An Implementation of Halfedge Data Structure in Catmull-Clark Subdivision for 2-Manifold Single-sided Surface

Yu Wang

August 2015

Contents

1 Introduction

Catmull-Clark subdivision is widely applied to construct a smooth surface from an initial mesh of polygons. It is independent of the topology of initial mesh.

2 Halfedge Data Structure

An object in the 3D Euclid space can be modeled as several meshes of polygons. For a single mesh, it comprises three types of geometry elements: vertex, edge, and face. Adjacency data structure is need to store the topological information (adjacency and connectivity) between these elements.
Several adjacency structures have been fully developed, including simple data structure, winged edge data structure (Baumgart, 1975), halfedge data structure (Eastman, 1982), QuadEdge Data structure (Guibas and Stolfi), and FaceEdge Data Structure (Dobkin and Laszlo, 1987).
Among all these data structures, the author chooses halfedge data structure in this project to realize Catmull-Clark subdivision, because 1) the storage size is independent of the mesh topology, and 2) it has a simple implementation. The author also extends its definition to add the ability in dealing with single-sided surfaces (or non-orientable object).

2.1 Vertex, Halfedge, and Face

The definitions and assumptions of vertex, halfedge and face follow the assumption of 2-manifold, as shown in Table 1. Element IDs are unique. When two elements fall into the same group of element, they can not have same ID. (Mobius sibling halfedges are exceptions as we discuss later). A quadrilateral face made with four halfedges and four vertices is shown in Figure 1.

Every element stores two types of information: self-information and adjacency information. As shown in Table 2. The adjacency information include adjacency in a face and adjacency between faces. The adjacency between faces include sibling links and boundary links. (And we will discussion more in the mesh section.)

	Definition	Assumption
Vertex	A 3-dimensional point.	No overlapping vertices exits in a mesh. But overlapping vertices can exist in different meshes.
Halfedge	An edge that starts from one vertex and ends at another vertex.	A halfedge connects exactly two non-overlapping vertices and it has a direction. Less than two halfedges start from the same vertex and end at the same vertex in a single mesh.
Face	A polygon that contains a loop of vertices and halfedges.	A face has at least three non-overlapping vertices so it makes a polygon. The face has to be constructed with a complete loop of halfedges with on openings.

Table 1: Definitions and assumptions of vertex, halfedge, and face

2.2 Mesh

We define a mesh as a collection of basic elements (i.e. vertex, halfedge, and face). Hashtable is implemented to represent the collection in this project, because of its constant serach time for element. We construct three hashtables for all vertices, halfedges, and faces in the mesh respectively. The keys for these hashtable are element IDs and the contents are the element pointers.

The ID of a halfedge is related with the IDs of its start vertex and end vertex. In this project, we define the ID of a halfedge as start vertex ID * maximum number of vertices in a mesh + end vertex ID. This definition will guarantee a unique ID for every halfedge when they have a different start or different end vertex from other halfedges.

A mesh also includes the adjacency and connectivity for elements within. They can be classifed into three groups: 1) halfedge flow in an single face, 2) face connections, and 3) boundary connections.

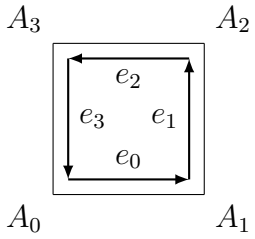

Figure 1: A quadrilateral face made with four halfedges

Element	Self-Information	Adjacency Information
Vertex	1. vertex ID 2. vertex position 3. vertex normal	1. one outgoing halfedge 2. on mobius connection?
Halfedge	1. edge ID	1. start and end vertex 2. link to parent face 3. predecessor and successor in the parent face 4. sibling links to adjacent face 5. boundary links to adjacent face
Face	1. face ID 2. face normal	1. one side halfedge

Table 2: Definitions and assumptions of vertex, halfedge, and face

2.2.1 Halfedge Flow in One Face

A face is constructed by a loop of consecutive halfedges. The start of one halfedge is the end of its previous halfedge, and the end is the start of its next halfedge. Every halfedge contains two pointers, pointing to its previous and next halfedge respectively. Every vertex in this face will also have a pointer to its outgoing halfedge.

