3D Model Deformation using Finite Elements

Michael Cairns

August 20, 2012

Abstract

In recent years major effects houses have started to use the finite element method to better model soft bodies. As more computing power has become available this technique has become more feasible in the timeframe of a production. The objectives of this project were to investigate how finite elements are used to model and simulate deformation in visual effects. Specifically, linear elasticity was covered and the beginnings of plastic deformation.

A program was created in C++ which allows a user to load a surface mesh and by setting material properties and applying forces, create a simulation. The results show how physically correct animation can be produced using the finite element method although it is mentioned that knowing exact material properties helps.

Although no new solutions have been proposed in this work it was more to show how the method can be implemented. It was hoped that this project could progress beyond elastic deformation to crack generation and object splitting but the amount of work required was underestimated. It is something that should definitely be investigated in the future though.

Acknowledgements

I would like to thank Jon Macey and Mathieu Sanchez for their help and inspiration throughout the year.

Contents

1

Introduction

Creating realistic images is something that many visual effects companies strive for. To be able to convince an audience that what they are seeing is real when it is not could be described as magical. For almost half a century film makers have been using computers to create images of places or people that do not really exist in order to make their films more immersive. The phrase “smoke and mirrors” has been used to describe the work done both in camera (special effects) and on computer systems (visual effects). Hilf (1997) describes the use of computer generated imagery as “pixel manipulation, transposed motion capture, and digital mattes” which is somewhat correct but modern CGI incorporates modelling 3D objects, simulating light reflections and much more. (Erleben et al. 2005)

There are several areas of visual effects that require attention if realistic images are desired. One is rendering, making an object look like it would look in the real world. Another is animation, the movement of an object must be natural and follow the laws of physics otherwise a human will very easily see that something's wrong. The idea of simulating natural phenomena is called physically based animation. Common examples include rigid bodies, soft bodies, fluids and joint systems. Soft body simulation involves modelling the deformation of objects under forces. This project is focused on deformation of objects, specifically using the finite element method.

Several methods exist for simulating deformation. Free-form deformations involves surrounding the object with a grid or lattice that the user controls by moving control points. Each grid square has influence over the part of the model within it. Generally an interpolation is used so influence decreases further from a control point, this allows a smoother movement. Most methods utilise B-Spline theory to create curves that the object deforms over. The mass-spring model is a common model for cloth simulation although it can be used for 3D meshes as well. This method involves connecting springs between neighbouring nodes and using Hooke's law with an integrator to calculate the nodal movement. Dampers are also added to counteract the spring force. These two methods are not the most physically accurate. FFD does not take into account the material properties and is more suitable for modelling a particular shape. The mass-spring model can model certain material types by tweaking the spring and damping parameters and by creating unique networks of springs. However it does not use extended material properties such as Poisson's ratio or the temperature.

The FEM is a structural engineering technique that was developed in the middle of the 20th century to solve problems involving complex structures. The basic concept is to take a continuous model and split it up into discrete domains to simplify the equations that act over the whole body. The system is solved by combining the result of simpler equations applied to the separate domains. Although discovered as an engineering technique the method has been used as a general numerical method for solving partial differential equations.

This project looked at the use of the finite element method to simulate deformation of an object in 3D. Specifically an implementation of a program in C++ that allows an artist to load in a model, apply forces and render out animation frames for use in a visual effects sequence. A real-time or close to real-time display of the model means the artist can see the effect and adjust values accordingly. The Qt framework was used for GUI development as well as OpenGL integration. The NGL library provided a wrapper to OpenGL calls and for some data types. A git repository was setup with an online remote for piece of mind and to make it easier to track the progress of the project through commit messages.

In this project we will assume that the material being modelled is elastic and behaves linearly. We also assume that the material is isotropic.

In chapter 2 we will highlight previous work in this field, focusing on computer graphics and visual effects.

In chapter 3 we will explain the theory of elasticity and plasticity along with how finite elements are used in this area. Formula used in the implementation are also presented and described.

In chapter 4 the software design and programming work will be presented with justification for using certain methods.

In chapter 5 sample models and the output images will be shown along with material properties used and data from the generated mesh.

In chapter 6 we formulate conclusions from the work carried out and put it into perspective with recent research and media.

2

Previous Work

Terzopoulos et al (1987) published some of the earliest research on using finite element techniques in computer graphics. They present an elastic deformation system that can be applied to 2D and 3D meshes. Although they use a finite differencing method it has many similarities to FEM and their contribution to simulating deformation is significant.

O'Brien and Hodgins (1999) introduce work on animating fracturing in objects, based on elasticity theory. Using finite element based deformations they calculate internal forces at all the nodes and if a defined threshold is crossed then the node is split into two nodes and a fracture plane is calculated. This determines the direction of the crack. In their method they use local re-meshing to keep good element proportions when they are split.

Müller and Gross (2004) present an interactive finite element implementation that shows elasticity, plasticity and fracturing effects in real-time. They avoid the artefacts of linear elasticity by introducing a warped stiffness matrix which takes into account the rigid body rotation of each element. They also implement a fracturing system by using calculating the tensile stress of each element and generating a crack if it is higher than chosen threshold.

Another paper based on O'Brien's work describes the use of previous work in a real-time game environment (Parker and O'Brien 2009). They present the development of a product now released by Pixelux and first licensed for the video game The Force Unleashed. They describe how they implemented previous work but optimised it for games consoles and managed to keep within polygon count and frame rate limitations. Furthermore they experimented with parallelising certain parts of the algorithm to reduce computation time.

Bargteil et al (2007) present research on large plastic deformations. This type of deformation is usually unobtainable with a linear elastic system of equations. Their model can be used to simulate soft solids such as dough or clay. They achieve their results by updating the basis functions of the elements and re-meshing the model to keep a high-quality mesh. This avoids the numerical instability that was previously a problem. The results of their work is amazing; a large variety of material types can be simulated and even major topological changes such as the merging of two objects are possible through the use of dynamic re-meshing.

A method of generating convex elements is presented in (Wicke et al. 2007). Their work is a progression from linear tetrahedra by utilising twelve nodes per element allowing a more flexible basis function. This allows the shape of the model to be changed without expensive re-meshing used in other methods. They show results of accurately slicing a model, instead of always having to generate new tetrahedra they can add new nodes to existing elements if the mesh accepts it.

3

Technical Background

3.1 Physics

Elasticity describes a material that can undergo deformation and then return to its original shape. A common example is a spring which can be stretched longitudinally but it will return to its original shape when let go. Two measurements that are commonly used are stress and strain. Stress is measured as force per unit area; strain is measured as the proportion of the extended length to the original length, thus it is unit-less. The theory of elasticity originally came from Louis-Marie-Henri Navier, Simon-Denis Poisson and George Green (Gould 1994).

