Modeling and Animation

Mesh decimation

Gustav Gahm
gusga293@student.liu.se

April 29, 2010

Abstract

In this short paper I present theory and results for the second lab session in the course TNM079 - Modeling and Animation. The lab session was about *Mesh decimation* and in particular *mesh decimation using error quadrics*. I give an brief overview what decimation is and why it is used. Then i describe how mesh decimation using error quadrics was implemented and show the results.

1 Introduction

During the course TNM079 - Modeling and Animation six lab sessions are to be performed by the participants. Each of these sessions are to be presented in a short report describing what has been done and how. This report covers my work in the second lab session about mesh decimation.

1.1 Mesh Decimation

Mesh decimation is the process of removing vertices and faces from a mesh in order to make it more light weight to store in memory and render.

Usually in computer graphics one want rendered pictures to be as good looking as possible. This often means having very detailed scenes, and one way of getting this detail is using high resolution models. Unfortunately high resolution models will use a lot of memory and take time to render, making them unusable for e.g. games. It is simply not possible to render a few hundred million faces in real-time. However, it might be possible to render one object of 100k triangles and the rest of the scene with another 100k.

By using mesh decimation one can automatically generate several versions of a model, e.g. one to use when the camera is near an object, one when it is a little bit further away and one when it is almost impossible to see the object. Using different versions of an object like this is known as level-of-detail [1].

The main idea of mesh decimation is to model an object with a desired level of detail and automatically reduce the number of triangles without losing the overall structure of the object.

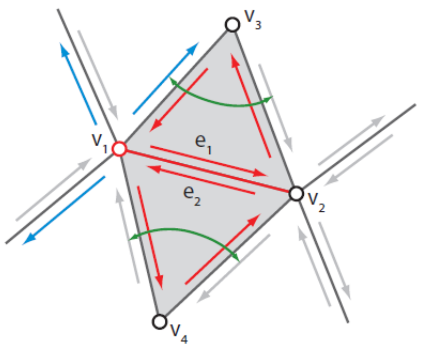


Figure 1: Edge collapse in quadric mesh decimation. Image courtesy of [4]

1.2 Simple mesh decimation

One way of doing mesh decimation is to enclose an object in a three-dimensional grid and merge all vertices that share a cell in the grid. This will result in a mesh of lower resolution, but it is impossible to tell the algorithm to do it in a way that preserves the overall appearance of the object.

1.3 Quadric based mesh decimation

In 1997 Garland et. al. presented a paper called *Surface Simplification Using Quadric Error Metrics* [3] in which they describe a way of decimating a mesh while having control over the overall appearance of the object.

The base operation of quadric based mesh decimation is the edge collapse. An edge collapse is an operation that reduce an edge into a single vertex, i.e. two vertices are merged into one. When this is done all edges and faces connected to the removed vertices has to be reconnected to the new vertex, Fig. 1. See [3] for an in depth explanation.

2 Method

2.1 Implementation of mesh decimation using quadrics

In the code base given for the lab session most of the edge collapse step in quadric based mesh decimation is already implemented. This reduces the implementation to two steps.

1. Calculating the matrix $\mathbf{Q}$ for all the initial vertices of the mesh.
2. Computing the new vertex $\bar{\mathbf{v}}$ for all collapsed edges.

As said above, collapsing an edge is the result of merging two vertices ($\mathbf{v_1}$ and $\mathbf{v_2}$) into one ($\bar{\mathbf{v}}$). The problem is two know where to position the new vertex. It would be possible to position the new vertex e.g. right between the old ones, but it could result in a change of the overall appearance of the object. The trick is to find the new position while minimizing the change in overall appearance. To do this one need a way of defining the change, or error.

When working with quadric based mesh decimation the error is defined as Eq. 1.

$$\Delta(\mathbf{v}) = \mathbf{v}^T \mathbf{Q} \mathbf{v} \tag{1}$$

2.1.1 Calculating the matrix Q

Before any decimation can been performed on a mesh one has to calculate an initial value for $\mathbf{Q}$ for every face in the mesh. One start by finding all neighbouring faces to a vertex $\mathbf{v}_i$. For each face $f_{i,j}$ it is possible to calculate the so called *fundamental error quadric* $\mathbf{K_p}$ [3] which can be used to calculate the squared distance between the plane of the face and any point in space.

$$\mathbf{K_p} = \begin{bmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{bmatrix} \tag{2}$$

The variables a, b, c and d of $\mathbf{K_p}$ are the corresponding variables in the plane equation $ax + by + cz + d = 0$. For a face $f_{i,j}$ a, b and c are given by its normal $\hat{\mathbf{n}} = (a, b, c)^T$. The variable d is found by inserting a point $\mathbf{p} = (x, y, z)^T$ into the plane equation and solving for d as $d = -(ax + by + cz)$. Because all vertices of $f_{i,j}$ are points in the plane the point $\mathbf{p}$ is chosen as $\mathbf{v}_i$.

