Simplification of Meshes in OpenGL using Quadric Error

**Andrea Pulita**

**Real-Time Graphic Computing**

**2023-2024**

Summary

[Introduction 2](#_Toc160583047)

[Level of details 2](#_Toc160583048)

[Simplifications 2](#_Toc160583049)

[Local Simplifications 2](#_Toc160583050)

[Simplification algorithms 3](#_Toc160583051)

[Geometric simplification 4](#_Toc160583052)

[Quadrics 4](#_Toc160583053)

[Quadric error metric 4](#_Toc160583054)

[Implementation 5](#_Toc160583055)

[Main Loop 5](#_Toc160583056)

[Data Structures 6](#_Toc160583057)

[Algorithm 7](#_Toc160583058)

[Performance e other solutions 8](#_Toc160583059)

[User Manual 8](#_Toc160583060)

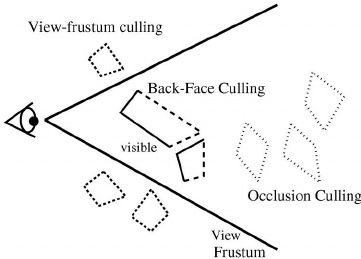
[References 9](#_Toc160583061)

## Introduction

In recent years the graphics power of computers and consoles has increased exponentially, allowing the use of applications with increasingly powerful and detailed graphics. At the basis of these applications, in addition to the graphics engines, there are geometric models called polygonal meshes. These models represent complex three-dimensional figures, in most cases formed by numerous triangles joined together. The more detailed the figure, the more numerous and difficult the triangles that compose it are to represent on the screen. One of the main problems therefore is finding the right relationship between detail and performance.

## Level of details

To achieve a good degree of realism without weighing too much on the performance of the application, various techniques have been developed over the years to lighten the load on the graphics card and maintain constant and realistic quality. The main techniques concern hiding the details of figures not present on the screen and in the field of view, eliminating hidden parts of elements not visible because they are behind other figures, applying detailed textures on simple meshes and reducing the quality of elements distant from the observer using, for example, detailed meshes in the foreground and use simpler meshes of the same object in the background. Immagine che contiene disegno, schizzo, testuggine, tartaruga

Descrizione generata automaticamente 

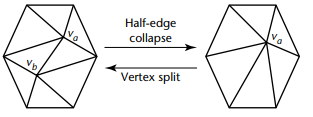
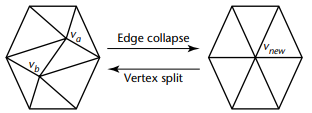
## Simplifications

This project takes advantage of the simplification of the meshes to create figures that are simpler to show on the screen. The theoretical basis behind this operation includes local simplifications and global simplifications. Furthermore, it must be considered that simplification can be divided between simplifications purely based on the sides to be eliminated and simplifications based on the fidelity of the figure you wish to obtain. While the former is based solely on the principle of decreasing the sides by minimizing the error obtained, the latter simplifications try to keep the figure as similar as possible to the original, producing (at worst) various simplified figures, each with an increasingly larger error. In both cases, the measured error varies depending on the method and the type of simplification desired.

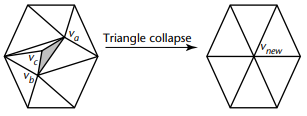
## Local Simplifications

Local simplifications allow the complexity of the mesh to be reduced and the main ones are:

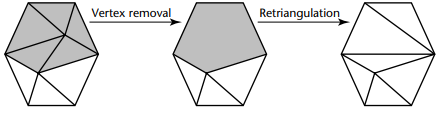
* Edge collapsing: this operation allows you to collapse two vertices, joined by an edge, Va and Vb into a new vertex Vnew. This operation also makes the triangles that had that side disappear. This technique is divided into two common uses called “Half-Edge Collapse” and “Full-Edge Collapse”. The first collapses the two vertices into one of the two previous ones, therefore Vnew = Va or Vb, while the second allows you to use a new vertex. This is also the technique used in the project.

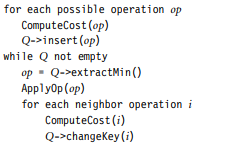
* Collapsing triangles: this operation allows you to collapse a triangle with vertices Va, Vb and Vc into a new vertex Vnew which can be one of the previous ones or a new vertex. This technique is faster but allows less control of detail.