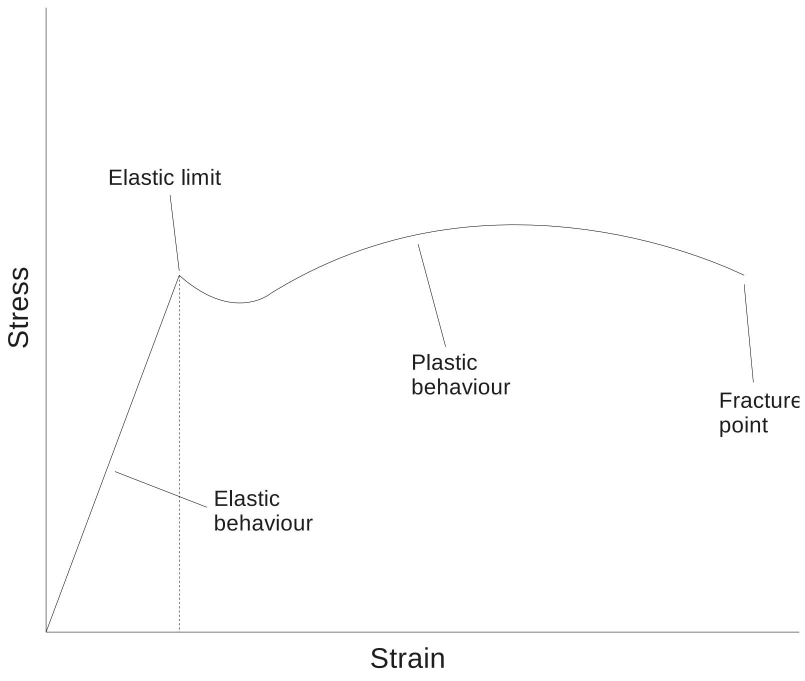


Figure 3.1: Stress-Strain curve.

Linear elasticity is a simplification of the elastic model that assumes a linear relationship between stress and strain and that only small deformations will occur. A stress-strain curve is a common way of displaying various material properties related to elasticity and plasticity (see figure 3.1). Hooke's law states that the stress is proportional to the strain below the point of proportional limit, where the graph is a straight line. The constant of proportionality, k, is called the Modulus of Elasticity or Young's Modulus.

$$\sigma = E\epsilon \tag{3.1}$$

The law also defines a relationship between force and displacement:

$$F = -ku \tag{3.2}$$

In linear elasticity the structure of a material can be categorised as: isotropic, meaning the properties are the same in all directions; orthotropic, where the properties are different in three perpendicular axis; and anisotropic, where the material properties are different in all directions. An example of an isotropic material is glass, examples of orthotropic materials are wood or carbon fibre. The anisotropic form of Hooke's law requires three values for the Young's Modulus and Poisson's ratio. Six stresses are used, three normal stresses and three shear, each in x, y and z. Six corresponding strains are also used (MacDonald 2011). The constant of proportionality can be represented as a matrix,

$$D = \begin{bmatrix} D_{11} & D_{12} & D_{13} & D_{14} & D_{15} & D_{16} \\ D_{21} & D_{22} & D_{23} & D_{24} & D_{25} & D_{26} \\ D_{31} & D_{32} & D_{33} & D_{34} & D_{35} & D_{36} \\ D_{41} & D_{42} & D_{43} & D_{44} & D_{45} & D_{46} \\ D_{51} & D_{52} & D_{53} & D_{54} & D_{55} & D_{56} \\ D_{61} & D_{62} & D_{63} & D_{64} & D_{65} & D_{66} \end{bmatrix} \tag{3.3}$$

The values of the matrix are calculated using the Young's Modulus and Poisson's Ratio values. For an isotropic material the matrix is simplified and sparse with only three unique values. For an anisotropic material all the values are different although the matrix is symmetrical so only twenty one unique values are required.

Going back to the stress-strain curve, once the stress reaches the elastic limit it will start to behave non-linearly and the deformations will become permanent. This is known as plasticity. The behaviour of plastic deformation varies greatly between materials. The properties ductility and malleability are aspects of plasticity and indicate how much a material can be stretched or compressed to change the shape. Examples of materials that exhibit obvious plastic deformation are clay, dough and aluminium. Another important material property is Poisson's ratio which is the ratio of contraction to extension. The effect of a material getting thinner as it is stretched or wider as it is squashed is called the Poisson effect. For an isotropic, linear elastic material it cannot be less than -1.0 or greater than 0.5.

Fracturing is the result of the stress on a material reaching the yield strength. At this point cracks will start to form as the material breaks into pieces. The physics of fracture forming is a whole different topic and is related to chemical composition and the location of any impurities and defects. O'Brien and Hodgins' (1999) method calculates the internal forces at each node and if they are working to pull the node apart then two nodes are generated in its place and a fracture plane is computed. Any tetrahedra surrounding the node are split along the fracture plane and their shape functions are recalculated. This will be easier to understand after the explanation of the Finite Element Method(FEM).

3.2 The Finite Element method

From a mathematical point of view the Finite Element Method(FEM) is a way of best approximating the solution to a problem, specifically those involving partial differential equations as well as integral equations (Henwood and Bonet 1996). It comes under the branch of mathematics called calculus of variations, the aim of which is to find x for the function $f(x)$ at stationary points, normally maxima or minima in practice. The Euler-Langrange differential equations were important discoveries in this field. A more detailed description of calculus of variations can be found in (Erleben et al. 2005). When discretising a domain a set of functions are defined which can be called piecewise linear basis functions. A full mathematical explanation is beyond the scope of this work but can be found in many textbooks .

As mentioned earlier the concept of elasticity is that the object tries to return to its original shape; the deformation is not permanent. This is a case of energy minimisation; during deformation, potential energy is built up in the object and an equilibrium is only reached when the object returns to its natural shape. Internal elastic forces are caused by the potential energy and produce movement back to equilibrium.

$$F_{elastic} = \frac{\delta \mathcal{E}(r)}{\delta r}, \tag{3.4}$$

where $\mathcal{E}$ is the energy and $\mathbf{r}$ is a point in the object.

The following equation is from (Erleben et al. 2005) where a full derivation is available.

$$\mu \frac{\partial^2 \mathbf{r}}{\partial t^2} + \gamma \frac{\partial \mathbf{r}}{\partial t} + \frac{\delta \mathcal{E}(\mathbf{r})}{\delta \mathbf{r}} = \mathbf{f}(\mathbf{r}, t), \tag{3.5}$$

where μ is the mass density, γ is the damping density, $\mathbf{r}$ is the position of a node, $\mathbf{f}$ is the applied external forces and $\mathcal{E}$ is the potential energy due to elastic deformation. The equation describes how the motion of nodes and the material properties influence the external forces. This equation is actually a form of Newton's second law. Notice that the first part is mass times acceleration, the second part is the viscous force and the third, elastic force. these forces can be summed together to attain something like $F = ma$.

The practical application of the FEM in engineering is called Finite Element Analysis(FEA). The first part of FEA involves discretising the domain into a finite number of elements. In the case of visual effects and animation most models are stored as 2-dimensional surface meshes so generating a 3D mesh is not too difficult. Delaunay triangulation is commonly used for this process but other methods exist for different element types and input geometries. Further information can be found by reading (Zhang et al. 2010) which describes a new method for generating meshes of different element types across various materials.

