Efficient polygon reduction in Maya

Marcus Flaaten

2015-05-29
Efficient polygon reduction in Maya

Examensarbete utfört i Medieteknik
vid Tekniska högskolan vid
Linköpings universitet

Marcus Flaaten

Handledare Andrew Gardner
Examinator Jonas Unger

Norrköping 2015-05-29
Upphovsrätt

Detta dokument hålls tillgängligt på Internet – eller dess framtidare ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

Copyright

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© Marcus Flaaten
Efficient polygon reduction in Maya

Marcus Flaaten
June 26, 2015

Abstract
Reducing the number of vertices in a mesh is a problem that if solved correctly can save the user a lot of time in the entire process of handling the model. Most of the solutions today are focused on reducing the mesh in one big step by running a separate application. The goal of this implementation is to bring the reduction application into the users workspace as a plugin. Many of the modellers in the various computer graphics industries use Autodesk Maya the plugins intention is to create a efficient tool which also give the modellers as much freedom as possible without the need to ever leave Mayas workspace. During the process the possible issues and solutions of creating this tool in Maya will also examined to help introduce the process of creating a tool for Maya. This plugin has the potential to improve on the existing reduction tool in Maya by giving the user more options and a more exact solution.

1 Introduction
Mesh reduction is one of the more complex problems in computer graphics. There are several different solutions which varies a lot in the result depending on what the user is looking to achieve. It is still one of the problems which needs a trade off between accuracy and execution time. It is possible to do an exact error calculation of how much error a removal of a vertex introduces to the mesh, but it is computationally heavy and not a viable option for a tool that needs to work relatively fast.

Creating a Maya plugin that will reduce the amount of vertices and thus the amount of polygons in the mesh can be done in several ways. Depending on what the use the tool will fulfil it can be very different, there are two extremes, either fast execution or accurate result. Creating an efficient
tool means finding a good middle ground between the two extremes. In this implementation the plugin aims to be as accurate as possible within a reasonable time, but emphasising the accuracy at the expense of the execution time. This involves several time consuming steps, converting the Maya mesh to and from another mesh structure more suitable for reduction, estimating the introduced error of a vertex removal, and the removal of a vertex.

By implementing a vertex reduction tool into Maya a lot of built in functionality is ascertained for free. Maya supplies many tools for selection and interactivity with a mesh and allows for a lot of data to be given as inputs to the plugin. With the data that Maya supplies a versatile tool can be made which allows the user to reduce the mesh in a customised fashion.

The goal of this project is to learn how to create a plugin for Maya that works similarly to the already existing reduction tool in Maya. This tool would ideally work as fast as the Mayas own tool while offer the user more options on how the mesh will be reduced.

The approach to this problem is to create a reduction tool and a plugin application and then merging the two, making Maya wrap around the reduction algorithm and communicate with Maya in an efficient way.

\section{Related work}

There are many different implementations and approaches to the decimation problem. In this section brief explanations of established methods in the major steps of the decimation. In section 2.1 the various ways to represent a mesh is discussed, also open source alternatives are presented. After the mesh structure has been established the various polygon reduction methods is discussed in section 2.2. Lastly the different criteria to decide which polygon to reduce will be examined.

\subsection{Mesh data structure}

Before any of polygon reduction methods can be discussed a decision about how the structure of the models will be built needs to be made. When building a mesh data structure one needs to take into account what it will be used for. The simplest way to represent a model is just to store all the vertices and their corresponding faces in a array without any form of navigation.
This will make it nearly impossible to perform any advanced operations and make assumptions about the topology of the model. This is why there are some mesh data structures where you can navigate from vertex to vertex and build up a neighbourhood which can be factor into the evaluation of a vertex. There are a couple of mesh data structures that incorporates some sort of navigation. All of the navigating mesh data structures is fairly similar, half edge mesh is probably the most simple and widely used. It offers a good base for navigating and building up the model.

Half edge mesh [1] is a mesh structure that not only stores the vertices and the faces of the mesh but also the edges that connect the vertices. Each edge is divided in two, one for each face it is connected to, if the edge is a boundary the other edge is stored as NULL. The navigation is constructed as shown in figure 1.