2.2.2 Face Connections and Sibling Links

There are two types of face connections, the normal connection and the mobius connection, as shown in Figure 2. In a typical halfedge data structure, with the assumption of double-sided surface, a pair of halfedges between two

faces are defined with opposite direction. We extends this idea to represent single-sided surface by adding another type of connection, named as mobius connection. In a mobius connection, a pair of halfedges are in same direction. The vertex on a mobius connection will also be marked for the purpose of vertex traversal in the future. In the example of Figure 2, on the top, e_1 and e_1' are siblings to each other. On the bottom, e_1 and e_1' are mobius siblings to each other.

One thing to point out is that mobius sibling halfedges have the same element ID. Because they have the start vertex and end vertex. However, we could check for the mobius sibling pointers in order to find all halfedges in the edge traveral of a mesh.

With the extension of mobius connection, there can be three different adjacent situations for a halfedge in a mesh: 1) it is on a normal connection and has a normal sibling, 2) it is on a mobius connection and has a mobius pointer, and 3) it lies on the boundary of the surface and does not have a sibling pointer nor a mobius poniter.

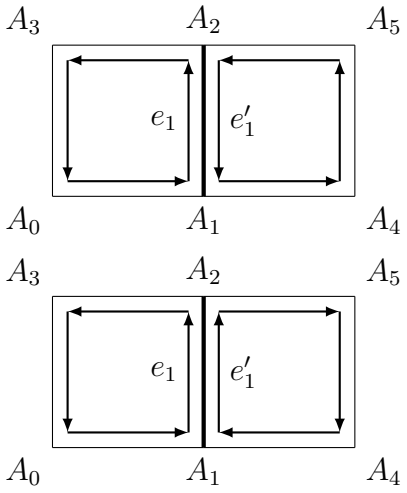

Figure 2: Normal connection (up) and mobius connection (down) between two faces

2.2.3 Boundary and Boundary Links

For halfedges on the boundary of a mesh, we connect them with boundary links. In a surface with no mobius connetion, the boundary halfedges follows

a continous flow. We can traverse the boundary when starting at one vertex, following the natural flow on the boundary, and ending at the starting vertex. Previous boundary pointerss and next boundary pointers are created to link these halfedges. When mobius connection occurs in a mesh, however, some adjacent boundary halfedges will start or end at same vertex, which blocks the natural flow of boundary halfedges. If this happens, we build mobius boundary pointers to link these halfedges. Figure X shows an example of boundary halfedges with and without mobius connections.

¡Add normal boundary links figure and mobius boundary link figures here!¿

2.2.4 Build Mesh from Elements

As a conclusion of adjacency in a mesh, we need four steps in order to construct a mesh from basic elements.

Step 1: Create individual vertices. We create instances of vertices with their position and ID. The position of two vertices can be the same but their ID should always be unique.

This step takes O(V) time, where V is the number of vertices in the mesh.

Step 2: Construct indivdual faces. To build an indivdual face, we create consecutive instances of halfedges, with their start and end vertices. Meanwhile, every vertex will be asigned a pointer to its outgoing halfedge when we create halfedges. For each halfedge, We then add the previous and next pointers to its previous halfedge and next halfedge respectively.

This step takes O(E) time, where E is the number of halfedges in the mesh.

Step 3: Build sibling links. For every halfedge, we need to find if there exist a sibling or mobius sibling from other faces in this mesh. If its start vertex is same with the end of another halfedge, and its end vertex is same with the start of another halfedge, we then find a sibling. If it has the same start and end vertex with another halfedge, we then find a mobius sibling.

In this project, with the implementation of hashtable, the ID of a halfedge is related with its start vertex ID and end vertex ID. The mobius sibling link is actually generated in step 2. When we create the halfedge, if its ID is equal to a halfedge that we created before, we know they are mobius siblings. The search of normal sibling is also in constant time because we can calculate the ID of its sibling halfedge knowing that their start and end vertex are reversed.

This step takes O(E) time, where E is number of halfedges in the mesh.

Step 4: Build boundary links. After step 3, if a halfedge does not have a sibling or mobius sibling, it lies on the boundary of this mesh. We need to build the boundary links to these halfedges.

