THE HALF-EDGE DATA STRUCTURE

MODELING AND ANIMATION

Dan Englesson
danen344@student.liu.se
Sunday 12th April, 2011

Abstract

In this lab assignment which was done in the course TNM079, Modeling and animation, the half-edge mesh data structure was implemented, which solves the access to neighboring faces that is lacking from an ordinary face- and vertex-list data structure. Because of the ease of getting the neighboring faces the calculation of vertex normals was easily implemented as well as methods for calculating the total area,volume and genus of the entire mesh were also implemented. Furthermore a naive method for calculating the curvature estimate with Gaussian curvature was implemented and it was also improved by dividing with the voronoi area instead. Further improvement was made by implementing the mean curvature and smoothing was done by moving the vertices's positions according to the mean curvature and the vertex normal. A description of how to reduce the half-edge mesh data structure to save memory is also presented in this report and what benefits and disadvantages it will bring.

1 Introduction

This lab was done in the course TNM079 Modeling and animation in spring 2011. The aim of the lab was to implement the half-edge mesh data structure and also implement methods for calculating meshes surface area, volume, genus and curvature. Getting the neighboring vertices and faces is often needed during calculations for example calculating the vertex normal or the curvature estimate, which both needs all its neighboring faces. The half edge data structure is very successful at retrieving information about the neighboring faces, which makes it a very attractive and useful data structure for meshes. The half edge data structure also handles large meshes much better than a simple mesh data structure.

2 Method

The lab assignments were graded with stars and assignments with one star, (*), was mandatory in order to pass the lab. One assignment with two stars (**) had to be completed to get a grade four and for grade five one needed to pass the requirements for grade four and also complete a assignment with three stars (***). Here follows all the completed assignments:

2.1 Implement the half edge mesh (*)

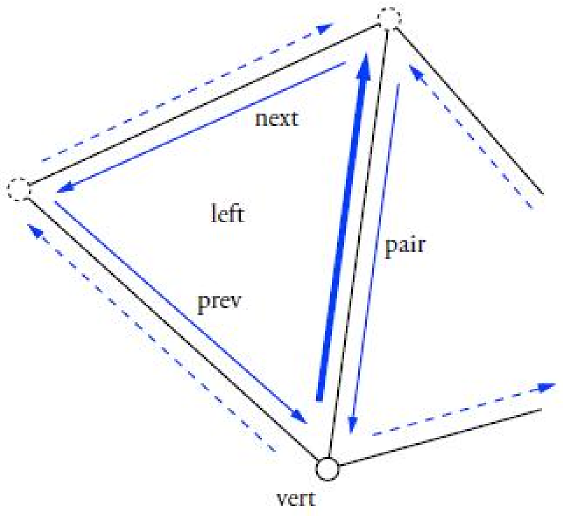

Figure 1. The half edge data structure. Image courtesy of [1]

Because the lack of neighboring face access in a simple mesh data structures, this half edge mesh data structure was implemented. For each vertex there is a half edge pair-, next- and previous- pointer and also the left face is stored. So to move around a face one uses the next and previous pointers and to go to the right face, the half-edge pair is used which is an inner,solid blue arrows, and outer edge, dashed blue arrows, connected by pointers, see figure 1.

The half edge data structure was implemented in HalfEdgeMesh.cpp, and here follows the things that needed to be done to implement the half edge data structure.

1. Firstly one needs to associate the three vertices to the face, with the AddFace function. The vertices is added to the vertex list mVerts if and only if they are unique and the AddVertex function returns the index of the found or newly inserted vertex.
2. Create all the half-edge pairs in the triangle, which was done with the AddHalfEdgePair. The addHalfeEdgePair takes in two vertices and returns the index of the half edge pairs.
3. Then connect the inner-ring by assigning the next and previous pointers to the corresponding edges in the triangle.
4. The face is then created and the half-edge pair and normal is connected to the face. The normal is calculated according to equation 1, where v1, v2 and v3 are the vertices spanning up the face, and then it is then normalized.