![Half Edge Mesh Diagram](image)

Figure 1: A picture showing the half edge mesh data structure where the whole arrows are directly accessible from the edge represented by a bold arrow.

This mesh data structure have all the necessary tools and possibilities needed for this project.

Since the mesh data structure is not the focus of this project an open source alternative might be useful to save time and get a very effective solution. There are two main options for open source mesh data structures OpenMesh and CGAL. CGAL is a more general solution that does several types of structures and operations all reasonably well, while OpenMesh is specialised on the half edge mesh structure.
2.2 Polygon reduction

A polygon reduction or mesh decimation can be divided into two steps. First an error estimation, this is the method used to decide which vertex to remove. The error estimation is a estimation of how big the error will be between the original mesh and the new one after the mesh has been decimated. There are several ways to do an error estimation, most famous is Garlands quadric error estimation [2]. Which is a good general method that works both on manifold and non-manifold meshes. It is possible to do an trial and error solution as well to get the true error, but this is not efficient since it takes a lot of computational power. The second step is the actual removal of the chosen vertex. There are some different ways to remove vertices from the mesh, in general it is applying some kind of weight value to vertices and then removing the vertex with the least weight.

2.2.1 Vertex removal

The problem of removing and adding vertices to a mesh for improving efficiency or detail is a well studied problem. There are a couple of fundamental methods that have been improved or reworked to fit more specific solutions.

**Vertex decimation:** The general idea is to remove a vertex and all it’s connecting faces and then re-triangulate the hole [3]. By iteratively doing this across the mesh you get a simplified version of the original structure. It also keeps the mesh structures shape very well without deforming it, which can be a benefit or a drawback depending on the goal of the application. Another drawback of vertex decimation is that it is limited to manifold meshes.

![Figure 2: In this figure a vertex is removed and the hole is re-triangulated.](image)

**Vertex clustering:** As the name suggests it clusters vertices together by creating a bounding box over the mesh and dividing it in a grid pattern [4]. All the vertices in a grid are compressed to a single vertex and then the faces are remade between the new vertices. This is a fast method but it is hard to
control the resulting mesh, since it can make big changes to the topology of the mesh and the number of resulting faces after the operation is uncontrollable.

**Edge contraction/collapse**: Another way of explaining an edge contracting is to say that the distance between two vertices is shrinking to zero. What this method does is to combine two vertices into a single vertex [5]. Calculating the new position can either be done by combining their two positions into a new point or just chose one of the two vertices that is to be removed. When the vertices has been combined it inherits all the edges of the two previous vertices. Edges that originated from the original two vertices to the same vertex will become duplicates, one of these are therefore removed. This is a variation of the vertex decimation approach, and has much the same benefits and drawbacks.

**Pair contraction**: This is an extension of the edge contraction rather than needing a edge between two points this can just identify two vertices and combine them into a single vertex [2]. This can be useful in a non-manifold mesh. The pair contraction method also supports the aggregation of vertices and allows for a good control over the number of faces and topology changes. It is a good middle ground for the above mentioned methods.

Depending on what is important for the application any of these methods can be a good choice.

### 2.2.2 Error estimation

To make a efficient polygon reduction a good approximation of what the mesh will look like after a vertex has been removed is necessary. This is usually done by estimating an error before a vertex is removed between the original and the resulting reduced mesh. The lower the error is for a vertex the more likely it is that it is not an important vertex for the mesh and can therefore be removed without much topological change. It is possible to make a very exact error estimation or even calculate the true error of a vertex removal, this is however not cost efficient and a trade of between time and accuracy is needed. What method is chosen or implemented is dependent on what the goal of the project is and might have to be revised along the way depending on efficiency. Below a couple of fundamental methods will be presented and a short conclusion of them will be made after.
**Average plane:** The first method is based on creating a plane out of the one-ring-neighbourhood around a vertex. The basic equation of the method is,

\[ \vec{N} = \frac{\sum_{i=1}^{m} \vec{n}_i A_i}{\sum_{i=1}^{m} A_i} \]  \hspace{1cm} (1)

\[ \vec{n} = \frac{\vec{N}}{|\vec{N}|} \]  \hspace{1cm} (2)

\[ \vec{x} = \frac{\sum_{i=1}^{m} x_i A_i}{\sum_{i=1}^{m} A_i} \]  \hspace{1cm} (3)

where \( \vec{n}_i \) is the normal of the faces in the one-ring-neighbourhood of the vertex, \( A_i \) the area of the same faces, and \( x_i \) the midpoints of the faces [3]. The distance from the vertex to the average plane can then be calculated as,

\[ d = |\vec{n}(\vec{v} - \vec{x})| \]  \hspace{1cm} (4)

the smaller this \( d \) the less error will probably be introduced to the mesh if it is removed. A simple and easy approach to error estimation, works well with simple meshes, but suffers with accuracy with more complex meshes.