In this project linear tetrahedra will be used, that is, a polyhedron with four vertices and six edges. It's also known as a triangular pyramid. Higher order tetrahedra can be used such as quadratic which have ten vertices, the extra vertices are for controlling the curvature of the edges allowing more complex shapes to be modelled. The downside is the shape functions are more complicated making computations involving them more expensive.

In a 3D tetrahedral mesh, nodes are shared between elements so each node has four local indices but each node also has a global index for the whole mesh. Barycentric coordinates can be used to define any point within a tetrahedron: $w_0\,w_1\,w_2\,w_3$; they should sum 1 and w_n equals 1 at node n and zero at all others. The volume of a tetrahedron is defined by:

$$V = \frac{1}{6}\begin{vmatrix} x_1 - x_0 & x_2 - x_0 & x_3 - x_0 \\ y_1 - y_0 & y_2 - y_0 & y_3 - y_0 \\ z_1 - z_0 & z_2 - z_0 & z_3 - z_0 \end{vmatrix} \tag{3.6}$$

Strain can be measured using the deformation of a node from its original position to the new position. In 3D we use the linear Cauchy strain matrix

$$\varepsilon = \begin{bmatrix} \varepsilon_{00} & \varepsilon_{01} & \varepsilon_{02} \\ \varepsilon_{10} & \varepsilon_{11} & \varepsilon_{12} \\ \varepsilon_{20} & \varepsilon_{21} & \varepsilon_{22}, \end{bmatrix} \tag{3.7}$$

where

$$\varepsilon_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \tag{3.8}$$

Because the matrix is symmetric some values can be removed and in fact by factoring out the displacement, u, it can become

$$\begin{bmatrix} \frac{\partial}{\partial x_1} & 0 & 0 \\ 0 & \frac{\partial}{\partial x_2} & 0 \\ 0 & 0 & \frac{\partial}{\partial x_3} \\ \frac{\partial}{\partial x_2} & \frac{\partial}{\partial x_1} & 0 \\ \frac{\partial}{\partial x_3} & 0 & \frac{\partial}{\partial x_1} \\ 0 & \frac{\partial}{\partial x_3} & \frac{\partial}{\partial x_2} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \tag{3.9}$$

Stress is a force per unit area, as mentioned previously there are six stresses in 3D. A 3x3 symmetric matrix is used to store the values but again this can be condensed to six values. Stress is related to strain through the elasticity matrix (from page 8); here we show the isotropic variation which is sparse and has three unique values:

$$D = \frac{Y}{(1+v)(1-2v)}\begin{bmatrix} 1-v & v & v & 0 & 0 & 0 \\ v & 1-v & v & 0 & 0 & 0 \\ v & v & 1-v & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1-2v}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1-2v}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-2v}{2} \end{bmatrix}, \tag{3.10}$$

where Y is the Young's Modulus and v is the Poisson's ratio. For efficiency the three values can be stored simply as D_0, D_1 and D_2.

As mentioned before the system will resolve to equilibrium by balancing the forces at each node. In the system there are stress forces (internal), nodal forces and load forces (external); to solve the system at equilibrium we use a stiffness matrix. The end result is we want to solve for u in $Ku = f$, where K is the stiffness matrix, u is a vector of displacements and f is a vector of external forces. In FEA a stiffness matrix is computed for each element and then they're summed together to form a global stiffness matrix. The element stiffness matrix is 12x12 and is constructed like so (the derivation is left out):

$$K^e_{nm} = V^e \begin{bmatrix} D_0 b_n b_m + D_2(c_n c_m + d_n d_m) & D_1 b_n c_m + D_2 c_n b_m & D_1 b_n d_m + D_2 d_n b_m \\ D_1 c_n b_m + D_2 b_n c_m & D_0 c_n c_m + D_2(b_n b_m + d_n d_m) & D_1 c_n d_m + D_2 d_n c_m \\ D_1 d_n b_m + D_2 b_n d_m & D_1 d_n c_m + D_2 c_n d_m & D_0 d_n d_m + D_2(b_n b_m + c_n c_m) \end{bmatrix} \tag{3.11}$$

The stiffness matrix contains values relating to the coordinates of each node in a tetrahedron. The global stiffness matrix contains values for all the nodes in the model thus the individual element matrices will overlap where there are shared nodes to produce the global matrix. Calculation of the global stiffness matrix is achieved simply by iterating through each element matrix, converting the node indices to global and adding the values to the global matrix at the applicable position. One of the issues with the Cauchy strain tensor is rotations can cause artefacts in the deformation. Rather than use a higher order strain tensor, which would lead to more complex computations, we separate the element rotation from the deformation. To do this we use the method introduced in (Müller et al. 2002); rigid body rotation of the element is calculated separately and then multiplied by the stiffness matrix to produce the warped stiffness matrix. We now introduce the mass matrix which is used for calculating intertia and damping. A lumped mass matrix gives the benefit of simple storage as the matrix is diagonal so the masses can be stored with each node. Assuming a constant density across the material the mass per element is ρV^e and for each node we just divide by 4. Note that shared nodes will have mass contributions from all surrounding elements.

Finally we look at plastic deformation. This means the position of the nodes is permanently changed by the external forces. By using a threshold value we can add to the plastic strains for each element if the total strain reaches a certain amount. We can also limit the plastic strain to a maximum value. A plasticity matrix is used to relate the plastic strains to plastic forces: $P^e_n = V^e B^T_n D$, where B and D are the same matrices used to contruct the stiffness matrix, K. Plastic forces are calculated by multiplying the plastic strains by the plasticity matrix and the rotational matrix.

In computer animation the deformation is modelled dynamically; to render per frame a time-step, dt, is used. The equation to solve is:

$$M\ddot{x} + C\dot{x} + K(x - x_u) = f_{ext} \tag{3.12}$$

The matrix C can easily be calculated by multiplying M by a factor of 0.2 as the contribution from the stiffness matrix is normally set to zero. The equation above can be re-arranged into a linear system like so:

$$Av(t+1) = b, \tag{3.13}$$

where

$$A = (M + \Delta t C + \Delta t^2 K), \tag{3.14a}$$

$$b = Mv(t) - \Delta t(Kx(t) + f_u - f_{ext}) \tag{3.14b}$$

and v is the velocity vector which is used to calculate the new positions by an integration method (Euler or RK4 typically). To solve the linear system in equation 3.13 the conjugate gradient method is used. This is an iterative method that works by approximating the direction to the solution at each step, getting closer each time. The exact solution is found after n iterations where n is the order of the system but generally the approximate value converges quickly after less iterations (Saad 2003).

4

Implementation in C++

4.1 Software Design

Given the description and objectives of the project the class types below were initially considered:

- Model3D
- Element
- Node
- MainWindow : QMainWindow
- GLWindow : QGLWidget

The Model3D class is named this way because C++ does not allow type names to start with a digit. It encompasses everything about a 3D model including material properties, it also includes the Element and Node classes by aggregation. The Element class contains element specific data such as the stiffness matrix, volume, shape matrix and a list of indices to nodes that make up the element. The Node class holds data such as position, mass, and the forces. In order to accommodate the large matrices required a matrix class was created that acts as a wrapper to `boost::multi_array` from the Boost library. The class includes operator definitions to allow eligible equations to be written in code. Figure 4.1 shows how the different classes are associated.