$$n = (v2 - v1) \times (v3 - v1) \tag{1}$$

5. Associate all half edge pair to the face because all half edge pair shares the same left face, as seen in Figure 1.

2.2 Implement neighbor access (*)

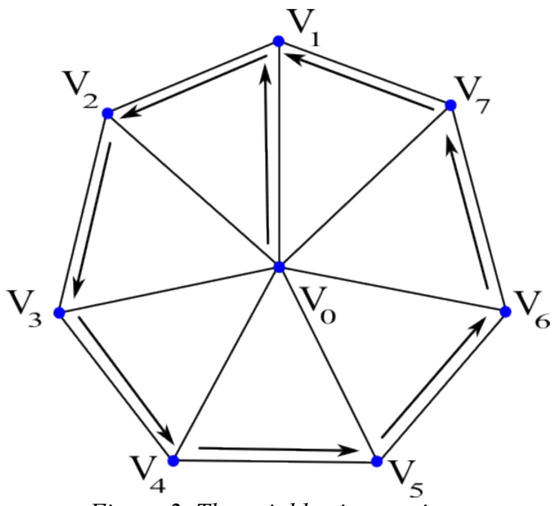

Figure 2. The neighboring vertices.

To be able to calculate the neighboring faces the neighboring vertices needs to be found first. This is done in the function FindNeighborVertices which takes in a vertex index and returns all the neighboring vertices which is stored in a vector. This is done from the current vertex (V_0), step ahead with the half-edge next pointer to the ring surrounding the current vertex, see Figure 2. Once out in the ring the only thing that needs to be done is to store the vertex index and step ahead with the half-edge next pointer until it points to the starting vertex in the ring.

The neighboring faces is done by using this ring of surrounding vertices and check their corresponding faces. Because every half-edge stores the left face it is easily retrieved by looping through the neighboring vertices and putting all neigboring faces' indices into a vector.

2.3 Calculate vertex normal (*)

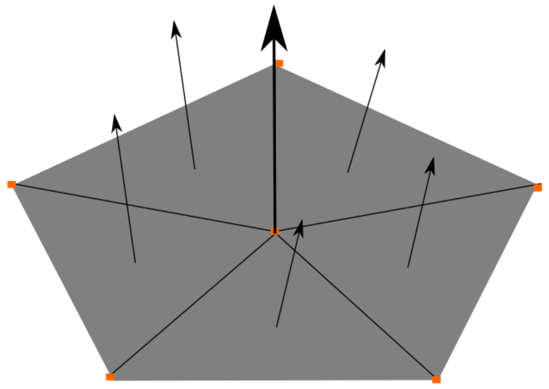

Figure 3. Vertex normal and its surrounding face normals.

The vertex(n_v) normal is calculated according to equation 2, which is the normalized sum of the neighboring face normals($n_{f,}$), see Figure 3, where the face normal is calculated as mentioned earlier with equation (1).

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

2.4 Calculate surface area of a mesh (*)

The area of the mesh is the integral over all area segments on the mesh, which can be approximated into a Riemann sum of the area of each face/segment area, see equation 3a.

$$A_S = \int_S dA \approx \sum_{i \in S} A(f_i) \tag{3a}$$

$$\frac{1}{2}|(v2 - v1) \times (v3 - v1)| \tag{3b}$$

The area of the i:th face, $\mathbf{A}(f_i)$ is calculated as half the magnitude of the cross product between the edges in the i:th face, see equation 3b.

2.5 Calculate volume of a mesh (*)

The volume of the mesh is calculated by using the divergence theorem, also know as the Gauss theorem[2] which gives a connection between the surface- and volume- integral. By integrating over the surface, where **F** is a vector field and **n** is the unit normal the surface integral can be written as a volume integral of the divergence of the same vector field **F**, see equation 4.

$$\int_S \mathbf{F} \cdot \mathbf{n}\, dA = \int_V \nabla \cdot \mathbf{F}\, d\tau \qquad (4)$$

Because that the vector field **F** can be set to any vector field, the field can be assumed to have a constant divergence gives equation 5. The result is the volume times a constant **c**.

$$\int_V \nabla \cdot \mathbf{F}\, d\tau = \int_V c\, d\tau = c \int_V d\tau = cV \qquad (5)$$

The vector field $\overline{F} = (x, y, z)$ which gives the constant **c**, a value of *3* [1]. Equation 4 and 5 can be written into a single equation, see equation 6.

