TNM079: Modeling & Animation

Laboratory report

Half-edge implementation

Teacher: Björn Gudmunsson

Farhad Mobasher Fard
Farmo306@student.liu.se

Contents

Abstract

This paper is explaining method of implementation different shape by half-edge data structure instead of the simple mesh formats. To implement the half-edge, for each triangle three vertices should be defined, then each vertex and its half edge pair will be set. Then Create a face, connect it to one of the edges finally connect the inner edges to the newly created face.

In the next step the neighbor faces and neighbor vertices for each vertex should be found. The vertex normal, surface area and volume of a mesh will be calculated. Implementation and visualization of curvature will be described. Finally computation of the genus and number of shell will be explained.

Half-edge data structure is a powerful method to represent a mesh and improve the speed of calculation for a mesh.

1. Introduction

Most common way to represent a mesh is based on triangles. Triangles are the smallest geometric shape, Easy to understand and use and well-adapted with the graphics hardware. One way to describe a triangle mesh is that define a float number for each vertex and with three vertex represent a triangle mesh. It is clear that this data structure has a large degree of redundancy because surly some triangles have common edges. In addition of the huge amount of memory, this structure will not give the random access to different vertex. So to find neighboring triangle of a vertex, all the pervious vertices should be traced. By all of these, although this method of representation is fast in rendering phase, it is not useful. An alternative way to illustrate a triangle mesh is half-edge data structure. Half-edge data structure help to gain more efficient access to neighborhood, also add edge information to the mesh.

Why it called half-edge? Each edge divided into two parts. One inner edge which represents the face that located in the left side of the edges. The outer edge which called pair edge represents the right face of current face. So each edge stores the information about its left face, tracing in a triangle is possible by using next and previous pointer and accessing to the next face is possible by the edge's pairs.

2. Method

As mentioned before, half-edge data structure is an efficient way to implement a triangle to represent a mesh. To implement a half-edge data structure, different steps should be done sequentially which will describe in following steps:

2-1. Half-edge implementation

- In the first step three vertices should be defined, which introduce three point of a triangle (Fig. 1). Each vertex has an index number which can be accessible with its index.

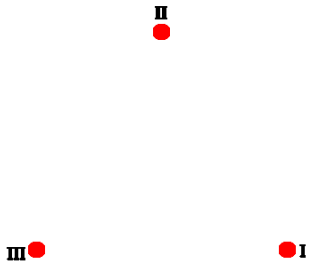


Fig 1: Three vertices in order to represent a triangle. Each vertex has a unique index.

- Then by passing the index number of each two vertices, an edge will be defined between those two vertices (Fig. 2-Left). Now each edge has an index which presents the inner edge of half-edge data structure. Also there is an extra index for each edge which presents the outer edge or pair edge for each edge (Fig. 2-Right)

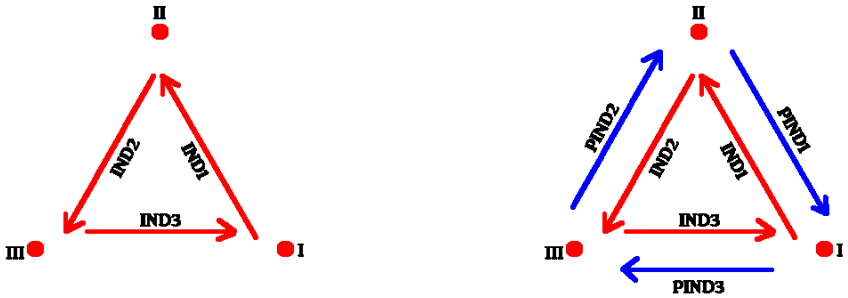


Fig 2: Left: Define three edges between the vertices. Right: Define three pair edge for each inner half edge.

- In the next step the next and pre pointer of each vertex should be assigned (Fig. 3). By doing this, it will be clear for each edge which edge is the next edge and which one is its previous one. In other word, it means to form an inner ring in the triangle.

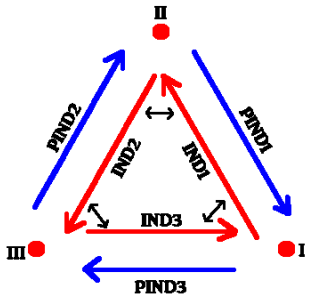


Fig 3: Forming the inner ring for each face.

