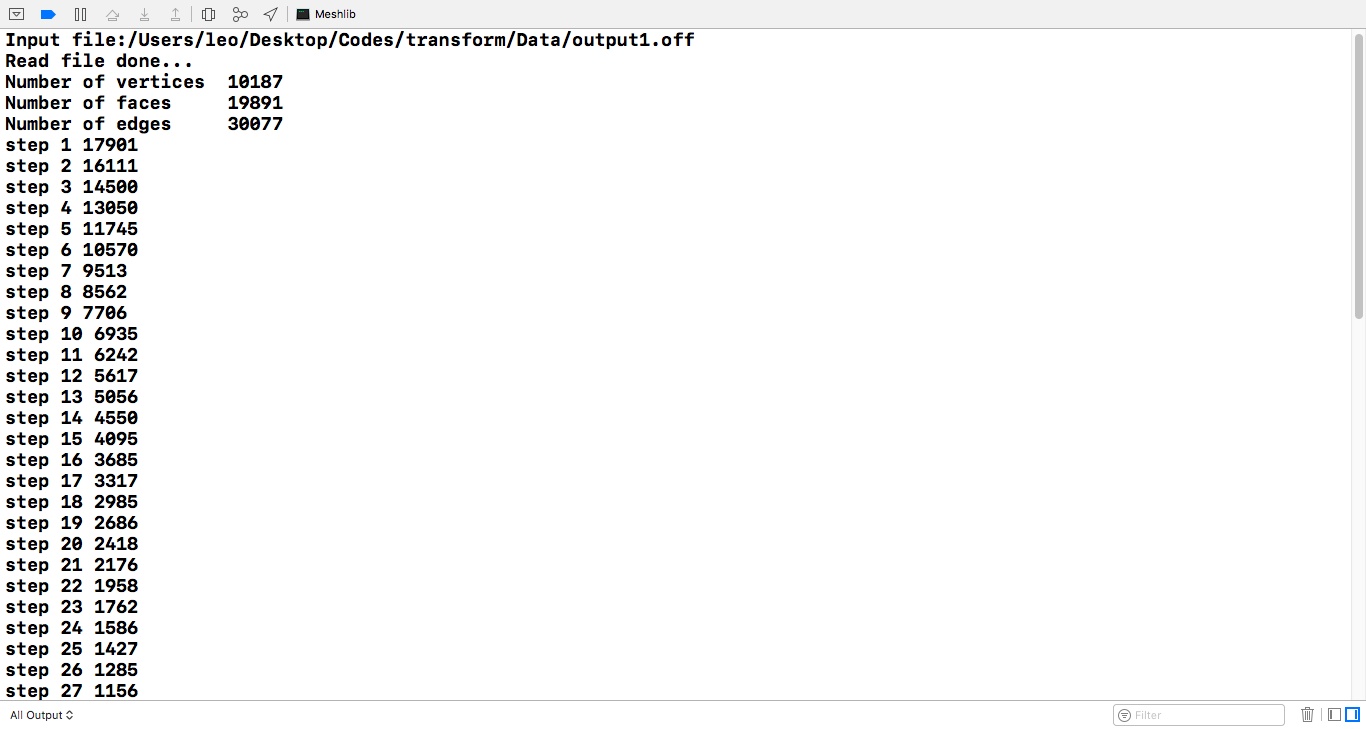
简化30次

简化50次

简化60次



1. 心得体会

**这次的大作业给我带来的最大收获应该就是提升了我的编程能力以及强化了我的编程思维。虽然作业感觉都比较复杂，一拿到手基本都不知道该怎么做，但是仔细读了MeshLib库之后再比对老师课上讲的算法也总是能写的八九不离十，可能真的像助教所说，这门课的编程让我切身的体会到了那句“程序就是算法加数据结构。”的一点含义吧。**

**说实话，可能我今后并不想继续研究这些网格中的三角面片简化之类的问题，但是这门课带给我能力的提升是通用的。**