$$\int_V \nabla \cdot \bar{\mathbf{F}}\, d\tau = \int_S \bar{\mathbf{F}} \cdot \mathbf{n}\, dA \;=\; 3V \qquad (6)$$

The surface integral can be computed by approximating it with a Riemann sum, see equation 7, where **n**(f_i) is the face normal and **A**(f_i) is the area for i:th face.

$$\int_S \bar{\mathbf{F}} \cdot \mathbf{n}\, dA \;\approx\; \sum_{i \in S} \bar{\mathbf{F}}(f_i) \cdot \mathbf{n}(f_i) A(f_i) \qquad (7)$$

The vector field $\overline{F} = (x, y, z)$ is simply chosen as the centroid of the face where **v1**, **v2** and **v3** are the vertices spanning up the face, see equation 8. Although the vector field varies over the face in contrast to the normal and area which are constant, the vector field can still be approximated at any chosen point on the face.

$$3V = \sum_{i \in S} \frac{(\mathbf{v}_1 + \mathbf{v}_2 + \mathbf{v}_3)_{f_i}}{3} \cdot \mathbf{n}(f_i) A(f_i) \qquad (8)$$

Equation 8 was implemented in the Volume function in the HalfEdgeMesh.cpp to calculate the volume of an object. By looping through all faces and summing all the faces according to equation 8, the volume was calculated.

2.6 Implement and visualize curvature(*)

There are several ways of describing the smoothness of a surface described by curvatures, and the most frequent methods used are the Gaussian curvature and mean curvature[1], where the former is composed by equation 9. The area **A** is calculated in the same way as before, see equation 3b.

$$K = \frac{1}{A}\left(2\pi - \sum_{j\in N_1(i)} \theta_j\right) \tag{9}$$

The sum of the angles Θ_j is calculated according to Figure 4. If the vertex $\mathbf{v_i}$ lies within the plane or is a saddle point the curvature estimate K is zero, because the sum of Θ_j adds up to 2π. The only way for a Gaussian curvature to be negative is if the total the sum of Θ_j adds up to more than 2π, which can only happen if some of the neighboring vertices are over the vertex $\mathbf{v_i}$, like a saddle-shaped mesh for example.

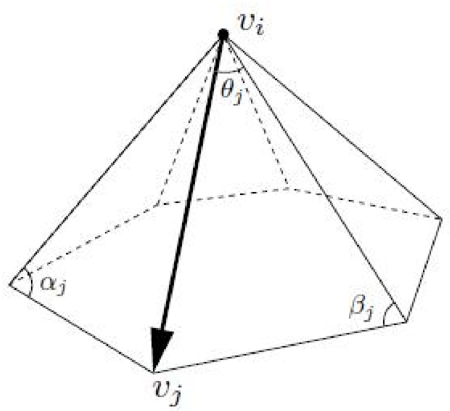

Figure 4. Representation of the angles α_j, β_j, Θ_j for the curvature estimate. Image Courtesy of [1].

2.6.1 Improve the curvature estimate (**)

To better represent the curvature, the area **A** in equation 9, is now instead calculated with the voronoi area, see equation 10, where α_j , β_j is the angles seen in figure 4.

$$A_v = \frac{1}{8}\sum_{j\in N_1(i)} (\cot\alpha_j + \cot\beta_j)|(\mathbf{v}_i - \mathbf{v}_j)|^2 \tag{10}$$

Further improvements can be made, for example calculate the mean curvature which will be discussed in section 2.8.

2.7 Classify the genus of a mesh (**)

The genus of a mesh is calculated using the Euler-Poincaré formula, see equation 11, where V is the number of vertices, E is the number of edges, F is the number of faces, L equals the number of loops, S is the number of shells and G, is the genus.

$$V - E + F - (L - F) - 2(S - G) = 0 \tag{11}$$

For convenience the number of shells is set to 1 and the number of loops equals the number of faces, because there only exists one loop within a face, so equation 11 can be reduced to equation 12, and finally the genus can be calculated as equation 13 which was implemented in the Genus function.

$$V - E + F - 2(1 - G) = 0 \tag{12}$$

$$G = \frac{E - V - F + 2}{2} \tag{13}$$

2.8 Calculate the mean curvature and do smoothing (***)

To be able to differentiate between concave and convex curvature, one has to implement the mean curvature, because the Gaussian curvature does not take this into account. The mean curvature equation is seen in equation 14.