The NGL library was used to parse Wavefront obj files, a basic parser for surf files was also written. Similar to obj files, surf files only hold a list of vertices and the faces defined with indices to the vertices. To generate the volume mesh a program called NetGen[1] was used. NetGen provides surface and volume meshing of various inputs as well as visualisation although in this case the separate programming interface was used which allows directly

[1]http://www.hpfem.jku.at/netgen/

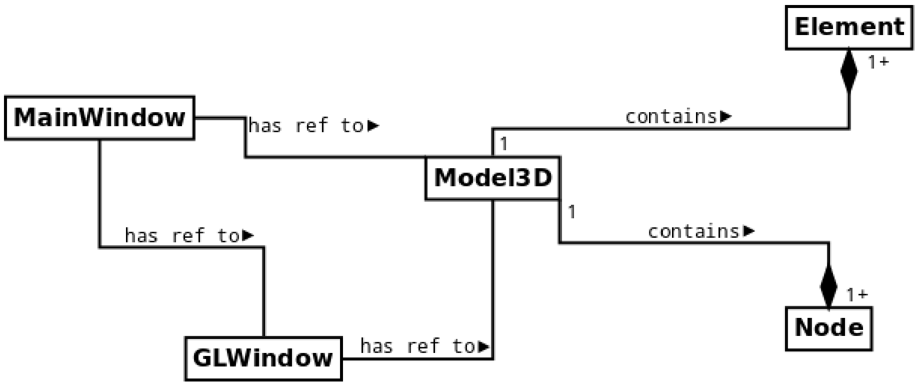


Figure 4.1: Basic object relationship.

adding vertices and faces to the mesh. Although released as a C++ library its design is very C-like with no object-oriented aspects. The library was not deeply integrated into the project; once the volume mesh was generated there was no need for it. The IML++ library[2] was used for carrying out the conjugate gradient method. This library has templated methods that require certain operators and methods to be implemented on the input data types. The previously mentioned matrix class was developed to work with this. Also, a wrapper to the STL vector container was created to implement similar requirements. Below is the prototype for the conjugate gradient method in IML++:

```
template < class Matrix, class Vector, class Real >
int CG(const Matrix &A, Vector &x, const Vector &b,
       int &max_iter, Real &tol)
```

The method was edited to remove the preconditioning code as it was not required for this project. It was also wrapped in a namespace so calls in the project could be easily identified.

Here we'll describe how the different matrices were calculated and stored. For each element stiffness and plasticity matrices were calculated and stored before the simulation was run as they only needed to be calculated once. Functionality was separated between the Element and Model3D classes depending on whether element-specific or global attributes were affected. Various methods were implemented to calculate and set initial values such as nodal masses and plastic strains.

[2]http://math.nist.gov/iml++/

The element and global stiffness matrices were constructed using a nested for loop. Each column and row correspond to a single node; in the case of the element matrix the dimensions are (3x4)x(3x4) because x, y and z are stored separately. This means the global matrix has dimensions 3nx3n where n is the number of nodes. The global matrix is assembled from the element matrices by summing values where nodes overlap. It's a simple process of looping over each node in each element and adding the value from the element matrix to the global matrix (by first obtaining the relevant global index of the node). Some efficiency can be gained because of the global matrix's symmetry, only half the values have to be calculated.

Basic plastic deformation was implemented although it was based on a linear elastic model so would not be entirely stable for large deformations. The following process was used to include effects of creep: Here we can see

Algorithm 4.1 Calculating plastic strains and forces

```
for all elements do
    $e = B(Re^{-1}x - x_u)$
    $e_{elastic} = e - e_{plastic}$
    if $|e_{elastic}|_2 > yield$ then
        $e_{plastic} += dt * creep * e_{elastic}$
    end if
    if $|e_{plastic}|_2 > max$ then
        $e_{plastic} *= max / |e_{plastic}|_2$
    end if
end for
$f_{plastic} = RPe_{plastic}$
```

that only plastic strains are stored, elastic strains are calculated at each frame as the difference between total and plastic strains. Also, plastic forces use the stiffness warping method by multiplying by the orientation matrix.

Nodal (or offset) forces were also calculated and stored in each node. Whereas the elastic forces are equal to the stiffness matrix multiplied by the current node positions, the offset forces are derived from the undeformed positions of the nodes. They are the opposite reaction required to return the system to equilibrium thus they are subtracted from the elastic forces when constructing the linear system.

To calculate the orientation matrix of an element, a matrix of the current deformed position was created (similar structure to the shape matrix). This was multiplied by the inverse of the undeformed shape matrix and then divided by $6xvol$. Gram-Schmidt orthonormalisation was then used to extract the 3x3 matrix from the previous result.

$$u_k = v_k - \sum_{j=1}^{k-1} proj_{uj}(v_k), \tag{4.1a}$$

$$e_k = \frac{u_k}{\|u_k\|} \tag{4.1b}$$

, where e_k is the normalised result, and v_k is the k'th column of the input matrix. The function $proj_u(v)$ is defined as:

$$proj_u(v) = \frac{\langle v, u\rangle}{\langle u, u\rangle} u \tag{4.2}$$

Qt Creator was used to design a GUI. It was made to give the user the ability to load a surface mesh, generate a volume mesh, apply any material properties and forces, and then run a simulation with some form of visualisation. It gives an option to save the simulated data to an output file that can be used for a high quality render. Initially a few widgets were added but as the feature set grew it became important to provide an intuitive and organised layout. A `QToolBox` widget was used as it allows maximum use of space while still being simple to use; it works like an accordian layout with ony one box of widgets available at a time.

An important part of the interface was to allow the user to interactively set forces at nodes on the model although producing real-time conrol akin to a modelling application was not the aim. Implementing a vertex selection procedure with the mouse was achieved by overloading the virtual method `mousePressEvent` from `QWidget` and then inverse projecting the mouse location. Given the position of the cursor on the screen, it can be converted to normalised screen space and then inverse projected back into world space by multiplying by the inverse of MVP (Model View Projection). Because the model position is used to rotate and translate the scene it was used along with the view and projection matrices in the calculation. This selection method does not discriminate by depth as an infinite ray is generated so an intersection with multiple points can occur. To make it easier to select vertices an imaginary sphere was drawn around each one and a sphere-ray intersection test was carried out. Although a space-partitioning scheme could have been used to make the selection more efficient for large numbers of vertices it was not a necessary feature as selection would not occur at the same time as running the simulation.