Once one have found $\mathbf{K_p}$ for all the neighbouring faces of $\mathbf{v}_i$ it is just a matter of adding them together to find $\mathbf{Q}$, see Eq 3.

$$\mathbf{Q} = \sum_{\mathbf{p} \in planes(\mathbf{v}_i)} \mathbf{K_p} \tag{3}$$

An important thing to notice is that the errors for all initial vertices are equal to zero. This is because the distance between a vertex and its connecting faces is always zero due to the fact that the vertex is located on the planes.

2.1.2 Calculating the new vertex $\bar{\mathbf{v}}$

By knowing that the error is equal to Eq. 1 finding $\bar{\mathbf{v}}$ becomes a task of minimizing $\Delta(\bar{\mathbf{v}})$, which is a linear problem that is solved by finding the vector $\bar{\mathbf{v}}$ that satisfies $\partial\Delta/\partial x = \partial\Delta/\partial y = \partial\Delta/\partial z = 0$, which can be written as Eq. 4

$$\bar{\mathbf{v}} = \begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{12} & q_{22} & q_{23} & q_{24} \\ q_{13} & q_{23} & q_{33} & q_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{4}$$

An important thing to notice in Eq. 4 is that the 4x4 matrix is not equal to $\mathbf{Q}$, it is only the first three rows of $\bar{\mathbf{Q}}$ with an additional row of [0 0 0 1]. $\bar{\mathbf{Q}}$ is defined as $\bar{\mathbf{Q}} = \bar{\mathbf{Q}}_\mathbf{1} + \bar{\mathbf{Q}}_\mathbf{2}$, in which $\bar{\mathbf{Q}}_\mathbf{1}$ and $\bar{\mathbf{Q}}_\mathbf{2}$ are the quadric matrices to the vertices of the edge to collapse. This is explained in detail in [3].

2.2 Cleaning up a decimated half-edge mesh

One drawback with the base code for the lab session is that it does not make more memory available when a mesh is decimated. This is because removed vertices are only flagged as removed, not actually deleted.

A quick fix for this is copying all vertices, faces and edges that are not removed into a new mesh, replace the old mesh with the new one and finally delete the old mesh.

2.3 Visualizing the quadric error metric

When an edge is collapsed into a new vertex $\bar{\mathbf{v}}$ the quadric error at this position is equal to value ϵ. In [2] it has been shown that it is possible to visualize the error ϵ as an iso-surface at the position of $\bar{\mathbf{v}}$. This is done using Cholesky decomposition [2] which will result in a matrix $\mathbf{R}$ which is the upper triangular matrix of $\mathbf{Q}$. By rendering a unit sphere and transform it with the inverse of R, one will see the iso-surface of the error at the position of $\bar{\mathbf{v}}$.

3 Results

3.1 Implementation of mesh decimation using quadrics

Implementing mesh decimation using quadrics was quite straightforward as most of the edge collapse algorithm already was implemented. An example of using the algorithm can be seen in Fig. 2.

In Fig. 3 the difference of using the quadric based decimation versus a naïve implementation is shown. It is clearly shown that the model decimated using quadrics has preserved more of its overall appearance. Both models were reduced to 1000 triangles.

3.2 Cleaning up a decimated half-edge mesh

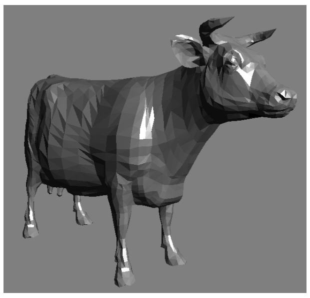
Implementing the cleanup function for a decimated mesh turned out to quite hard. Due to a bug in the base code the outcome of running the cleanup function was not allways more available memory. When tested we could not see any difference in memory use.