$$H\mathbf{n} = \frac{1}{4A} \sum_{j \in N_1(i)} (\cot \alpha_j + \cot \beta_j)(\mathbf{v}_i - \mathbf{v}_j) \tag{14}$$

Equation 14 returns a vector **Hn** which is compared to the vertex normal through scalar product, see equation 15. The area **A** is the voronoi area and the see figure 4 for the summation of the vectors.

$$H = \begin{cases} |Hn|, \; if\ Hn \cdot n_{v_i} > 0 & \text{(convex)} \\ -|Hn|, \; else & \text{(concave)} \end{cases} \tag{15}$$

For clarification about the summation and sign of H, see Figure 5.

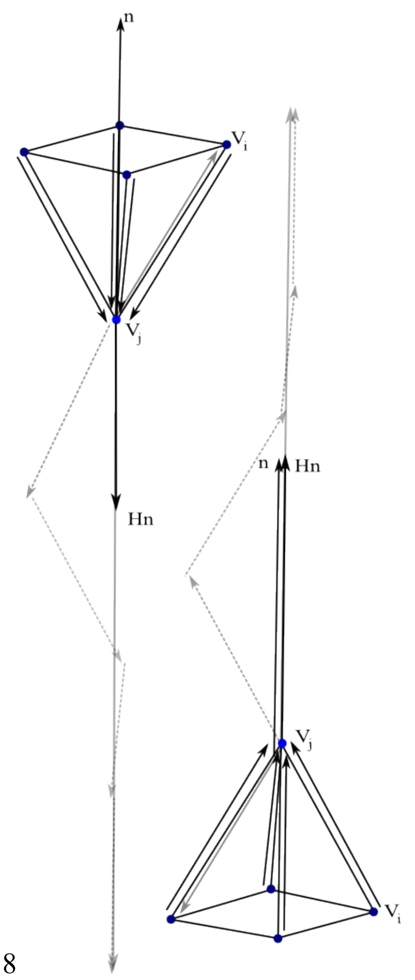

Figure 5. Concave and convex calculation, to the left concave, right convex.

The mean curvature estimate is then used to move the vertices in an out- or in-wards direction along the vertex normal according to the sign and magnitude of the curvature estimate. This is done with explicit Euler intergration [3], which is just to take the previous position and add a small step in the direction of the vertex normal according to the curvature estimate, see equation 16.

$$v_{i,new} = v_i - step * curvature_i * normal_i \tag{16}$$

2.9 Present a lower memory bound for the half-edge data structure (***)

The half-edge data structure has some information that can be reduced in order to save memory.
In listing 1. the structure of the entire half-edge mesh data structure is seen.

Listing 1: Half-edge data structure.

```
struct Face ;
struct Vertex ;

struct Halfedge{/ / topology
	Vertex* vert ;
	Halfedge* next ;
	Halfedge* prev ;
	Halfedge* pair ;
	Face* left ;
};
struct Vertex{/ / geometry
	float x , y , z ;
	Halfedge* edge ;
}
struct Face{
	HalfEdge* edge;
};
struct Mesh{
	Vertex verts [ V ] ;
	Face faces [ F ] ;
	Halfedge edges [3 F ] ;
} ;
```

Assuming that a float takes 4 bytes and $F \approx 2V$, the entire half-edge data structure gives us according to equation 17, 72F.

$$\begin{aligned} & V * \texttt{sizeof}(\texttt{Vertex}) + 3F * \texttt{sizeof}(\texttt{Halfedge}) \\ & \qquad + F * \texttt{sizeof}(\texttt{Face}) \\ & \approx \frac{F}{2} * 16 + 3F * 20 + F * 4 = 72F \end{aligned} \tag{17}$$

Some reduction can be made in the topology. The previous pointer is the same as to take next->next so the topology could be reduced by 4 bytes by removing the prev pointer. The face can also be constructed by the three pointers by getting each vertex, so there is no need to store a face, however no attributes like color can be set to a face when doing this. This makes the Faces class redundant as well, which gives a reduction of 8 bytes, 4 for the face pointer and 4 bytes for the face class which contained an edge.