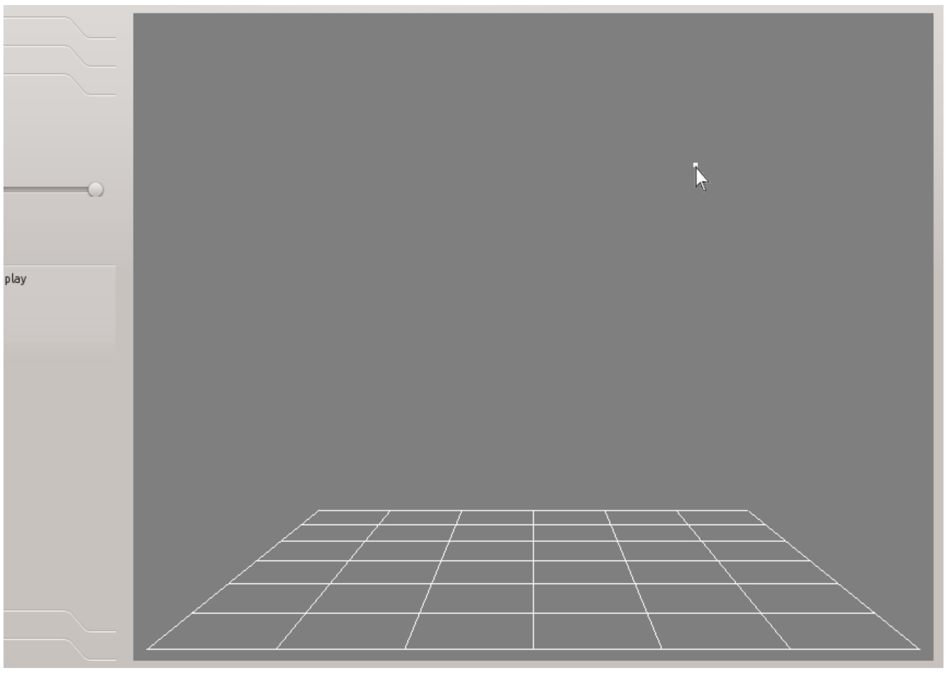


Figure 4.2: Inverse projection.

Per time-step the following calculations and updates had to be made:

- Calculate element orientation
- Calculate element strains
- Assemble global warped stiffnes matrix
- Calculate nodal force offsets
- Calculate plastic forces
- Assemble matrix A and vector B, solve $Av(t+1) = b$ using CG method
- Use itegration method to calculate new point positions
- If required, write out data to file

This set of operations was put into a class called Solver that was instantiated from MainWindow when the run simulation button was clicked. To ensure the GUI did not become unresponsive the method in the Solver class was run on a separate thread using Boost to create and destroy it. The Qt signals and slots system was used to notify MainWindow when frames were completed so the user could be informed and data could be written to file. Both Wavefront obj and RenderMan rib file writers were written. Although NGL provides an rib writer it was felt that more control was needed over the contents of the file. Writing the obj writer was a trivial task so it will not be explained here. One thing to note, though, was that the list of vertices held in Model3D included those inside the model which are redundant in a surface mesh. We should also point out that a frame rate control was included so the user could create time-based effects similar to a high-frame rate camera.

An important aspect of creating a tool for artists is to allow dynamic control of variables. A simple keyframing system was implemented which allowed start and end forces to be entered, and a start and time for when the forces should be applied. Linear interpolation was implemented like so:

$$y = y_0 + (\frac{y_1 - y_0}{x_1 - x_0})(x - x_0), \tag{4.3}$$

where y_0, y, y_1 are the starting, current and ending forces respectively; x_0, x, x_1 are the starting, current and ending times respectively. Although this is an unsophisticated approach, anything more advanced would be reinventing what many modern 3D applications can already achieve.

4.2 Visualisation

A large variety of data can be generated in this project which means a lot of different information can be shown to the user. Shaders were written in GLSL to provide different views of the object. Initially a basic colour shader was created to visualise the tetrahedra. Later a geometry shader was created to allow for on-the-fly resizing of the elements using a control in the GUI. This required the tetrahedron centres to be passed with the vertices. The scaling was achieved by simple displacement of each point along the line from its origin to the centre point.

```
gl_Position = gl_in[0].gl_Position
              - scale * (gl_in[0].gl_Position - centre[0]);
gl_PrimitiveID = gl_PrimitiveIDIn/4;
```

The second line is for random colouring of the tetrahedra to better identify them. A 1D texture was created on the CPU with 128 random colours and sent to the GPU. Using the `gl_PrimitiveID` value, a colour could be chosen from the texture in the fragment shader. Another shader was also created to show just the surface mesh as this is what the final renders would look like. Basic colour shaders were also used for the grid and vertices.

To display the nodal forces and element strains at each time-step the magnitude of the total force at each node was stored, then the maximum was found so a scaling factor could be calculated. For each node the force was inverted and scaled to the range 0 to $^2/_3$. This value was then used as the hue for the generated colour thus producing a range of colours from blue (no forces) through green to red (maximum forces). We should point out that the visualisation is relative so a red area would show the maximum force across the whole model but it might not necessarily by an absolute high value. The colours were stored in the node and sent to the GPU in the `updateVAO` method. The same set of steps was used to calculate the colours for element strains and the colour was stored in each element.

Basic 3D camera tracking and panning was implemented by altering values in the model matrix. It was based on code from the NGL demos. Because the program only allowed one model to be loaded more advanced controls were not required. When exporting rib files the model view matrix was used to translate the whole scene, this way the user could compose a shot in the program before rendering.

Algorithm 4.2 Procedure for calculating vertex normals

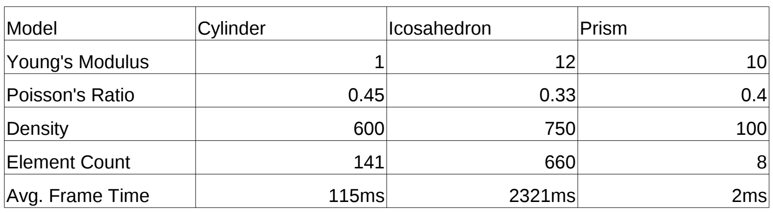
```
for all i in nodes do
    vec3 normal
    int adjFaces
    for all j in surfaceIndices do
        if surfaceIndices[j] == i then
            normal += faceNormals[j/3]
            adjFaces++
        end if
    end for
    normal /= adjFaces
    normalise(normal)
end for
```

`VertexArrayObject`s from NGL were used to manage the OpenGL buffers and attribute data. These were stored in the Model3D class. Two were used for the volume mesh and surface mesh and another for the points. The `updateVAO` method sets up data for drawing by passing the vertex positions and any required attributes to OpenGL. For the surface mesh, vertices were sent with normals using indices to construct the triangles more efficiently. The normals are actually first calculated for each face and then averaged at each vertex. A slightly obscure method was chosen to avoid having to store a face data type. Algorithm 4.2 outlines the process. Note that `surfaceIndices` is 3 times the size of `faceNormals` because each face is a triangle.

5

Results and Analysis

A variety of primitive models were loaded into the program. Unfortunately the NetGen library was strict with how polygons are described so not every model could be meshed. What follows is an explanation of the material properties and images of the resulting deformation. Table 5.1 above shows



Model	Cylinder	Icosahedron	Prism
Young's Modulus	1	12	10
Poisson's Ratio	0.45	0.33	0.4
Density	600	750	100
Element Count	141	660	8
Avg. Frame Time	115ms	2321ms	2ms

Figure 5.1: Properties for sample models.

properties of three models that were used to test the program. The simulations were run on a quad-core CPU clocked at 2.8GHz. From the data it can be seen that computation time per frame increases with the number of elements/nodes; this was to be expected.