![Figure 3: A figure showing the imaginary plane that is created by the neighbourhood.](image)

**Curvature:** By evaluating the Gaussian or the mean curvature [6] an estimation of the topology of the area can be obtained. The two methods produce different results with the same input, depending on what feature is deemed important a choice between which method to use needs to be made. The Gaussian curvature focus more on the close proximity of the interesting point while the mean draws a broader conclusion depending on the surrounding area. The Gaussian curvature is calculate as,

\[ K = \frac{2\pi - \sum_{i=1}^{m} \alpha_i}{\frac{1}{3} \sum_{i=1}^{m} A_i} \]  \hspace{1cm} (5)

where \( \alpha \) is the vertex angle for all the neighbouring vertices. The mean curvature is calculated by taking the average of the principal curvature, the general equation looks as follows,

\[
H = \frac{1}{n} \sum_{i=1}^{n} \kappa_i
\]  

(6)

The curvature can be a good complement to an error estimation but it is too inaccurate for a stand-alone estimation.

Volume estimation: The goal of the volume estimation method is to keep the volume of the mesh as close to the original as possible. This is done by introducing a virtual vertex that creates a tetrahedron with the three vertices of a face [7]. The volume of this tetrahedron is then the cost of removing the vertex, the lower the cost the less the volume of the mesh will change if the vertex is removed. This is a good way to prevent extreme shrinking of a mesh as more and more vertices are removed. This is a rather heavy operation if done over the whole mesh, but can be a good complementary tool in a refinement process.

Quadric error estimation: The pair contraction is tightly connected to the quadric error estimation written by Garland [2]. To calculate the error at a vertex it uses a 4x4 symmetrical matrix that is multiplied with the vertex \( \vec{v} = [v_x, v_y, v_z, 1] \) where \( v_x, v_y, \) and \( v_z \) is the vertex coordinates. The equation is simply,

\[
\delta(v) = v^T Q v
\] 

(7)

where \( \delta(v) \) is the error and \( Q \) is the quadric. The smaller the error is the more important the vertex is to the model.

These methods are all capable of decimating a mesh by themselves but by combining them in different situations or with some restrictions a better result can be achieved. Either by speeding up the process or by decreasing the error of the resulting mesh. As with most computer graphics fields there is a trade-off between speed and accuracy, the key is to find a good middle ground that is satisfactory to the end user.

3 Method

There are three main steps to make a reduction tool for Maya, Maya plugin code, vertex reduction, and error estimation. To make the plugin usable it
also needs a user interface which is intuitive to use. In the beginning the code can be divided into two separate parts, the Maya wrap around and the reduction code. With both these two parts a complete plugin is achieved, in order to manipulate the mesh the decimation part of the code can then be written. Before starting with any of these steps the methods to use should be decided. By planning what should be used from the start the connections between the different parts of the program can be written more specific and make for an easier connection later on.

3.1 Choosing the methods

In order to make design decisions to the code and know what information needs to be obtained from the Maya workspace to the plugin code, decisions about what methods to implement to achieve the decimations needs to be made. The best way to do this is to start with the most complicated part of the process and then see what it needs to function. Maintaining the shape of the mesh is a complex mathematical problem, it makes the decision on which vertex to remove and where the new vertex should be placed. This is exactly what the error estimation is designed to do. By choosing the error estimation method first and then see what it needs to function a decision about the mesh structure and the reduction method can be made.

Since the goal is to keep the characteristics of the mesh as true as possible to the original mesh while still maintaining a relatively low computational cost many of the more simple methods can be disregarded as viable options. There are two main candidates that can achieve a decimation while still preserving the shape of the mesh in a satisfactory way, volume estimation and Garlands quadric error estimation. Of these two the quadric error estimation is the faster and gives the best result in keeping the overall shape of the mesh.