The size of the vertices is still 16 bytes and the size of the half edge structure is now 12 bytes, which gives a total of 44F. This reduces the memory to ~61 percent of the total memory. However constant time neighboring access can not be acquired because the face pointer is removed and the face needs to be calculated instead. To be able to have constant neighboring access the face pointer and face structure can not be removed, which makes the size of the half edge structure to 16 bytes and also the Face class is no longer redundant which is 8 bytes and therefore gives a total of 60F. The lower memory bound and with constant time neighboring access the data structure can be compressed to ~83 percent of the total memory.

3 Results

3.1 Implementation of the half edge mesh

The half edge mesh data structure loads in an object a lot faster than the normal vertex- and face- data structure. This was not so noticeable on simple meshes but for a large mesh for example the bunny it was noticeable, see table 1, for a comparison between the simple data structure and the half edge data structure. The time is only approximate because the time was manually timed and should therefor only be looked at as an example of that the half edge mesh is somewhat faster than the simple data structure, when it comes to large meshes.

Table 1: An approximation of how much faster the half edge mesh is for large meshes.

	Simple mesh data structure	Half edge mesh data structure
Stanford Bunny	4.4s*	2.3s*

*Note that it is manually timed.

3.2 Calculate the vertex normal

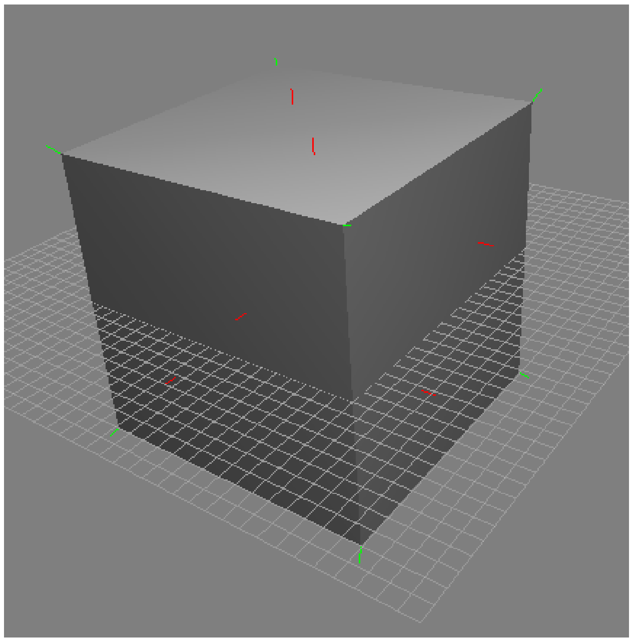
Figure 5. Vertex normals, shown in green and face normals shown in red.

The vertex normal is calculated by equation (2) where the neighboring face normals are summed together and normalized. Figure 5, shows the calculation of the vertex normals, and if one looks carefully at the bottom vertex normal, one sees that the direction of the normal is not totally correct. This is because that if there are two or several neighboring faces on one side of the cube and on an other side, the right side in the image for example, has a smaller amount of neighboring faces the direction of the normal will be slightly tilted in the direction of the side with more neighboring faces. Several different thing can be taken into account to get better performance, for example weighting according to the edge lengths, area of the face, angles to the neighboring faces. However, this method worked just fine for this purpose.

3.3 Calculate surface area and volume of a mesh

The area and volume were calculated according to equation 3a, respectively 8.
By comparing the area and volume of a unit sphere mesh with the calculated area and volume, see equation 18, one sees that computed area and volume of the mesh resembles closely the calculated area and volume, but it is not exact, see table 2.

$$A = 4\pi r^2, V = \frac{4}{3}\pi r^3 \qquad (18)$$

Table 2: Area and volume calculations for the mesh and calculated from equation 18.

	Mesh	Calculation
Area of a unit sphere	12.5048	$4\pi \approx 12.566$
Volume of a unit sphere	4.14881	$\frac{4}{3}\pi \approx 4.18867$

The area and volume of the mesh will converge to the correct value as the number of vertices approximating the sphere goes to infinity.