* Removing a Vertex: This operation removes a vertex and all edges and triangles associated with it. This will then result in a hole in the mesh which will then have to be triangulated again through another algorithm.



## Simplification algorithms

These local simplifications are used by various algorithms to create various layers of new, increasingly simpler figures with fewer vertices. The simplest algorithm is obviously to apply a local simplification to the new figure obtained until reaching the desired level. However, this approach does not take into account any factors and for this reason the algorithm used by the program uses a greedy algorithm which exploits the calculation of an error to find the most convenient operation to use. This pseudocode explains how our algorithm works: we start by assigning each side of the triangulation a positive real number, later called "error". These numbers are sorted into a queue. At the end of this operation, the smallest error is considered: if this number is above the threshold set a priori or the queue is empty, the procedure ends. Otherwise, a local simplification is applied by contracting the side corresponding to the value mentioned above. Then the errors are recalculated for all the sides involved in the contraction, since after collapsing the side the error of the neighbors will change, and we start again from the beginning.

## Geometric simplification

We now need to explain how to associate the "error" with each side of the triangulation. First, we need some premises about quadrics.

## Quadrics

A quadric is a surface that can be expressed as the locus of the zeros of a second-degree polynomial in x, y, and z. Quadric surfaces can be represented in Euclidean space as quadric surfaces in two dimensions of the form ax2+ 2bxy + 2cxz + 2dx + ey2 + 2fyz + 2gy + hz2 + 2iz + j = 0. Furthermore, this surface can also be expressed in the form where and are the row and column vector [x y z 1] and Q is a 4x4 matrix formed by

### Quadric error metric

There are various measurement possibilities when it comes to the error to be used in these algorithms and the one used in our case is to evaluate the distance between a vertex and a plane. Given a plane p with equation ax + by + cz + d = 0 and a point v in space, the distance between the point and the plane turns out to be the absolute value of the scalar product between the vector p of the coefficients of the plane and the vector v of the coordinates [x y z 1] of the point, considering.

Each vertex of the figure to be simplified will have a support plane for each triangle of which it is the vertex. In our algorithm, the error that is used and associated with each side is calculated in this way:

1. the vertex V is determined into which the side in question should collapse (which may be a vertex already present in the triangulation or not, see later);
2. the distances between V and all the support planes of the two vertices of the side are calculated;
3. these distances are squared and added.

To carry out the above explanation, however, we first assign an error △(v) to each vertex of the triangulation in this way:

This 4x4 matrix is called the fundamental quadric error and is used to find the squared distance of any point from a p-plane. An interesting feature is that these matrices can be added together to represent with a single matrix the sum of the distances (squared) relating to a set of support planes at a single vertex v.

At the beginning of the algorithm, for each vertex v, we determine all its support planes p, we determine this matrix Kp, for each p, and we add them all together to obtain a matrix Q(v). For each side of the triangulation, of vertices v1 and v2, which could collapse into a point vnew:= V, the relative "error", i.e. what we will attribute to the side, is calculated by adding the two matrices Q1 := Q(v1) and Q2 := Q(v2) and using their sum like this:

To complete the algorithm, all that remains is to explain how to choose, for each side of vertices v1 and v2, the vertex V to use as vnew for the calculations described above. For each side of vertices v1 and v2 we calculate:

* the error relating to v1, using only the planes of which v2 is the vertex;
* the error relating to v2, using only the planes of which v1 is the vertex;
* the error relative to the midpoint of v1 and v2, using the union of the planes of which v1 and v2 are vertices.

V will be the point, among the three considered, for which the corresponding error is smaller.

## Implementation

The program consists of a main part where the game loop is located with the various calls for managing the graphic environment, the menu and the models. Then we have the part of structures for managing the meshes to be used within the algorithm. Finally, we have the part of the code that deals with the execution of the algorithm with related errors.

### Main Loop

The main loop consists of two parts in the app.cpp file.

In the first the various checks are made to manage the various menu variables to perform certain actions. Below some examples:

* the change of the current mesh.
* the simplification animation or the standard execution.
* the choice to show the rigid or smooth mesh.
* the rotation animation.
* the wireframe view.