The quadric error estimation needs a way to traverse through the mesh to calculate the importance of each face, vertex, and edge. It also needs to be able to navigate through the one-ring-neighbourhood of a vertex to calculate the new importance of the vertices around the changed vertex. The mesh structure that offers the best navigation tools is the halfedge mesh structure. It supports navigation along all the edges and is able to ascertain information of the vertices and faces connected to the edge. Using an open source alternative to creating a halfedge mesh from scratch offer time saving both in computational and implementation time. Since OpenMesh has a more specialised direction towards halfedge meshes and has a good support system it is ideal for this implementation.
The choice of which reduction method to use is reduced to two options for the quadric error estimation. Depending on what meshes will be manipulated by the implementation, it is either pair contraction or edge collapse. Garlands error estimation uses pair contraction since it does not restrict the user to work on manifold meshes. This method is however more complicated. If the restriction is made to only handle manifold meshes edge collapse is cheaper computationally and simpler to implement. Since pair contraction is an extension of edge collapse and OpenMesh uses edge collapse in its built in polygon reduction it is a good choice with the possibility to extend to pair contraction and non-manifold meshes if the time allows.

3.2 Maya plugin

In order to preform operations on the mesh in the Maya workspace the data needs to be retrieved into the code. The Maya wrap around is the code in the implementation which communicates with Maya and wraps around OpenMesh and the error estimation to convert it into information that fits the mesh structure. This is done in two steps, input and output.

Before anything else is executed the information that is sent into the plugin needs to be identified. By using Mayas selection list the elements that are selected can be retrieved and stored. Their are two valid options of selection that this plugin can handle, component and object selection, with the help of looking what kind of selection is done in the selection list the code can distinguish between the two options or inform the user that an invalid choice have been made.

The operations that are done after the selection has been identified is similar with both component and object selection. In order to use the undo function in Maya the original mesh needs to be stored before it is manipulated in anyway, by setting the selected object back to the original mesh in the undo function all the changes are restored. If several undo calls are made Maya will remember what was stored in each step and restore the mesh for each step.

After the mesh have been stored away the conversion from the Mayas own mesh format MFnMesh to a OpenMesh halfedge mesh structure is made. MFnMesh is a regular polygon mesh structure containing polygons which is built up from vertices that are connected with edges. It also contains normals for each polygon and vertex and several more variables which are not rele-
vant to this implementation. The main difference between the MFnMesh and OpenMesh is that MFnMesh lacks a navigation tool to traverse through the mesh. This is supplied in the OpenMesh instance in the form of halfedges. These are simply made by splitting each edge into two, one for each vertex connected to the edge.

Even though OpenMesh uses a different structure this does not need any more input in the creation since each edge simply needs to be split in two. The data needed to create the OpenMesh instance is the same to create the MFnMesh, because these elements are the same the information needed from either instance is readily available.

The information needed from the MFnMesh object is; an array of all the vertices, an integer with the number of polygons, a list with the size of the number of polygons which contains the index of the connected vertices, and the polygons normals. With this information a halfedge OpenMesh object can be created which allows more freedom of navigation through the mesh and manipulation of the edges and vertices.

After the polygon reduction is done the mesh needs to be converted back into a Maya mesh and the model in the workspace needs to be updated. Creating the new Maya mesh uses the same variables which was extracted from it in the input part with modified values after the reduction has been made. This creates a Maya mesh object which then firstly needs to be connected to a model. By using the MFnMesh function createInPlace the new model is created where the original model was placed and inherit it’s information. Lastly the object is selected again since it was removed from the selection list when the object was replaced.