3.4 Implementation and visualization of different curvatures

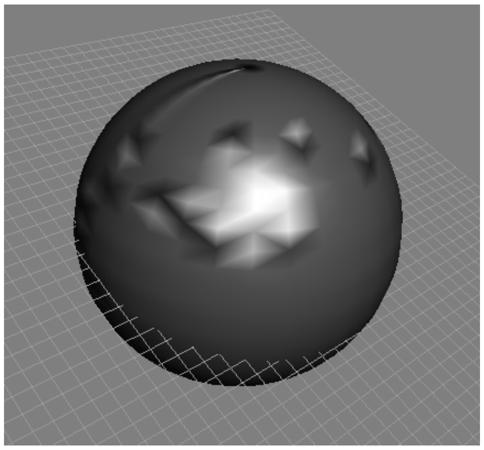
Figure 6. Gaussian curvature

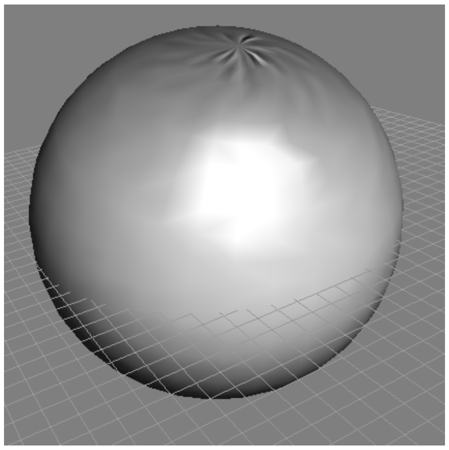
Figure 7. Gaussian curvature but, divided by the voronoi area.

As seen in figure 6 and 7 the Gaussian curvature estimate is rather poor, look especially at the bumps in figure 6, but by dividing with the voronoi area instead better smoothing is achieved,see figure 7 and 8. There are though still some problems around the poles of the sphere, but it is a lot better.

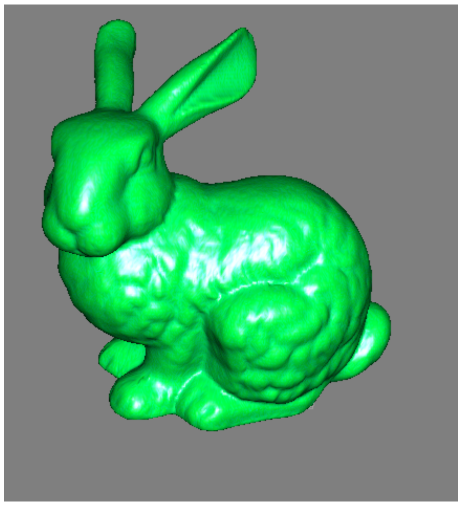
Figure 8. Voronoi Area, Gaussian curvature.

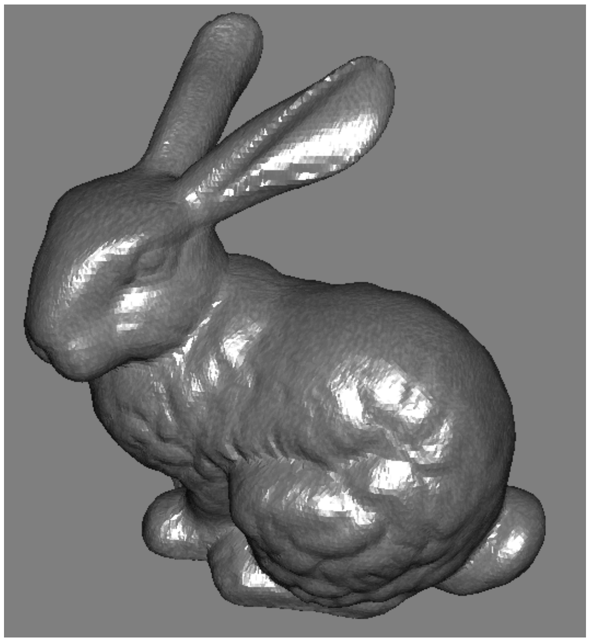
Figure 9.Original bunny before smoothing.