Figure 5.3 shows a cylinder being squashed by a force from above. The ottom nodes were fixed. As the force was applied the model started to distort as the tetrahedra shifted and resized. Figure 5.2 shows a similar deformation but of a sphere. Again forces were applied to the top nodes, downwards, and the bottom nodes were fixed. This created a very strange-looking effect as the sides of the sphere dipped lower than the fixed area, like a soft plastic ball would. Figure 5.3 shows a very simple prism consisting of only 8 elements. As such the calculations were well within the time required for real-time visualisation. The prism was pulled upwards but because of the low number of elements one node went much higher than the others.

These tests highlight that it is important to know the properties of the material that is to be simulated otherwise unnatural or non-existent materials

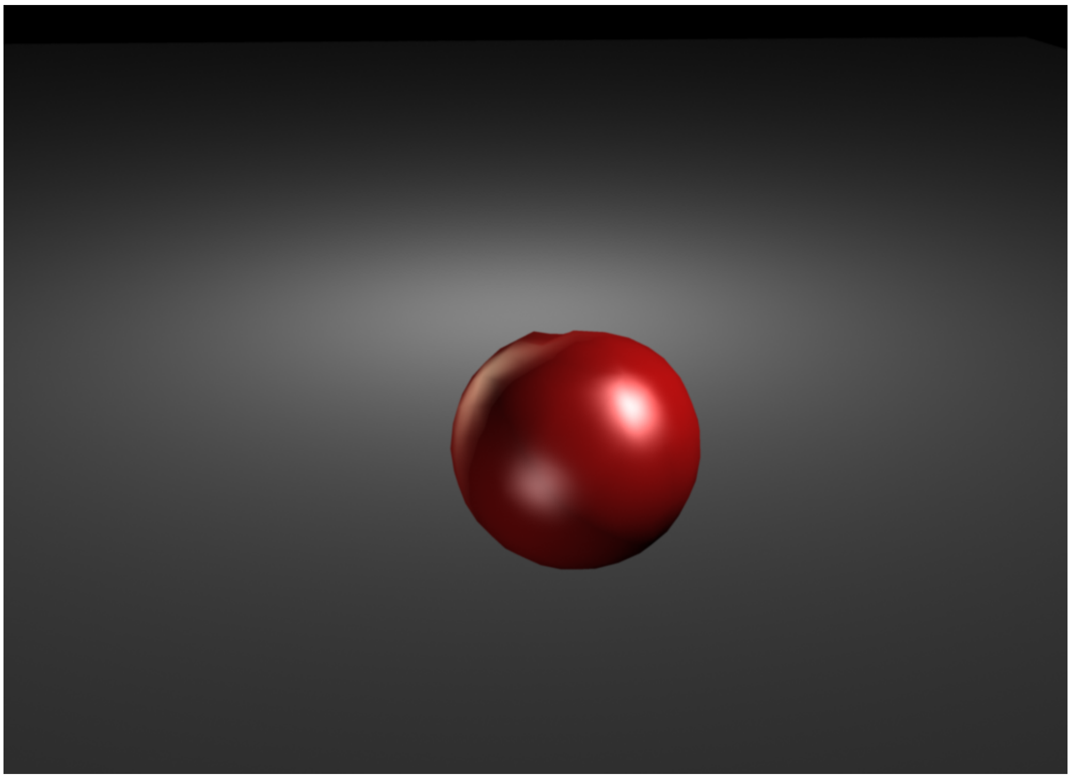


Figure 5.2: The icosahedron with forces applied from above.

could be simulated. For an artist, looking up material properties is not something they want to be doing so creating a simpler set of controls to changes attributes such as softness or strength would allow easier use.

Included also with this paper are two rendered simulations of an icosahedron and a cylinder. They were created by writing out an obj file per frame and then loading the sequence into Autodesk Maya 2011. Each video shows the same deformation at different recorded frame rates to better show the effect.

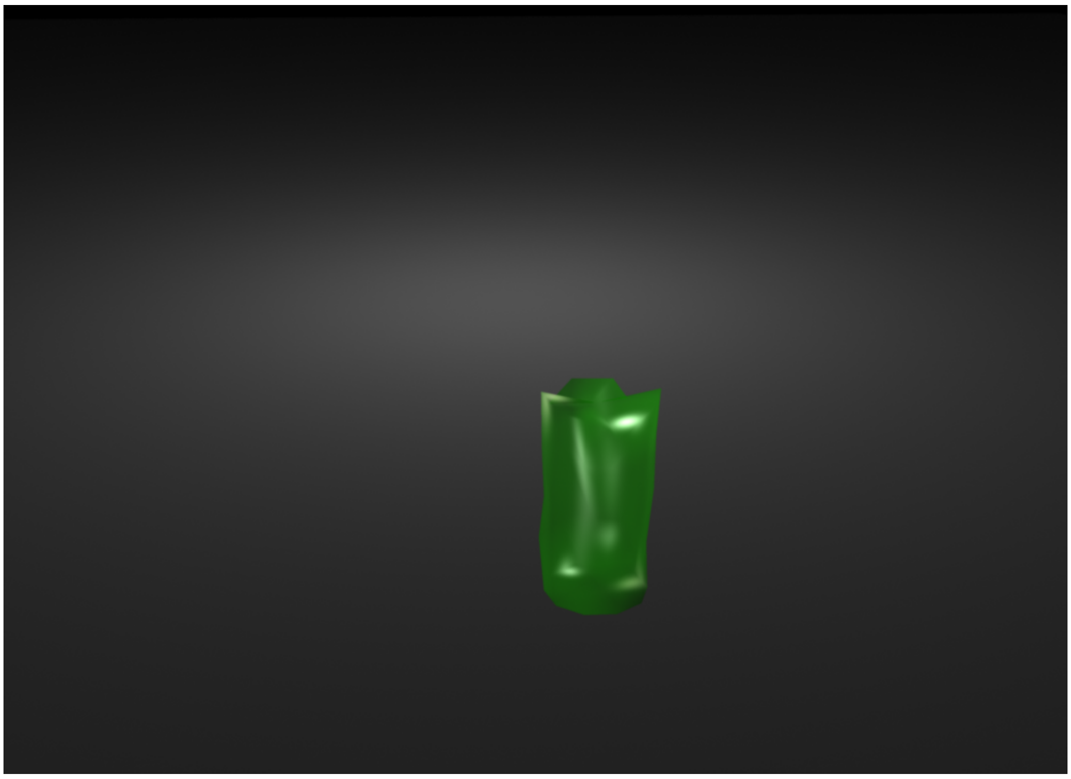


Figure 5.3: The cylinder being squashed from forces above.