3.2.1 Getting the Maya plugin working

To create a plugin for Maya the programming environment needs to be set up to create a file compatible with Maya. This is done by changing the properties of the project. First the target extension of the file generated should be set up as a .mll file. These are the files that can be imported to the Maya plugin manager when Maya is running. The configuration type of the project should also be set to dynamic library(.dll) to generate the .mll file correctly. In order to use the Maya standard variables the libraries and include files should also be imported into the project.
Once the set up is done the actual Maya compatible code can be written. The plugin is using a different layout from a regular C++ application. The bare essentials to create a simple command plugin is a DeclareSimpleCommand() and a doIt() function. The command function takes three arguments, the name of the class, the name of the organisation owning this command, and the plugin version number. With the DeclareSimpleCommand function in place the class name can be used as a run call for this plugin from the MEL script editor in Maya. The doIt function is what happens when the call is made from Maya. Using just these two functions a simple “hello world” example can be made.

```c++
DeclareSimpleCommand(Name, Creator, Version)

doIt()
{
    Run code;
}
```

To create more advanced applications more standard functions can be added. A generally good way to start adding complexity is to also introduce an undoIt and redoIt function. These work similarly to the doIt function, and are called when undo and redo is made from the Maya workspace. By introducing these early complexity can be added as the application manipulates in more advanced ways. For instance if polygon reduction is made, the original mesh can be stored and sent to the undoIt function. In the undoIt the mesh can then be created again with the original vertices and polygons. A simple way to use the redoIt function is to put the entire code in the function, and then make doIt call the redoIt. This can however lead to problems, if the program takes time to execute every time the user presses redo the program will run again in its entirety. The selection might also have been changed if the plugin relies on what the user have selected in the workspace.

```c++
doIt()
{
    MeshToUndo = OriginalMesh;

    reduceMesh(OriginalMesh);

    MeshToRedo = ReducedMesh;
```
undoIt()
{
    setMeshTo(MeshToUndo);
}
redoIt()
{
    setMeshTo(MeshToRedo);
}

In order to make Maya load the plugin correctly in the plugin manager the code also needs an InitializePlugin() function and UninitializePlugin(). These will be called when the plugin is loaded into Maya. If there is certain circumstances that needs to be fulfilled to run the plugin they can be added into this stage and unmade in the Uninitialize. This might however affect the user without any warning and can therefore be unclear for the unaware user. If nothing needs to be added to this stage of the code, these functions can just be left empty.

The DeclareSimpleCommand() can be replaced by a more versatile command that allows for more advanced operations. This is called creator, which is a void pointer that will later call the doIt function. The creator is called when the command line is given in the MEL script in the workspace.

3.3 Polygon reduction with OpenMesh

In the initial OpenMesh implementation there were two main goals, create a mesh from coded vertices, and collapsing several edges on the mesh. The first part of the implementation involves defining a simple mesh. There are two ways of creating a mesh, either loading an existing obj file or defining the points of the mesh by creating vertices and connecting them into edges and faces. Loading a obj file was not relevant for this project since a conversion will be done from the data Maya supplies. The process of defining the mesh needs vertex points and how these vertices are connected to create faces. After supplying these to OpenMesh it creates the edges and halfedges between the vertices which enables navigation inside the mesh. When the mesh had been created OpenMesh can create a obj file of the mesh which can be viewed in Maya, this was helpful to get an understanding of how the OpenMesh operations manipulated the mesh.
Since the mesh now is created in such a way that it will function with the data that can be extracted from Maya the next step will be how the actual decimation of the mesh will be preformed. OpenMesh has a collapse function which will collapse an edge. An edge connects two vertices and if the edge collapses these two vertices will end up on the same point, i.e one vertex will be eligible for deletion. Thus the mesh has been decimated by one vertex. The collapse function in OpenMesh takes a halfedge as input. This is done because when OpenMesh collapse an edge it moves one vertex on another and then removes it, and since an edge does not have a direction one of its halfedges is used. Which one of the two halfedges is supplied to the function decides which of the two connected vertices are removed. Every halfedge is composed of a "to" and a "from" vertex, and when it is collapsed the "from" vertex is removed and all its connected edges are instead connected to the "to" vertex. An important note here is to test if the edges is actually eligible for deletion, some edges cannot be deleted without destroying the entire model. The check can be done by using another function which has a halfedge as input and returns a true or false statement depending on whether or not it is eligible for deletion.

![Figure 4: A figure depicting the different stages of a collapse with the help of OpenMesh.](image)

### 3.4 Reduction algorithm

When all the pieces is implemented into a working program an edge can be removed. To keep the shape of the model a way to find the edge which introduces the least amount of error if removed. This is where the error estimation algorithm will be introduced to the program. To find the least important edge to the model a cost will be assigned to each edge, a new position will also be calculated for all the edges which will determined the position of the remaining vertex. Garlands quadric error estimation was used
in this project. Garland's method involves creating quadric matrices first for the faces and then combining these to create quadric matrices for the vertices and edges.