- Then a triangle as a face should be created and connect to one of edges (Fig. 4). It is important to calculate the normal for this face which will be used later.

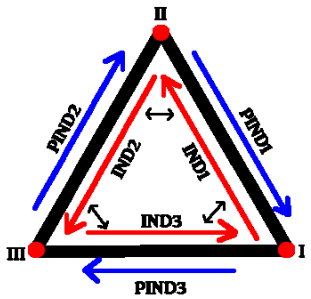


Fig 4: Create a face and connect it to one of half-edge.

- Finally all the three inner edges should be linked to the new face (Fig. 5). Do not forget that in the half-edge method, each half edge introduces its left face.

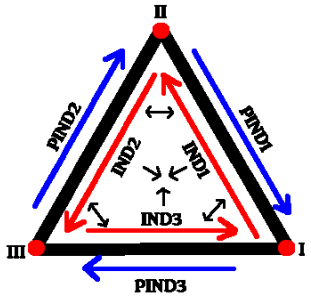


Fig 1: Linking all the three inner half-edge to its left face.

2-2. Find the neighbor vertices and faces

Now a face with its inner half-edge and pair edges was formed. In the next step all the neighbor vertices and faces around a vertex should be found.

- To find all the neighbor vertices around a vertex, the tracing will be started from the input vertex. By a pre pointer from the input point, the first neighbor vertex will be obtained (Fig. 6). Continue with a pair pointer following by another pre pointer the index of second vertex will be found. These steps will be done successively until the current pointer refers to the first vertex. Now all the neighbor vertices around the input vertex were pushed into the stack.

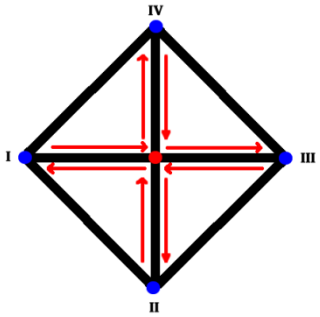


Fig 6: tracing the entire vertices around given vertex to find its neighbor vertices.

- The method to find the neighbor face is the same as method to find the neighbor vertices. Just instead of push the vertices to the stack, the edge index (which presents its left face) should be pushed on the stack. Be aware of the finishing condition of the while loop. Will the pointer be back to the exact point that tracing start?

2-3. Find the vertices normal

The purpose of this step is to find the normal of each vertex (instead of the normal for each face). Equation1 shows how the normal for each vertex can be calculated.

$$\sum_{i=0}^{neighbor\ face\ No.} N_{f(i)}$$

Equation 1 – Calculate the normal for each vertex

It is clear for calculating the normal for each vertex, at first all the neighbor faces of the given vertex should be found, then the neighbor faces normal should be sum and finally the result should be normalized (Fig. 7).

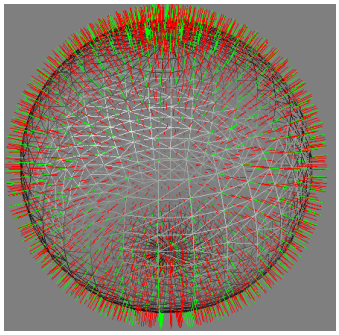


Fig 7: Faces and vertices normal. Red lines are the normal of faces and green are the Normal of faces.

2-4. Calculate surface area of a mesh

Equation2 helps to calculate surface area of a mesh.

$$A = \frac{1}{2} * (V_2 - V_1) * (V_3 - V_1)$$

Equation 2 – Calculate surface area of a face.

V1, V2 and V3 in equation2 are the coordinate of three vertices of each face. By calculate the area of each face. Then the area of a mesh can be calculated by equation3.

$$\sum_{i=0}^{Number\ of\ faces} A_i$$

Equation 3 – Calculate surface area of a mesh.

2-5. Calculate volume area of a mesh

Volume of a shape can be found by using equation4.

$$Vol = \frac{1}{3} \sum_{i=0}^{Number\ of\ faces} \frac{(V_1 + V_2 + V_3)}{3} * A_i * N_{f(i)}$$

Equation 4 – Calculate surface area of a mesh.