To do this, we need a counter to keep track of how many times we cross a mobius connection. We 1) set counter equal to zero and start from one halfedge on the boudary, 2) we go to its next halfedge if the counter is even, or its previous halfedge if the counter is odd, 3) we then go to its the sibling or moibus sibling, the counter increase by 1 if it is a mobius sibling, 4) check if the current halfedge is on boundary, if not, we repeat 2) and 3) until we reach to one, and 5) this boundary halfedge shares one vertex with our last boundary halfedge, so we can build boundary links between them, and 6) we repeat 1) to 5) to build bounary links until we reach the starting boundary halfedge in 1). From 1) to 6), one boundary loop of this mesh is built. And we move on to build other loops by repeating 1) to 6), until every boundary halfedges have boundary links to its adjacent boundaries.

This step takes O(E) time, where E is the number of halfedges in the mesh.

Now this mesh contains everything that we need to start a Catmull-Clark subdivision. In total, it takes O(E) time from step 1 to step 4.

2.3 Mesh Operations

Building from basic elements is not the only way to create the initial mesh for Catmull-Clark subdivison. A new mesh can also be made by the following mesh operations: 1) copy a mesh, 2) 3D transformation of a mesh, and 3) merging the boundaries for two meshes.

2.3.1 Copy a Mesh

In order to make a copy, we create one element instance for every element that belongs to the original mesh. The copy of a mesh will keep the adjacency for elements and positions of vertices. Therefore, new element instances remain exact same adjacency information as the original mesh, except that the element ID is different and pointers are pointing to the new instances.

2.3.2 Mesh Transformation

In a 3D tranformation, the positions for all vertices in the mesh will perform tranformation. The adjacency of original mesh remains the same, so halfedges and faces will transform at the same time while vertices transform. Typically, for linear transformations, we mutiply the transformation matrix to the position of every vertex from the oringal mesh.

2.3.3 Merging Meshes

Meshes can be merged into a new mesh if sibling links can be made between some of the boundary halfedges that belongs them. This merge of boundary halfedge will keep the new mesh as a 2-manifold. We also want the elements ID from these meshes stays different, so no collision occurs after we merge them together.

Two types of mesh merging are implemented in this project: automatic merging and manual merging. In automatic merging, we define a very small tolerance value. If any pair of vertices on the boundary of the two meshes have a distance smaller than the tolerance, we check if their boundary halfedges can be merged with sibling links. If they do, we merge them, continue to trace along the boundaries of these two meshes and merge boundary halfedges until we can't.

In manual merging, we force to merge boundaries from two meshes even if the distance of their vertices are larger than tolerance. In practice, we have four ways to perform the manual merging. Assume we would like to force merge the bounday of Mesh 1 and Mesh 2, we can apply one of the following stratergies:

1) Vertex positions in mesh 1 remain the same after merging. The boundary faces on Mesh 2 extend to the boundary of Mesh 1.

2) The opposite of 1).

3) Use the arithmetric mean position for vertex from Mesh 1 and Mesh 2 as the final vertex position after merging. The boundary faces from Mesh 1 and Mesh 2 both extend to these new vertex positions.

4) Build new faces between the two meshes and the vertex positions for Mesh 1 and Mesh 2 remain the same.

2.4 Mesh Traversals

Catmull-Clark subdivision also requires two types of traversals in a mesh: 1) traversal around a face, and 2) traversal around a vertex. Traversal around a face is necessary to build face points and calculate face normals in the face. Traversal around a vertex is necessary to build vertex points and calculate vertex normals in the face.

2.4.1 Traversal Around Face

The traversal around a face lead to all the edges and vertices belong to this face. It starts from one side halfedge of this face, follows the halfedge flow, and ends at starting the halfedge of the traversal. Traversals of all faces in a mesh takes O(E) time, where E is the number of faces in the mesh.

2.4.2 Traversal Around Vertex

The traversal around a vertex lead to all edges and faces that contains this vertex. The traversal of a vertex need to consider two issues: 1) is this vertex on a boundary, and 2) is this vertex on a mobius connection. This makes four different types of vertex traversals. See Figure 3 - 6 for examples.

