# Half-Edge Mesh Data Structure - Lab 1 TNM079

Jonathan Bosson
jonbo665@student.liu.se

Wednesday 20th July, 2016

## Abstract

The paper discusses how mesh data structures work and how to implement a half-edge mesh data structure. It also takes up physical attributes such as normal, curvature and volume and methods to compute these with the mesh data structure.

## 1 Introduction

There are many different kind of mesh data structures. A simple one would be the TriangleSoup, where triangles may have an arbitrary number of vertices and both vertices and edges can be shared by abritrary triangles. However, this method is not structured and there is no simple way to access a neighbouring triangles. Instead we introduce the half-edge data structure that is a more organised data structure at the cost of more memory. The implemented half-edge structure assumes a manifold mesh which defines a volume.

## 2 Background

The half-edge data structures is built on three different classes. Vertex, Halfedge and Face which together form a Mesh. The Vertex defines the position in space a vertex point will have along with a pointer to its edge. A face represents a full triangle, connecting three vertices with edges. The Halfedge has pointers to a vertex and a face, pointers to the next and previous edges as well as a pointer called pair that will point to the opposite half-edge that belongs to the neighbouring face.

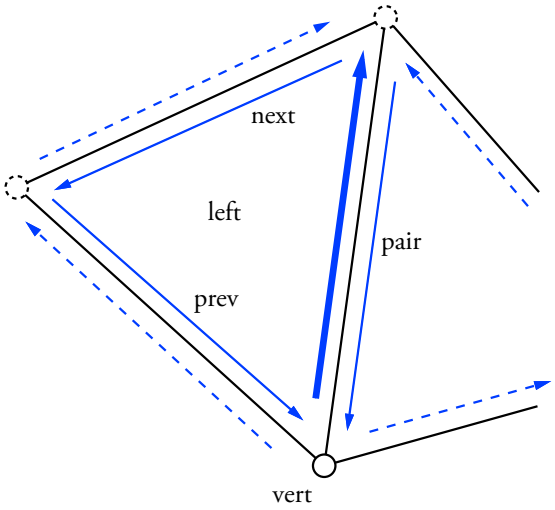


Figure 1: A sketch of the half-edge mesh structure.

1 visualises the data structure in a face. The bolded blue line is the current edge from where we can access another edge with *next*, *prev* or *pair*, current face with *left* as well as the vertex the edge points towards with *vert*.

The normal vector is a simple geometric differential which will point perpendicular to the local surface. Given three vertices *v1*, *v2* and *v3* with counter clockwise orientation we can compute the plane normal as the cross product of $(\mathbf{v}_2 - \mathbf{v}_1)$ and

$(\mathbf{v}_3 - \mathbf{v}_1)$. This generates the discussed normal vector. Another property of the normal is that a approximate of the area of its face can be computed cheaply with $\frac{1}{2}|(v2 - v1) \times (v3 - v1)|$.

$$\mathbf{n} = (\mathbf{v}_2 - \mathbf{v}_1) \times (\mathbf{v}_3 - \mathbf{v}_1) \tag{1}$$

However, this normal computation causes linear properties in the mesh. There are many different techniques to minimise the linear properties but the one implemented in the lab is called mean weighted equally (MWE) and is defined as the sum of all neighbouring face normals called the 1-ring neighbourhood.

$$\mathbf{n}_{v_i} = \widehat{\sum_{j \in N_1(i)}^{n}} \mathbf{n}_{f_j} \tag{2}$$

An analytical computation of the volume of the manifold mesh is not possible but the volume can be sufficiently approximated through Gauss' Theorem.

$$\int_S \mathbf{F} \cdot \mathbf{n} \, dA = \int_V \nabla \cdot \mathbf{F} \, d\tau \tag{3}$$

The theorem describes the relation between the surface integral of a vector field times the unit normal and the volume integral of the divergence of the same vector field. Since this is true for any vector field we can choose a field with constant divergence.

$$\begin{aligned} \nabla \cdot \bar{\mathbf{F}} &= \nabla \cdot (x, y, z) \\ &= \frac{\partial x}{\partial x} + \frac{\partial y}{\partial y} + \frac{\partial z}{\partial z} \\ &= 3 \end{aligned} \tag{4}$$

Since the volume integral will compute $3V$ we can approximate the volume according to equation 5 where $\mathbf{v}_1$, $\mathbf{v}_2$ and $\mathbf{v}_3$ are the vertices of the $i$:th face. The error of this approximation is directly related to the area of the largest face.

$$3V = \sum_{i \in S} \frac{(\mathbf{v}_1 + \mathbf{v}_2 + \mathbf{v}_3)_{f_i}}{3} \cdot \mathbf{n}(f_i) A(f_i), \tag{5}$$

The curvature is one of the most important mesh quality measures and it is used in many algorithms and it describes the smoothness of the mesh. Mathematically this is computed as how much the normal at point $p$ changes as we move $p$ over the surface. There are two very frequent schemes to determine the curvature of the mesh is *gaussian* or *mean curvature*. In this lab was only the gaussian curvature implemented which is defined as $K = \kappa_1 \kappa_2$.