In the second part we configure and pass the variables to the shaders and show the models and the skybox on the screen.

### Data Structures

To apply this type of algorithm, the most optimal solution for working with meshes is to use the half edge structure because it allows us to work quickly with adjacent vertices and edges. For this reason I have implemented a structure that saves for each half edge its previous one, its next one, the face it refers to and the vertex it points to.

Immagine che contiene testo, schermata

Descrizione generata automaticamente

Finally, the initial mesh is then analyzed and transformed into a mesh that contains these new half edges. In this way the new half edge mesh can take care of management, simplification and conversion.

Immagine che contiene testo, schermata, software, Sistema operativo

Descrizione generata automaticamente

### Algorithm

The algorithm is executed as described above in a faithful manner. Programmatically the algorithm is divided into a structure that calculates and saves the error for each half edge and the main loop in the simplification.h file.

This is the structure that saves the half edge and the error associated with it. At initialization the error is calculated through the points of the half edge and the two matrices linked to the two vertices.

Immagine che contiene testo, schermata, software, schermo

Descrizione generata automaticamente

The main simplification loop consists of some very specific steps. At the begin, the matrix Q is calculated for each vertex position. Subsequently, for each half edge the error is calculated using the Qs and added to a set in order of lowest error.

Immagine che contiene testo, schermata, software

Descrizione generata automaticamente

Then we take the half edge with the lowest error from the set and collapse it, removing the faces associated with it and the related vertices. In this way all the errors of the adjacent vertices will be modified and for this reason they will all have to be updated.

## Performance e other solutions

In terms of performance, the algorithm implemented is not the best, but it still gives good results and focuses more on the fidelity of the mesh to the original. Other alternatives at the moment are the fast edge collapsing with quadric error algorithm that uses the same principle, but does not update the quadrics at each pass, obtaining a faster result and still very similar to the original. Other viable alternatives are Blender's algorithms such as the decimate modifier, ParallelQSlim that uses the same error and the power of multithreading to make the algorithm faster and the MeshLab Quadric Error Collapse Decimation and Surface Semplification with Local Geometric Error.

In the realization of this algorithm, I tried to use the best method to use in the most expensive part of the process, i.e. updating the sides with each collapse and extracting the edge with the lowest cost. For this reason, I have tried some data structure solutions that can be used, including the C++ basic Set, a min-heap implemented in an external library and a basic approach of a C++ vector reordered each time. In conclusion, the most optimal choice was to use the basic set with a custom ordering on the error that led to an automatic update of the tree at each collapse.

The table below shows the times and the initial and simplified faces of some of the algorithms tested. Obviously, the data was taken from small meshes as my implementation, not being very optimized, has much lower performance on very large meshes as the complexity increases according to the number of faces of the mesh.

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | Initial Faces | Removed Faces | Time |
| This project | 10000 | 7000 | 1sec |
| Fast Quadric Error | 10000 | 7000 | 100ms |
| Blender Decimation | 10000 | 7000 | 250ms |
| MeshLab LGE | 10000 | 7000 | 200ms |

## User Manual

Once the program has been started, the user will find this screen. In the centre there is the mesh to be simplified. At the top you will find the menu to modify the graphic aspects and start the simplification. Furthermore, you can also use keys and the mouse to navigate the scene and apply some changes to the scene.



In the initial menu you will find the various keys you can press while you can change panels to see the simplification settings and scene settings. Here below there are the various commands that can be used:

* WASD + Mouse: navigate the scene.
* 1, 2, 3, 4: change the shader.
* ESC: quit the program.
* G: show the mouse for interacting with the menu.
* L: show/hide the wireframe.
* E: collapse one single edge if possible.
* P: start/stop the rotation of the figure.

In the simplification menu you can choose to collapse any number of edges, set an error for which the figure remains faithful to the original and if you want to see the animation or show the final figure directly.

In the graphics menu you can choose the values ​​you prefer for the shader.

## References

* <https://faculty.cc.gatech.edu/~turk/my_papers/memless_tvcg99.pdf>
* <http://mesh.brown.edu/DGP/pdfs/Lindstrom-vis98.pdf>
* <https://www.cs.cmu.edu/~./garland/Papers/quadrics.pdf>
* <https://github.com/sp4cerat/Fast-Quadric-Mesh-Simplification/tree/master>