Where V1, V2 and V3 are the coordinate of three vertices of a face, A is the area of that face and N is the normal of current face.

2-6. Implement and visualize curvature

Curvature can be used as a measure to find the smooth value of a surface. One way to calculate it is Gaussian curvature (Equetion5).

$$K = \frac{1}{A}(2\pi - \sum_{i=0}^{neighbor\ face\ No.} \theta_i)$$

Equation 5 – Calculate curvature for each vertex.

In equation5, K is a Gaussian curvature for a vertex. A is the area of the entire mesh. And θ is the angle between all the edges that connected to the input vertex.

2-7. Improve the curvature estimate

To improve the curvature estimate, the area of a mesh should be re-calculated by Voronoi area. This method improves the accuracy of this calculation. Equation6 shows the Voronoi area. After calculating area by this formula, it is enough to put the new A in eqeution5 (Fig. 8).

$$A = \frac{1}{8}\left(\sum_{i=0}^{neighbor\ face\ No.} (\cot \alpha_i + \cot \beta_i) * (V_c - V_i)^2 \right.$$

Equation 6 – Formula for Voronoi area.

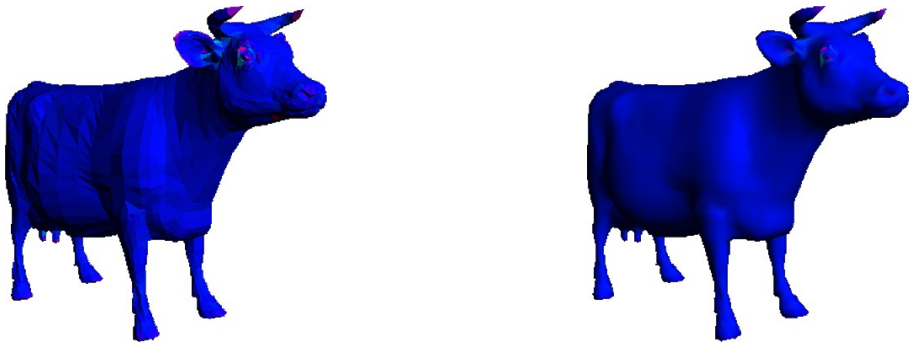


Fig 8: Implement and visualize curvature in a mesh.
Left one is before visualization and right one is after visualization and Improvement of the curvature estimate

2-8. Classify the genus of a mesh

Genus of a mesh shows the number of holes in a mesh. Euler-Poincar´e formula (equation7) helps to calculate the genus of a mesh.

$$V - E + F - (L - F) - 2(S - G) = 0$$

Equation 7 – Euler-Poincar´e formula.

In Euler-Poincar´e formula V shows number of vertices, E is number of edges, F present number of faces, L is number of loops and finally G shows number of genus. Since each face has one loop, the number of loops and faces are equal. Normally number of shell in each shape mentioned as 1. So equation7 can be rewritten as equation8.

$$G = -1 * (\frac{V - E + F - 2}{2})$$

Equation 8 – Modified Euler-Poincar´e formula to find genus(S=1).

In part 2-9 a method will describe to calculate number of shells, so this formula should be rewritten as equetion9.

$$G = -1 * (\frac{V - E + F - (2 * S)}{2})$$

Equation 9 – Modified Euler-Poincar´e formula to find genus(S!=1).

2-9. Compute the number of shells

As mentioned before, number of mesh defined as one. The aim of this part is to calculate the real number of shale in a mesh (Fig. 9).

Three “std::set” was defined.
The first one was included all the vertices in a mesh:

- Vertex: filled by all the vertices of a mesh.
- Qvertex: in each loop, one vertex will be push on it. The loop will be finished when Qvertex is empty.
- Tagvertex: compare the vertex index in the Qvertex with its neighbors and add the new vertices to Qvertex.

```
while(!vertex.empty()) //Do ..... While all the vertices of mesh be traced
{
    vertexit =  vertex.begin();
    Qvertex.insert(*vertexit); //push the first vertex of mesh into Qvertex
    while(!Qvertex.empty()) // Do ..... While Qvertex is not empty
    {
        Qit = Qvertex.begin();
        tagvertex.insert(*Qit); // Push the first element of Qvertex to tagvertex
        oneRing = FindNeighborVertices(*Qit); // Find all the neighbors of the element in tagvertex
        for (unsigned int i = 0 ; i < oneRing.size() ; i++) // For all the neighbors of that the element in tagvertex
        {
            if( tagvertex.find(oneRing.at(i)) == tagvertex.end())
            {
                Qvertex.insert(oneRing.at(i)); //if it was in the tagvertex do nothing, else add it to Qvertex
            }
        }
        cleanit = vertex.find(*Qit);
        vertex.erase(cleanit); // Remove the element from vertex
          tmpit=Qit;
        Qvertex.erase(tmpit); //Remove the element from Qvertex
    } //end While
  shell++; //Inc Shell No.
}//End Main while
return shell;
}
```

Fig 9: Finding Number of Shells in a mesh – C++ Program

Fig 7 shows the code that calculates number of shells in a mesh. Each vertex will be push to a new set and pop from main set. If Qvertex is empty then number of shells will be incremented otherwise all the neighbors of it will be found and will be added to Qvertex (just if do not find in Qvertex). By this function, all the connected neighbor vertices will be traced and if there is another vertex which did not trace, then it will calculated as a new mesh (Fig 10).

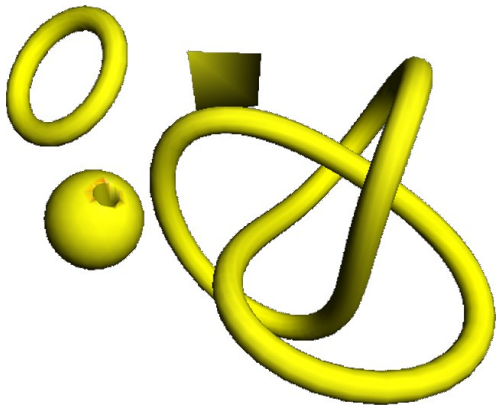


Fig 10: A mesh with 3 Genus and 4 Shells.

3. Result

3-1. Half-edge vs. Polygon mesh

As mentioned before, Half-edge data structure is a powerful format for tracing a mesh. It is a time efficient data structure especially for complex mesh.

As a test, three different shapes were loaded two times. First time by simple mesh structure and then by half-edge data structure (Table 1). Loading time was stored by a stop watch. The result shows the difference between the times of loading for different shape. The winner is Half-edge. The interesting point is that the difference of loading time is not a linear function. More complex mesh will be taken more time than a simpler mesh to render in mesh format.

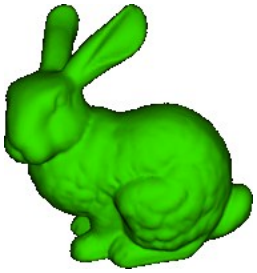
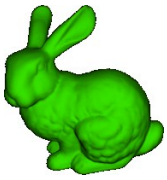
Mesh	Object Name	Number of Faces	Loading Time (mm:ss:hs)	
			Half-edge	Simple mesh
	Bunny Large	280256	00:18:86	02:08:17
	Bunny Medium	70064	00:05:13	00:23:82
	Bunny Small	35032	00:02:38	00:05:73

Table 1: Different rendering time for different shapes with half-edge and mesh

3-2. tracing a mesh

Tracing a mesh is easy and meaningful in half-edge data structure. Working with pointer to pass the next, pre and pair edge makes it fast and simple to implement complex algorithm in meshes and find the adjacency.

3-3. Disadvantages

There are some disadvantages to use half-edge data structure.

- As explain before and according to Euler-Poincar´e formula (Equation 7) half-edge data structure can used in both closed and open mesh but the mesh should be manifold.
- Half-edge data structure uses more memory than a simple polygon mesh.

4. Conclusion

Overall half-edge data structure is a powerful method to represent a mesh. Although this data structure uses more memory than simple mesh polygon, It is easy to trace and find the adjacency of a vertex, calculate the normal of faces and vertices and to calculate area and volume of a mesh. So it is good idea to implement and represent meshes with half-edge data structure.

Since I have completed all the * and ** tasks in addition one *** (Compute the number of shells), I should get grade 5.