In the vertex traversal without boundary and mobius issue, we start from one outgoing halfedge of this vertex. We continue to go to the next outgoing halfedge by going to the successor of its sibling until we hit the first outgoing halfedge. In the example of Figure 3, if start and end at halfedge e_1, the sequence of traversal is: e_1 (sibling link to) e_1' (successor link to) e_2 (sibling link to) e_2' (successor link to) e_3 (sibling link to) e_3' (successor link to) e_4 (sibling link to) e_4' (successor link to) e_1.

In order to address the issue of a vertex on boundary, instead of using sibling links, we use boundary links. In the example of Figure 5, we have a boundary e_4' to e_2. If start and end at halfedge e_1, the sequence of traversal is: e_1 (sibling link to) e_1' (successor link to) e_2 (boundary link to) e_4' (successor link to) e_1.

To address the issue of vertex on a mobius connection, instead of using normal links, we use mobius links. At the same time, we switch between the successor and predecessor to the sibling every time we hit a mobius connection. In the example of Figure 3, e_1 to e_1' and e_3 to e_3' have mobius siblings rather than normal sibling. If start and end at halfedge e_1, the sequence of traversal is: e_1 (mobius sibling link to) e_1' (predecessor link to)

e_2 (sibling link to) e'_2 (predecessor link to) e_3 (mobius sibling link to) e'_3 (successor link to) e_4 (sibling link to) e'_4 (successor link to) e_1.

If boundary and mobius connection both occur, we use a combination of the two methods above. In the example of Figure 3, e_1 to e'_1 and e_3 to e'_3 have mobius siblings rather than normal sibling and we have a mobius boundary connection between e'_4 and e_2. If start end end at halfedge e_1, the sequence of traversal is: e_1 (mobius sibling link to) e'_1 (predecessor link to) e_2 (mobius boundary link to) e'_4 (successor link to) e_1.

As a summary of the four different situations above, in a vertex traversal, we 1) start the traversal from with one outgoing halfedge of the vertex, 2) go to sibling or boundary link halfedge, 3) go to the next or previous halfedge that contains the vertex as one end, and 4) repeat 2) and 3) until we reach to the starting outgoing halfedge. This vertex traversal runs in O(E) time, where E is the total number of halfedges in the mesh.

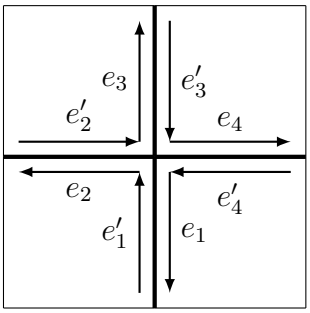

Figure 3: Vertex traversal without boundary and without moibus connection

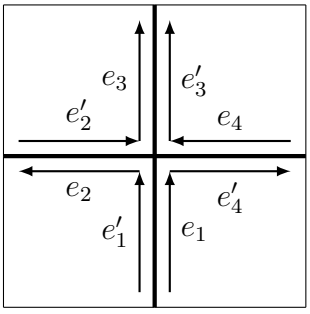

Figure 4: Vertex traversal without boundary and with moibus connection

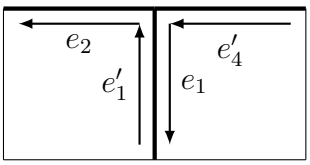

Figure 5: Vertex traversal withboundary and without moibus connection

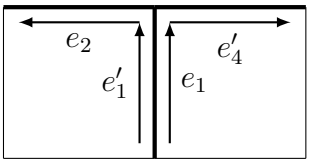

Figure 6: Vertex traversal with boundary and with moibus connection

3 Catumll-Clark Subdivision

Catmull-Clark subdivision is a recursive call on a mesh that we build in section X. Four each level of Catmull-Clark subdivision, we divide every polygon face in the mesh into N quadalaterial faces, where N is the number of halfedges of the polygon face. The new vertices for these sub-faces can be classifid into three groups: 1) face points, 2) edge points, and 3) vertex points. We also need to add adjacency information to new sub-faces so that they be subdivided for the next level. Therefore, each level of subdivision is done in two steps: 1) compute the positions of new vertices, and 2) compile a new mesh given the new vertices.