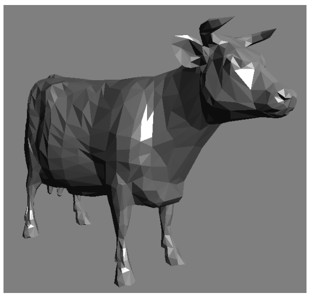
3.3 Visualizing the quadric error metric

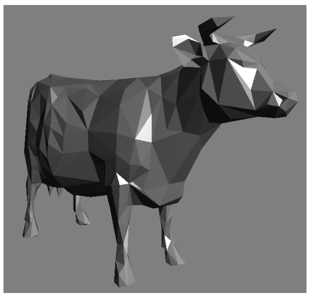
Using the technique descibed above to visualize the error resulted in Fig.4.

4 Conclusions

I have learned that it is possible to automatically decimate a mesh without losing the overall appearance of the model. It is also possible to do relatively aggresive changes to the mesh e.g. 5000 to 1000 triangles in the case of the cow model. When using a half-edge mesh the implementation of decimation becomes quite easy.


(a) Original


(b) 2500 triangles


(c) 1000 triangles

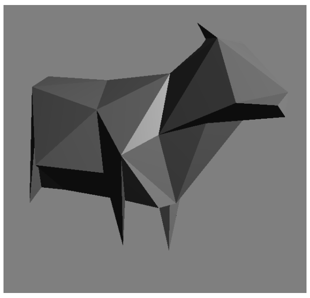
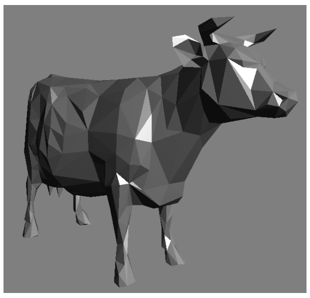

(d) 100 triangles

Figure 2: Cow model decimated to different resolutions.


(a) Quadric mesh decimation

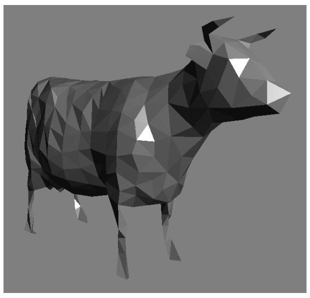

(b) Naïve decimation

Figure 3: Quadric mesh decimation vs. naïve decimation ($\bar{\mathbf{v}} = \mathbf{0.5}(\mathbf{v_1} + \mathbf{v_2})$).

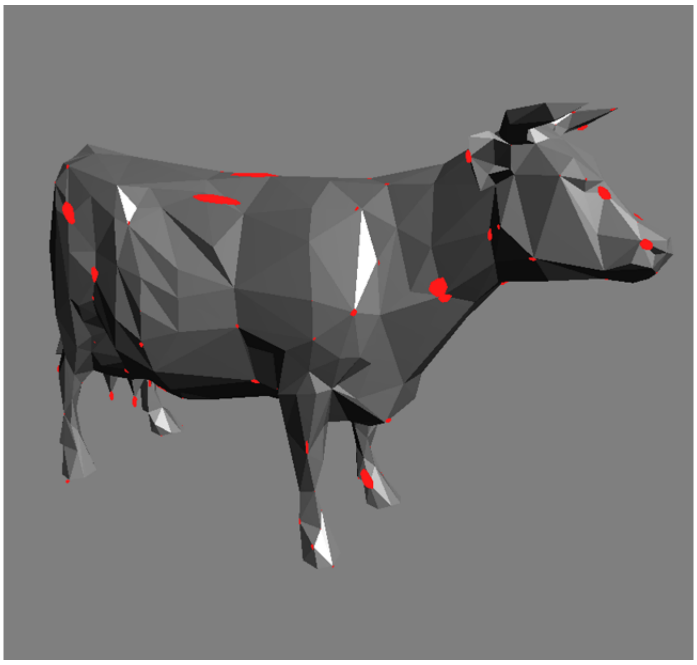


Figure 4: Visualization of the error quadric matric.

References

[1] Tomas Akenine-Möller, Eric Haines, and Natty Hoffman. *Real-Time Rendering 3rd Edition*. A. K. Peters, Ltd., Natick, MA, USA, 2008.

[2] Michael Garland. *Quadric-based polygonal surface simplification*. PhD thesis, Pittsburgh, PA, USA, 1999. Chair-Heckbert, Paul.

[3] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 209–216, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[4] Gunnar Läthén, Ola Nilsson, and Andreas Söderström. Mesh decimation, 2010.