To create the quadric matrices for the faces a point on the face in question, i.e. a vertex, and the face normal is needed. The plane equation is then calculated to find the constants $a$, $b$, $c$, and $d$, as shown in equation 8 and 9.

$$ax + by + cz + d = 0$$

(8)

where

$$a^2 + b^2 + c^2 = 1$$

(9)

After solving this a quadric matrix is created using $a$, $b$, $c$, and $d$.

$$
\begin{bmatrix}
  a^2 & a*b & a*c & a*d \\
  a*b & b^2 & b*c & b*d \\
  a*c & b*c & c^2 & c*d \\
  a*d & b*d & c*d & d^2
\end{bmatrix}
$$

(10)

As Garland writes in [2] the quadric can be used to find the squared distance of any point in space to the plane that it represents. If a quadric is summed together with another quadric they represent a set of planes by the new quadric which is created by the summation. Equation 10 represents a face of the mesh, these matrices will be used in the creation of the quadric matrices representing the vertices. This is done by summing together all the matrices for the faces around the vertex in question.

Once the quadric matrices for the vertices is made the cost needs to be associated to each vertex. This is done by defining the cost as

$$v^T Q v$$

(11)

where $v$ is a four dimensional vector with the position and the $w$ as 1.

To calculate the optimal position after two vertices have been contracted the sum of the two vertices quadric matrices is calculated. The goal is to find the position that introduces the least error into the system. Solving the equation where the derivative of $x$, $y$, and $z$ is equal to 0 results in the following equation.

$$v_{new} = \begin{pmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{11} & q_{12} & q_{13} & q_{14} \\ q_{11} & q_{12} & q_{13} & q_{14} \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

(12)
After the contraction the new position of the remaining vertex is then set to result of equation 12.

The error estimation is run once over the entire mesh before any manipulation is done. This creates a list of the vertices which is sorted by the estimated error for each vertex. This list forms a queue on which vertex to remove in each iteration of the reduction code. After a vertex has been collapsed the vertices in the neighbourhood needs to be updated with new error estimations and inserted back into the queue.

3.5 User interface

The goal of this plugin is to create a reduction plugin that anyone can use, not matter how little experience they may have with polygons and vertices. In order to achieve this a user interface goes a long way. Instead of manually having to input the different variables a python script was made with the minimum of complexity. Since the object or selective decimation is chosen dynamically depending on what the user have selected the only other options that needs to be made is how much reduction will be made. There are two options of choosing how much to reduce the model, either by giving a percentage or the number of vertices that the user want to remove. To distinguish between the two options two radio buttons exist that mark which method to use. A slider represents the amount of reduction that will be made on the mesh, for the percentage it varies between 0 to 100, and the vertices counter varies between 0 and the number of vertices in the selection or model depending on the choice.

![Figure 5: An image showing the user interface window in Maya.](image)

(more on how to create the script?)
3.6 Future work

The plugin is able to correctly reduce a mesh in an accurate way, there are however improvements that still needs to be made in order for the tool to be production ready.

The second largest issue of the implementation at this point is the undo and redo functions for selective decimation. The selection list is not transferred to the undo function. This will cause problems if the user decides to undo a selective decimation and then redo it again. Then the selections will be wrong since the selection list was not restored after the undo. This is related to the next issue.

In order to save time for the user the use of the doIt and redoIt functions in the code should be revised. Calling the redoIt function in the doIt is a easy solution for the program, but will cost the user time every time the redo button is pressed in the workspace since that will call the whole decimation to run again. To fix this the actual decimation should be moved to the doIt function and it should send the resulting mesh to the redoIt function which will be stored and created if it is called.

Another issue that should be addressed to avoid unnecessary crashes is to restrict the number of vertices the user can choose to decimate. The user interface does rely on the user not choosing an impossible choice of vertices to remove. If the user for instance chooses to remove 100 percent of the vertices in the mesh an infinite loop is created. This can also happen if the selective method is used and there are a lot of boundary vertices which will not be removed. To fix this issue an estimation or a exact calculation of how many vertices the mesh can be decimated by needs to be introduced into the python user interface script.