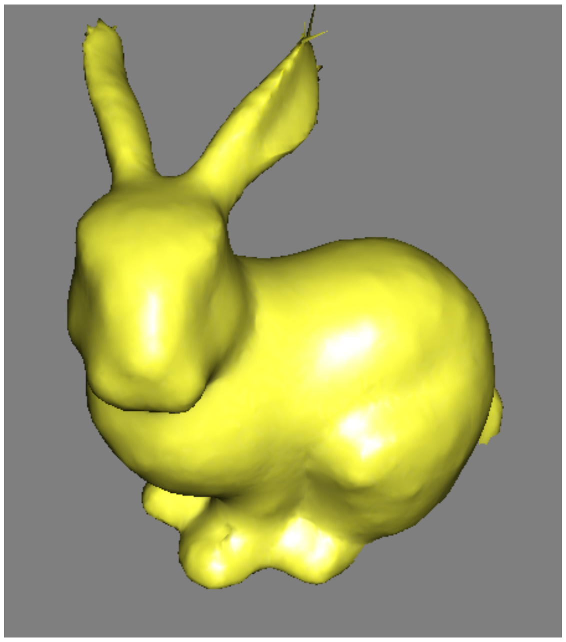
Figure 10. Smoothed bunny after many iterations.

The mean curvature differentiates between convex and concave curvature, see section 2.8, and with this smoothing of the actual mesh can be done. The **step** in equation 16 was chosen to be a value smaller than 0.01 for stability. However, the step needed to be very small for the cow mesh, since the cows' mesh is very small compared to all other meshes in the program, and if the vertices area moved when iterated with a quite large step compared the the vertices's positions the new mesh will be very different from the original mesh. See image 9, and 10 for mean curvature smoothing.

If the mesh is iterated too many times the sharp edges will break, probably because the iteration step is too large so the vertex will cross the face and be treated as a concave point instead of a convex point which it should have been, or vice versa. This error is seen for example around the ears on the bunny, see image 11.

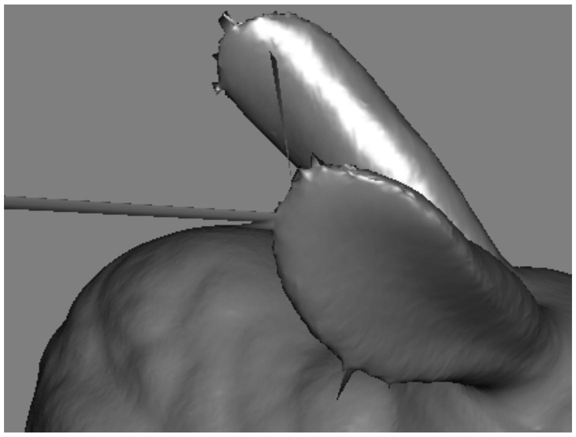
Figure 11. Sharp edge crash

3.5 Present a lower memory bound for the half-edge data structure

As seen in section 2.9 the memory bound for the half-edge data structure can be reduced to a memory of 44F bytes instead of 72F bytes. However, when reducing this memory bound the calculation time to get the triangles and vertices will increase, so one has to choose either a larger memory bound with greater speed or a lower memory bound but with loss of computational speed. If compared to a simple vertex- and face list that uses the indices in the face list, the memory is calculated to be 60F compared to the half-edge mesh that uses 72F bytes. However the half edge data structure is is a lot faster when it comes to large meshes. The half edge data structure can be reduced to 60F,which is the same amount as the simple data structure, and still have constant time neighboring access.

4 Conclusion

This lab gave a good understanding of the useful half edge mesh data structure and how fast and useful it is for geometric calculations such as, the mean curvature estimate, calculation of the surface area and volume, genus classification and vertex normal calculation. It was a very good theoretical and practical assignment which gave a good and useful understanding of very important parts of mesh data structures and curvature estimates.

Lab partner and grade

The lab was carried out by me, Dan Englesson, and my lab partner Emil Brissman. We implemented all mandatory assignments as well as all assignments needed with grade 4, one needed for grade 4, and two assignments with grading 5 where only one was needed to be done in order to be able to write for grade 5. As stated in the previous sentence me and my lab partner have full-filled the requirements for grade 5.

References

[1] David Porush. A Short Guide to Writing about Science. Longman Publishing Group, 1997.
[2] http://en.wikipedia.org/wiki/Divergence_theorem, 2011-03-10
[3] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr.
Implicit fairing of irregular meshes using diffusion and curvature flow.
In SIGGRAPH'99: Proceedings of the 26th annual conference on Computer
graphics and interactive techniques, pages 317–324,
New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.