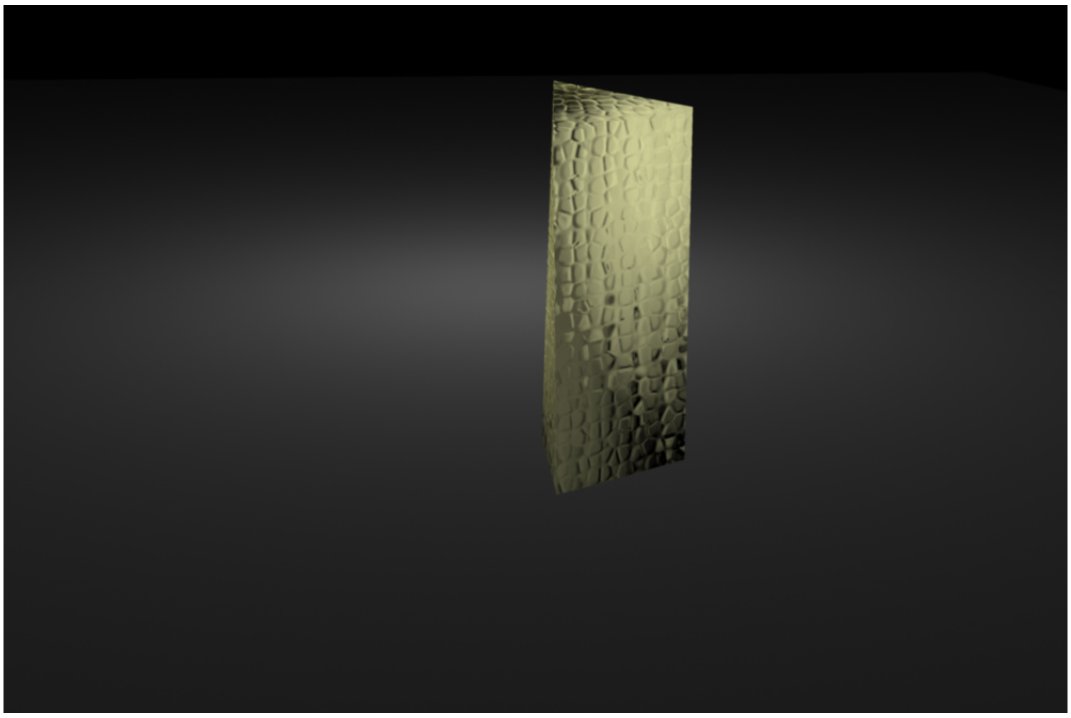


Figure 5.4: The prism being stretched upwards.

6

Conclusions

Improvements could be made to the interface so artists are more comfortable using it (the keyframing system being a good example) but there's a point where it's better to make a plugin for an existing application such as Autodesk Maya or SideFX Houdini. That way the powerful controls offered by the software can be utilised to create a more interesting program. However, both Maya and Houdini already have plugins available that achieve better results so it would only be worthwhile as an academic exercise.

Porting the program to a plugin would allow better incorporation into the 3D pipeline allowing more efficient use in a real production. Also, integration with a RBD system with dynamically created external forces would create a very usable system for creating deformation effects. Only linear elastic deformation was simulated in this project. Although it was an objective to progress to non-linear plasticity, too much time was spent on implementing the basic system. Plastic forces were included but large deformations led to stability problems that affected the realisticness. Crack generation and fracturing would also be a good next step for this project so results similar to those in previous research could be achieved.

A future path that has not been mentioned is a GPU implementation using CUDA or OpenCL. The advantages of parallelising this system would make a very fast and probably real-time program. By separating the global matrix up and solving in parallel, massive performance gains could be achieved that would greatly increase productivity in a live production environment.

Appendix A

Sample Models

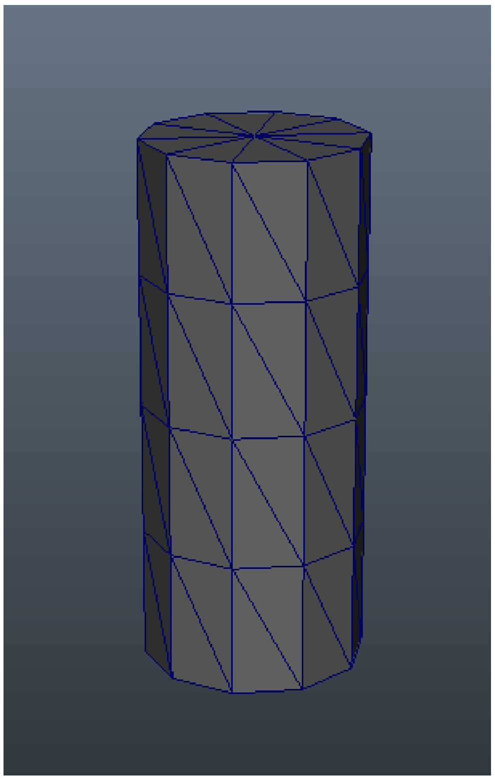


Figure A.1: Surface mesh of the cylinder model.

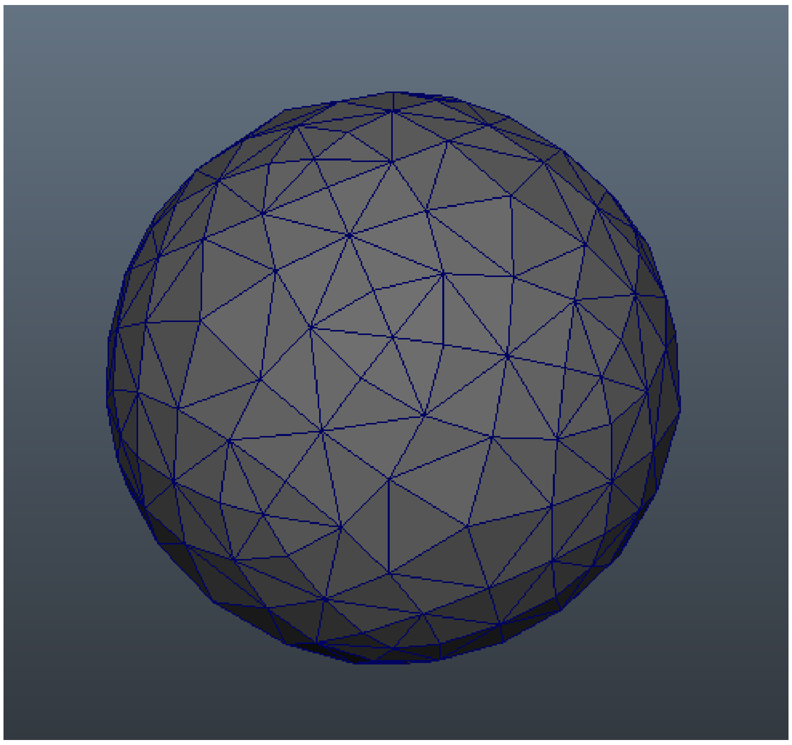


Figure A.2: Surface mesh of the icosahedron model.