4 Results

In this section the results from running the decimation plugin on various models in Maya will be shown. The first part will present the decimation of the entire mesh for various models, the second part will focus on the selective decimation of various models and areas.
4.1 Decimation for entire meshes

The main focus of this implementation is to decimate a model while minimizing the error of the mesh, which will make the mesh keep its original shape as much as possible while vertices are being removed. This will make the mesh contain the key features as long as possible. In figure 6 the key features are contained until the second to last and the last where the number of vertices left are unable to give an accurate representation of the eyes and feet of the bunny. In the last image the vertex which creates the left ear is removed since it is deemed to introduce the least error to the model, at this point the mesh starts to lose its characteristics because there is too few vertices left.

Figure 6: Decimation on the Stanford bunny in 5 steps each reducing the mesh by 75 percent of the vertices. The original mesh on the left contains 34835 vertices and from left to right the other images contain; 8709, 2178, 545, 137, and 35 vertices.

The implementation is able to keep the shape of the mesh even under hard decimation, figure 7 and 8 represents a reduction of 95 percent of the mesh, the mesh is able to keep the essential contours of the original model with a reduction of 33093 vertices. It does smooth out the bumpier areas of the bunny and introduce some sharp edges in the curvier parts. Similarly in figure 9 and 10 there are some areas where the simplification makes smaller changes like the flat area on the bottom plate which suffers small changes visibly, but looses several vertices. While the face and the jewellery on the stomach loose its distinctive contours. Depending on the user and how the mesh will be used these visual simplifications might be acceptable or not. The algorithm does however clearly strive to keep the shape of the mesh, although the change that introduce the least error to the mesh might end up changing a big visual cue, like the shadow on the bottom plate that is introduced in figure 10. This can cause annoying flickering effects if the meshes would be used in a level of detail implementation, where the mesh suddenly pops.
Figure 7: A figure showing the Stanford Bunny model before(left) and after(right) the decimation with the wireframe. The original mesh contains 34,835 vertices and the decimated mesh contains 1,742.

Figure 8: A figure showing the Stanford Bunny model as in figure 7, before(left) and after(right), smooth shaded using the Maya inbuilt shader.
Figure 9: A figure showing the Stanford Budda model before (left) and after (right) the decimation with the wireframe. The original mesh contains 49,990 vertices and the decimated mesh contains 9,998.

4.2 Selective Decimation

The goal of this feature is similar to the decimation over the entire mesh, but it only focuses the decimation to the specified areas chosen by the user.
There are two options to using the selective decimation, either decimate the entire mesh as heavy as the selected area would be decimated or remove the same amount of vertices but over the entire mesh. In figure 11 the result of the three options is presented on the decimation on the ear of the Stanford bunny.
Figure 11: Comparison between the different options of decimating the ear of the bunny. In the far left image the bunny has been decimated by 2039 vertices, in the middle image the entire mesh has been decimated 95 percent and 33093 vertices have been removed, and in the far right image the ear has been decimated 95 percent and 2039 vertices have been removed.

The three images in figure 11 give varying results. The first image is very similar to the original mesh since a relatively small amount of vertices have been removed, ca 6 percent. The only changes that have been made are on the flatter surfaces of the ear. The second image on the other hand show a larger change since in that case 95 percent of the vertices where removed. This achieve a very similar result to the last image on the ear of the bunny, but also reduce the rest of the mesh similarly. In the last image a big change have been made but it is localized to the ear of the bunny and conserve the rest of the vertices of the mesh.

This can be used to preserve features that are important to the scene while removing unnecessary complexity from features that will have no impact on the image. In the figure 12 the entire body of the dragon is selected except for the head and then decimated. 33118 of the 50000 vertex points of the mesh are selected and reduced to 300 vertices. This will simplify the entire mesh, while still preserving all the vertices that form the head of the dragon.

If instead the 33118 vertices is removed over the entire mesh the features of the dragons head will be compromised as shown in 13. The face will be less smooth and contain sharper edges around the curves of the features.