To implement gaussian curvature we use the following formula:

$$K = \frac{1}{A} \left( 2\pi - \sum_{j \in N_1(i)} \theta_j \right). \tag{6}$$

which describes it as the deviation from $2\pi$ weighted by the area of the 1-ring neighbourhood. Note that gaussian curvature cannot determine if the surface is convex or concave but only give a measure of how smooth or non-smooth the area is. A positive value states how pointy the surface is, zero describes a saddle point or a plane. A negative value implies that the surface is concave in one direction and convex in the other.

# 3 Results

## 3.1 Generating the mesh

To finish the implementation of the mesh the functions *AddVertex*, *AddFace* and *AddHalfEdgePair* had to be completed. *AddVertex* simply adds a vertex point with a position and an index to find it in the mesh. If a vertex already exists on the current position will the function exit and return the index of said vertex.

*AddHalfEdgePair* will genereate two HalfEdges on each call between two vertex points, each other's pair. Apart from the pair pointer, each HalfEdge has a pointer towards its face and vertex, a next- and a prev-pointer to the HalfEdges that belongs to the same face. Assuming the HalfEdgePair between sent in vertices doesn't already exist will all mentioned pointers be linked. Furthermore must the vertex' HalfEdge pointer be connected with the

newly generated one. Once done will both edges be pushed into the HalfEdge list.

*AddFace* will first add the vertices and HalfEdges that exists in the triangle by calling on previously implemented functions. Furthermore will it connect the inner ring by linking each corresponding HalfEdge with it's next-, prev- and pair-halfedge. The face then links its pointer to any HalfEdge existing within the triangle. The face is added to the triangle list, and it's normal is calculated through the function *FaceNormal(indx)*. Lastly will all HalfEdges that belongs to the face use the triangle list index the face has as their face pointer.

## 3.2 Implement neighbour access

For efficient access of neighbours have two functions been implemented that are based around making a list of indices of either surrounding faces or vertices, this list is called the 1-ring. Both *FindNeighborFaces* and *FindNeighborVertices* gets a vertex index. The *FindNeighborVertices* will then step through the 1-ring using the pointers to prev and pair HalfEdge, through each iteration will the index for each vertex be stored in the 1-ring list. The *FindNeighborFaces* is very similar but instead of accessing the vertex for each HalfEdge that is processed will each face be saved. Both functions exits their loops and returns the 1-ring list once a it returns to the same HalfEdge it started with.

## 3.3 Calculations of the mesh

The normal of a vertex is calculated by taking the sum of all face normals in the 1-ring of faces around the vertex and thereafter normalising the result.

The mesh surface area is calculated in the *Area()* function as previously mentioned $\frac{1}{2}|(v2 - v1) \times (v3 - v1)|$. The sum of all faces' surface area in the mesh represents the total surface area.

Just like the *Area()* function is the *Volume()* function computed as a sum of all existing faces in the mesh. Each face approximated volume is calculated through equation 5. The sum divided by 3 results in a good estimation of the physical volume of the model.