3.1 Compute New Vertex Positions

There are three steps to compute the positions of vertices in the new mesh: 1) make new face points, 2) make new edge points, and 3) make new Vertex Points. Apart from finding the positions for these new vertices, we also need to assign unique IDs for them so no collision will occur for the new mesh.

3.1.1 Face Points

Face points are related with faces from the mesh. For every face in the mesh, the position of its face point is defined as the average for positions of all vertices belong to this face. If we label the face point as f and vertices of

this face as v_i, the equation to calcuate face point is,

$$v_f = \frac{v_1 + v_2 + ... + v_n}{n}$$

Therefore, in order to get the face point position, we can do a traversal around a face, find all vertices and get the average of their positions.

This step takes O(E) running time, where E is the number of halfedges in the kth level subdivision mesh.

3.1.2 Edge Points

Edge points are related with halfedges from the mesh. In this project, we used the idea from (Reference) and define the sharpness of a halfedge as either infinite sharp or smooth. Depends on the position and sharpness of the edge, we have two ways to calculate the edge point.

If a halfedge does not lie on the boundary, it has a sibling link or mobius link to anthoer halfedge. For the halfedges that are not marked as sharp, the position of edge point of a halfedge is the average of its start vertex position, end vertex position, the face point position of the face it belongs to, and the face point position of the face its sibling or mobius sibling belongs to. If we label the edge point as v_e, the start and end vertices as v_1 and v_2, the face point of its face is v_3 and the face point of its sibling face is v_4, the equation to calcuate edge point is,

$$v_e = \frac{v_1 + v_2 + v_3 + v_4}{4}$$

If a halfedge lie on the boundary or it is marked as sharp, the position of the edge point is the average of its start vertex position and end vertex position. If we label the edge point as v_e, the start and end vertices as v_1 and v_2, the equation to calcuate edge point is,

$$v_e = \frac{v_1 + v_2}{2}$$

This step takes O(E) running time, where E is the number of halfedges in the kth level subdivision mesh.

3.1.3 Vertex Points

Vertex points are related with vertices from the mesh. In order to find the vertex point, we traverse around an oringal vertex in the mesh, find the

information we need from adjacent faces and halfedges and count the nuber of sharp halfedges we cross in the traversal. The number of sharp halfedges linking with this vertex determines methods to calculate vertex points.

1) A vertex with three or more incident sharp edges is called a corner, the new vertex point has same position with the original vertex. If we label the vertex point as v_p, the original vertex as v_1, the equation to calcuate vertex point is,

$$v_p = v_1$$

2) A vertex with two incident sharp edges is called a crease vertex. If we label the vertex point as v_p, the original vertex as v_1 and the two halfedges are labeled with v_1v_2 and v_1v_3. the equation to calculate vertex point is,

$$v_p = \frac{v_2 + 6v_1 + v_3}{8}$$

3) A vertex with less than two sharp edges is a normal vertex. There are two different approaches in calculating the vertex point for a normal vertex. We label the vertex point as v_p, the original vertex point as v_1, the average for all midpoints of edges that contains the original vertex is v_2, the average for all edge points of edges that contains the original vertex is v_2', the average of the face points of all faces adjacent to the old vertex point as v_3, and the number of faces adjacent to the vertex as n, and the number of edges adjacent as n'. Catmull-Clark (Reference) defined vertex point as

$$v_p = \frac{(n-3)v_1 + 2v_2 + v_3}{n}$$

While DeRose (Reference) defined vertex point as

$$v_p = \frac{(n'-2)v_1 + v_2' + v_3}{n'}$$

Catmull-Clark's equation used the average of all midpoints of incident edges and has one more weight on it and one less weight on the original vertex. DeRose's equation used the average of all edge points incident to the vertex. Meanwhile, in a mesh with boundary, n and n' would be be difference by 1. However, the actual difference between these two equations is very small.

In the example of Figure 7, if we do one more level of calculation and represent the new vertex point only with the original vertices, the results from Catmull-Clark and DeRose equations are,

$$v_p = \frac{18}{32}v_0 + \frac{6}{64}(v_2 + v_4 + v_5 + v_7) + \frac{2}{128}(v_1 + v_3 + v_6 + v_8)$$

and,

$$v_p = \frac{27}{32}v_0 + \frac{3}{64}(v_2 + v_4 + v_5 + v_7) + \frac{3}{128}(v_1 + v_3 + v_6 + v_8)$$

respectively. We can see that Catmull-Clark has more weights on the immediate neighbour vertices v_2, v_4, v_5, and v_7, while DeRose has much more on the original vertex. In this project, we used DeRose's equation to calcuate vertex point for a normal vertex.