5 Conclusion

The end result of the plugin is accurate and is able to give the user more options than the regular Maya built in decimation. It does not however improve on the execution time that the standard method is able to achieve for reducing the entire mesh. The plugin does also rely on the selection and
Figure 12: In the right image the selected vertices that will be decimated is shown in yellow. These 33118 vertices is reduced to 300 vertices in the right image. The border between the selected and not selected vertices is preserved to avoid changing any of the features of the not selected vertices.

Figure 13: Head shot of the dragon after selective decimation in figure 12(left) and after a decimation over the entire mesh by the same amount of vertex points(right).

input from the user, if the user enters the wrong input it might result in an unstable process.
The implementation is not as efficient as it could be with further optimization of the code, there are ways to minimize the amount of data that is being processed in each iteration. The time constraints did however not allow for this to be implemented.

The process of creating this plugin would be smoother and easier if a change in operative system was made to linux from windows. The linux version of Maya allows for more precise error messages and libraries and includes to the code would be included smoother.

The error estimation process could be more complex with further mathematical operations that would be fairly straight forward to introduce, as long as they work on a half edge mesh system. Which would give the user more options and control over the mesh. There are several ways to introduce more artificial weights that could influence the error estimation. Either mathematically or feature depending with the help of Maya functions.

5.1 Implementation challenges

During the implementation of this project several compromises have been made. In this section issues, design choices, and compromises will be discussed and explained. There were a lot of challenges during the making of this plugin, some had a satisfactory resolution, and others where resolved with a quick fix that is more or less stable.

A problem that occurred a number of time was a null pointer exception that caused Maya to crash. If a vertex reduction was made in two steps without selecting new vertices in between Maya would crash. This was because the selection list of vertices was wrong. The vertices would change index since some was removed and a new mesh was created. In order to fix this, the position of all the remaining vertices was stored and then a new selection list was made out of each vertex in the new mesh that had a corresponding position to one of the remaining vertices. This eliminated the crashes while also making sure that all the correct vertices was selected after a reduction was made.

A design choice that was made was to make the choice about whether or not the boundary vertices in a selective decimation was removable or not. There is two options when implementing selected vertices decimation. One approach is to decimate all the selected vertices this would potentially also
change edges that are connected to vertices not selected. The other way is to not decimate any of the vertices on the boundary of the selection, i.e all the selected vertices connected to a vertex not selected. This will give the user a more controlled area of decimation, it might be less intuitive to use since you would like to decimate all the selected vertices. In this program the second option is used, since it would give more control to the user over the model and potential important edges and vertices will not be moved. In figure 14 the effect on the connecting polygons are shown in the left image. This is a design choice, if the user is aware of the effect and chooses the vertices with this in mind both methods works similarly.

Figure 14: An comparison between not reducing the boundary vertices and doing so. The area marked by a red circle shows the effect it produces on polygons connecting to vertices that were not selected.

This implementation of decimation used a iterative process of removing vertices, the iteration will try to remove a vertex in each run through. In some situations the edge that is meant to be collapsed will not be eligible for collapse, this will result in a iteration that does not remove anything. This can be a problem if the user have a specific desired amount of vertices in mind, by having a iteration that does not remove a vertex will result in a model with one more vertex than desired. Solve this problem the implementation checks if something was removed in each run through of the loop otherwise it will not count the run through as a iteration. Additional problems can occur with this approach, depending on the case the loop might become infinite if the remaining edges are not removable. Another more expensive solution to the iteration problem would be to add another loop inside that goes through all the the edges until it finds the next cheap edge that is eligible for collapse, if no edge was found then break the loop and inform the
user that the decimation can’t continue further. It will potentially be slower than the more trivial approach, but more stable.

After a decimation is done the program will create a replica of the original model in the place of the previous model. This generates problems when components are selected before the decimation is done. These selections will be removed when the new mesh is created. Also some of the components will have been removed in the decimation when selected vertices decimation is run. Another problem with this approach is that many of the index numbers of the vertices will be changed in the garbage collection of OpenMesh. The only information left to select the correct remaining vertices is their position, by comparing the previous positions with the ones in the new mesh the selected vertices can be reselected. It is computationally heavy to check all the remaining selected vertices with all the vertices in the new mesh, a option is to just let the plug-in deselect all the vertices to save run time, it depends on the end users requirements.

**References**