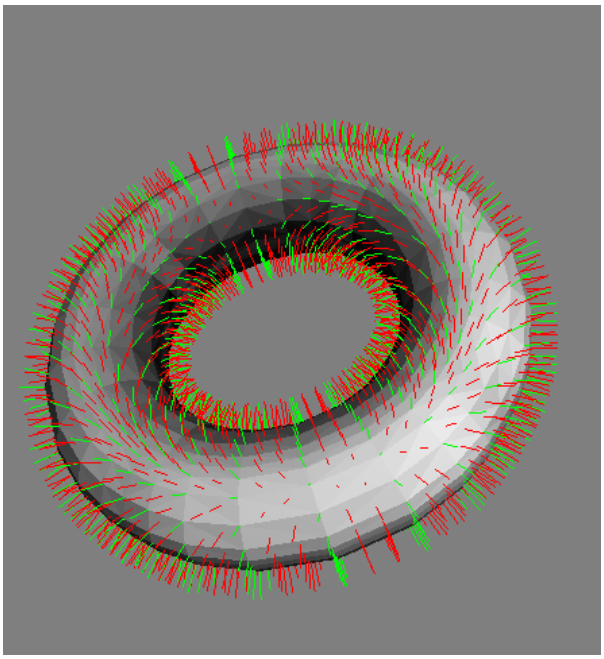

Figure 2: Vertex normal visualised on a Torus

The gaussian curvature is calculated in the *FaceCurvature()* function. Each face curvature is assumed to be the average curvature of all of its vertex points. The curvature of each vertex is calculated through equation 6 by going through the 1-ring of vertices the vertex is connected to. The angle is computed through a scalar product of the vector that goes between the center vertex and two neighbouring vertices in the 1-ring.

With the vertexcurvature can a smoother shader be applied than previous flat shading. However as seen 4, does the gaussian curvature have some issues with specific areas of the mesh. One reason for this is since the calculated curvature becomes higher the more dense the triangles are. This is seen by the black circle on the top of the sphere.

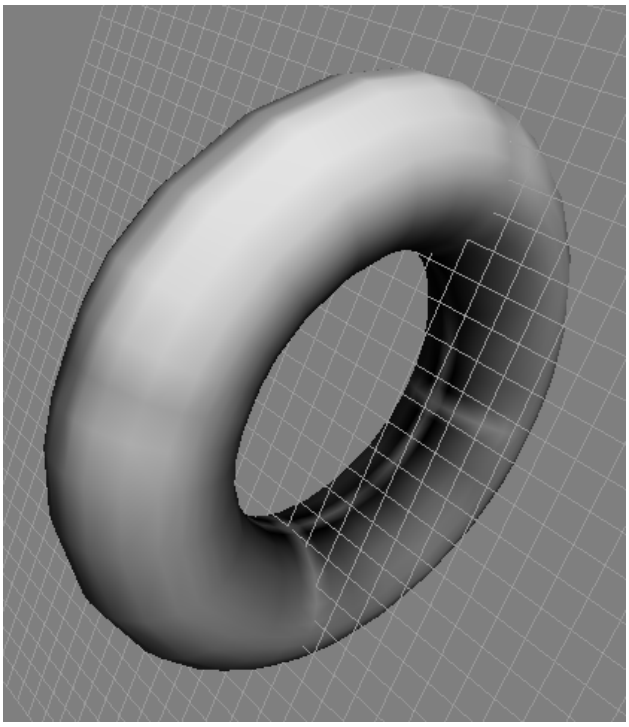

Figure 3: Curvature making for smoother shading.

### 3.4 Reference list

## References

[1] M. E. Dieckmann, *Lecture Slides for TNM079*, **Lecture 1, 2**, 2016.

This report aims for grade 3.

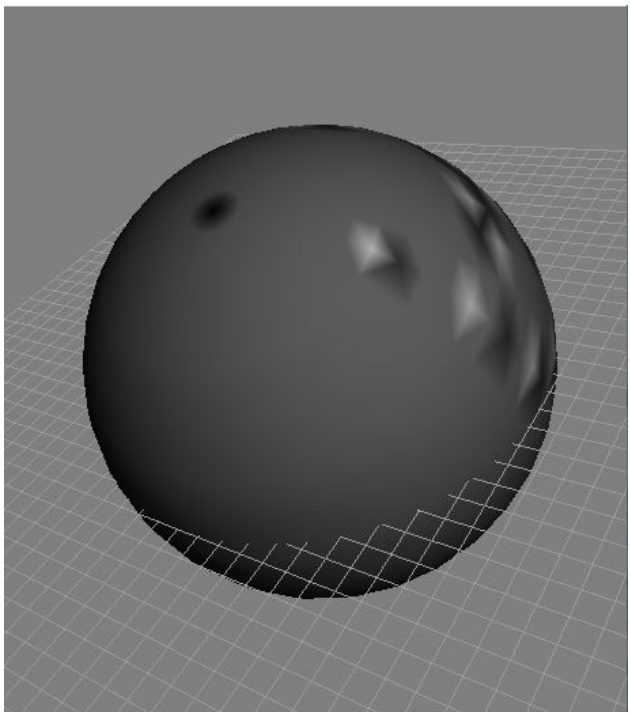

Figure 4: Vertexcurvature causing some odd artifacts in the shading