This step takes O(E) running time, where E is the number of halfedges in the kth level subdivision mesh.

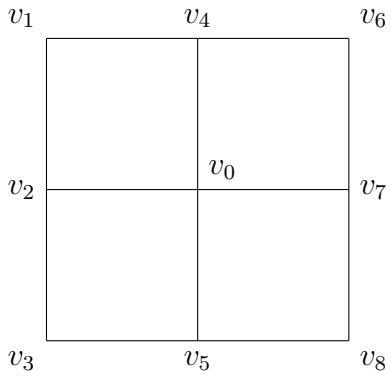

Figure 7: Example of difference of Catmull-Clark and DeRose Equation

3.2 Compile a New Mesh

The newly generated vertices also need to be linked with each other. We compile the new mesh with traversal around face for every face from the original mesh.

For every halfedge v_iv_j in the face, we create four new halfedges. If we denote its edge point of the current halfedge as v_e and the face point of its face as v_f, the four new halfedges are v_iv_e, v_ev_j, v_ev_f, and v_fv_e. An example

in shown in Figure 8. The halfedge v_1v_2 generated v_1v_e, v_ev_2, v_ev_f, v_fv_e. We also add the following adjacency information while generating sub-halfedges:

1) Add one outgoing halfedge pointer to every new vertex. If the new halfedge is on mobius connection, mark the new vertex on mobius connection.

2) Add previous and next pointers to the sub-halfedges.

3) Add sibling pointers to every pair of v_ev_f and v_fv_e.

4) v_iv_e and v_ev_j need to inherit the sharpness and boundary feature from their parent halfedge,

5) If v_iv_j is not on boundary, we add sibling links between v_iv_e and the corresponding sub-halfedge of its parent's sibling and add sibling links between v_ev_j and the corresponding sub-halfedge of its parent's sibling.

6) If v_iv_j is on the boundary and has boundary links with its boundary neighbour halfedges, we add boundary links for v_iv_e and the corresponding sub-halfedge of its parent's boundary neighbour. We also add boundary links for v_iv_e and the corresponding sub-halfedge of its parent's boundary neighbour.

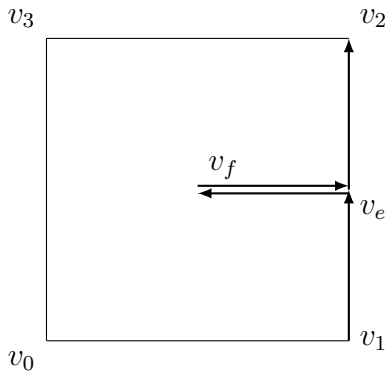

Figure 8: Create four sub-halfedges for one halfedge in subdivision.

When the traversal around face is done for every face in the level k subdivision mesh, the level k + 1 subdivision mesh would be created.

This step takes O(E') running time, where E' is the number of halfedges in the (k + 1)th level subdivision mesh.

As a summary, the Catmull-Clark at kth level takes O(E') running time, where E' is the number of halfedges in the (k + 1)th level subdivision mesh.

4 Offset Mesh

The offset with value offVal for a mesh M contains three parts: 1) Positive offset, where we translate every vertex of M by offVal along the direction of its vertex normal, 2) Negative offset, where we translate every vertex of M by offVal along the opposite direction of its vertex normal, and 3) the cover mesh between positive offset and negative offset, which is the extrusion for M's boundaries along positive and negative vertex normal direction.

In two-sided surfaces, the positive offset mesh is always seperated from the negative mesh. However, in single-sided surfaces, the positive offset mesh will be connected with the negative mesh at the mobius connection of the original mesh.

In order to get the offset mesh, we need two steps: 1) calculate vertex normals for all vertices from the original mesh, 2) build positive and negative offset meshes, and 3) build cover mesh between positive and negative offsets.

4.1 Compute Vertex Normals

The vertex normal for a vertex is commonly defined as the average for the surface normals of all faces adjacent to the vertex. To get the surface normal for a face of N polygon, we use Newell's Method.

4.1.1 Newell's Method and Surface Normals

Newell's method defines the surface normal as the normalized average for cross products of all consecutive halfedges in the face. For a polygon face with n halfedges, we denote them as v_0v_1, v_1v_2, ...,$v_{n-1}v_0$, where $n \geq 3$. The surface normal N_f is,

$$N_f = normalize(\sum_{i=1}^{n-2}(v_i - v_{i-1}) \times (v_{i+1} - v_i) + (v_{n-1} - v_{n-2}) \times (v_0 - v_{n-1}) + (v_0 - v_{n-1}) \times (v_1 - v_0))$$

Therefore, we calculate surface normals for every face in the mesh by traversal around face. It takes O(E) running time, where E is the number of halfedges in the mesh.

4.1.2 Vertex normals

We calculate vertex normals by traversal around vertex. If a vertex v is not on a mobius connction, we denote the m faces adjacent to vertex v are $f_0, f_1, ..., f_m$, the vertex normal N_v is,

$$N_v = normalize(\sum_{i=1}^{m-1} N_{f_i})$$

However, when v is on mobius connection, the halfedge flows for some adjacent faces are in opposite directions (clockwise vs counter clockwise). It leads to surface normals in oppsite directions. We need to fix this issue by introducing a mobius counter. In the vertex traversal, we start by adding the surfaces normals when we find a face. When we cross a mobius connection, we add the negative surface normals for the upcoming faces until we cross another mobius connection.

Another way to address this mobius vertex issue is to check the dot product of every surface normal with the sum that we have got. If the dot product is positive, we add this surface normal, if not, we add its negative surface normal.

Calculating all vertex normals takes O(E) running time, where E is the number of halfedges in the mesh.

4.2 Positive and Negative Offset Meshes

For a mesh with no mobius connection, the positive and negative offset meshes will have no connection with each other. We get the positive offset by copy the mesh, and translate every vertex along its normal with an offset value.

For negative offset, we still want to copy the mesh but the halfedge flow in the faces is in opposite directions. We then translate every vertex along the opposite direction of its normal with an offset value.

When there is a mobius connection, positive offset mesh and negative offset will join each other at the mobius connection. Normals of vertices on mobius connection may not reflect the direction for its positive translation.

In this project, we make the translations of vertex positions with traversal around face. If a vertex is on mobius connection, we can check the dot product of its normal and the face normal for the face it belongs to. If the dot product is positive, we translate along this vertex normal for its positive

offset and the opposite for its negative offset. If the dot product is negative, we translate along this vertex normal for its negative offset and the opposite for its positive offset.

This step takes O(E) running time, where E is the number of halfedges in the mesh.

4.3 Cover Mesh

We also need to build the cover mesh to connect positive offset with negative offset. For a mesh with no mobius connection, we traverse along its boundary. For every halfedge v_iv_j on the boundary, we build a quadrilateral face $v_{posJ}v_{posI}v_{negJ}v_{posJ}$, where v_{posI} and v_{posJ} are the positive offset for v_i and v_j, and v_{negI} and v_{negJ} are the negative offset for v_i and v_j.

As we mentioned before, normals of vertices on mobius connection may not reflect the direction for its positive translation. For a vertex v_i on mobius connection, we need to check the dot product of its normal with the surface normal that this boundary halfedge belongs to. If the dot product is positive, we use v_i's vertex normal to calcuate v_{posI} and negative vertex normal to calculate v_{negI}. If the dot prodcuct is negative, we use v_i's vertex normal to calcuate v_{negI} and negative vertex normal to calculate v_{posI}. After this check for v_i and v_j, we build the quadrilateral face $v_{posJ}v_{posI}v_{negJ}v_{posJ}$.

This step takes O(E) running time, where E is the number of boundary halfedges in the mesh.

The offset mesh will be the combination of the positive offset, negative offset and cover mesh that we build above. In total, it takes O(E) running time to build the offset mesh.

5 Input and Output

6 Test Cases and Discussions

7 Future Researches