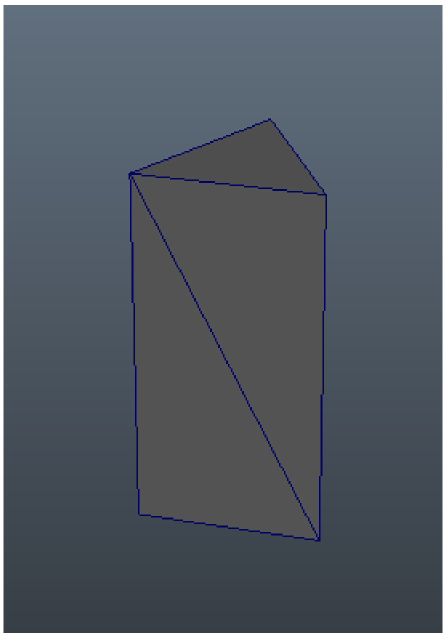


Figure A.3: Surface mesh of the prism model.

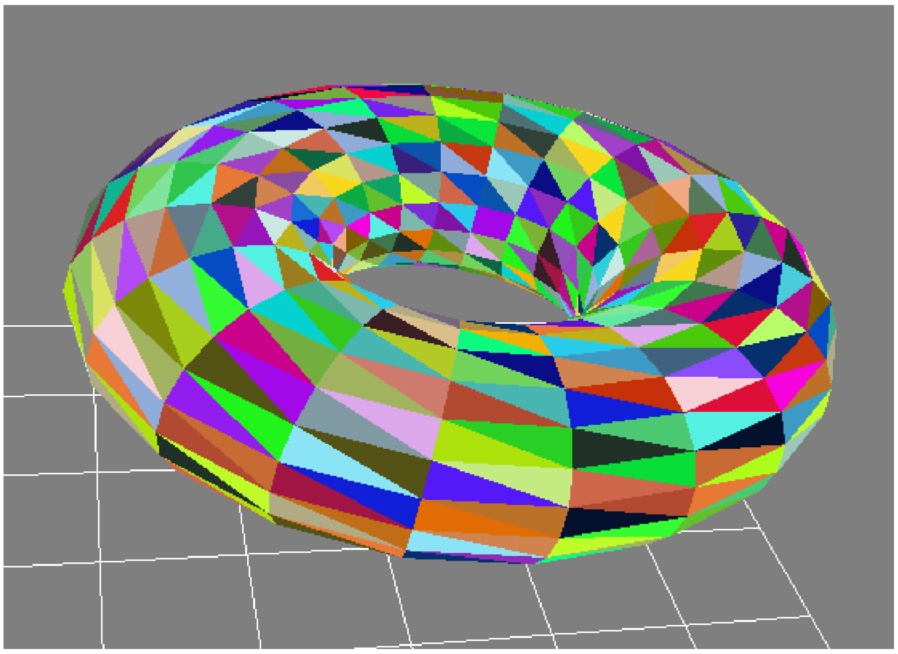


Figure A.4: Surface mesh of the torus model with the volume shader.

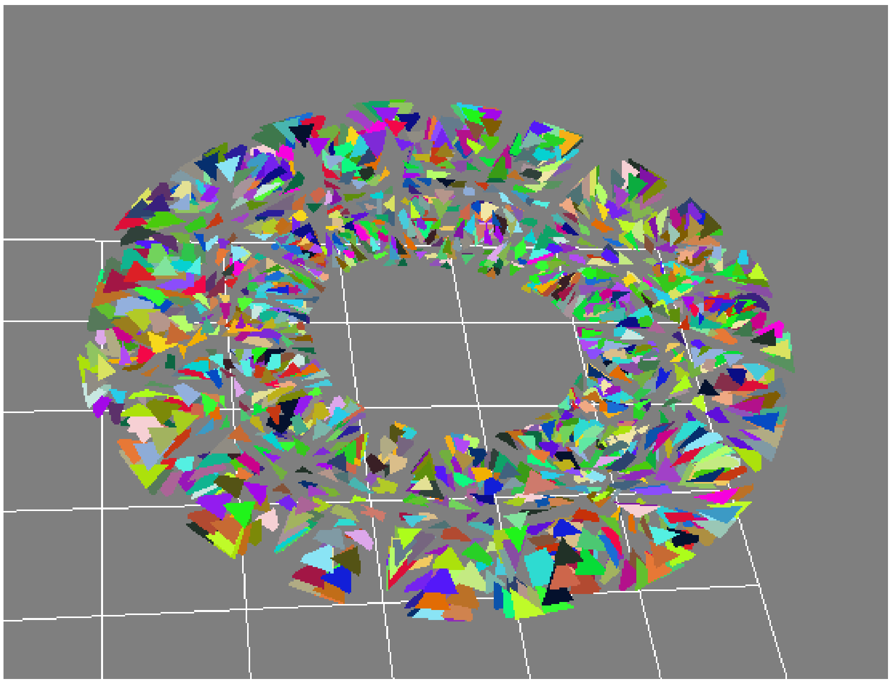


Figure A.5: Surface mesh of the torus model with the volume shader, scaled to 0.25.

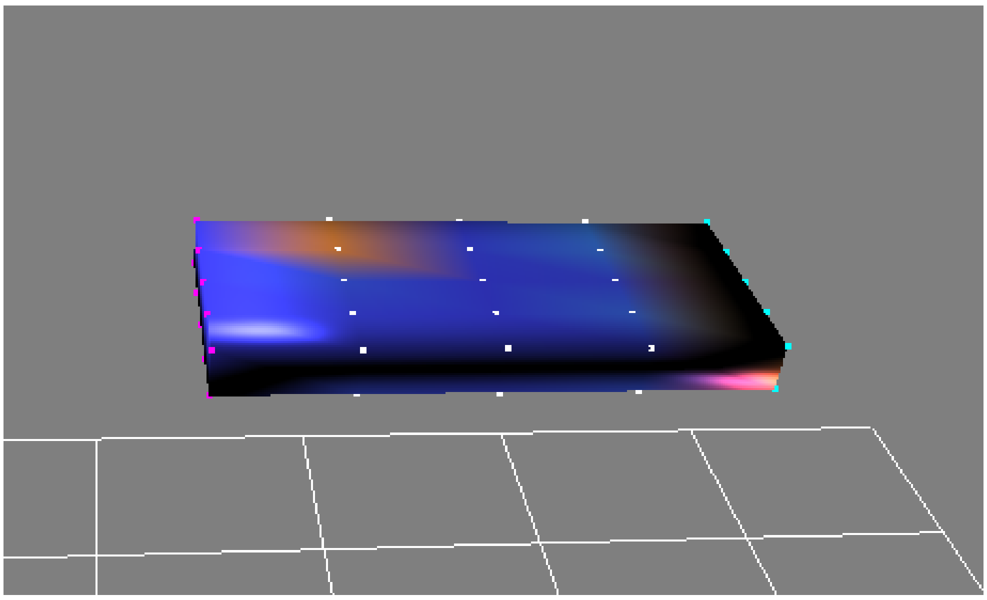


Figure A.6: Surface mesh of the bar model with nodal forces shown on the surface shader.

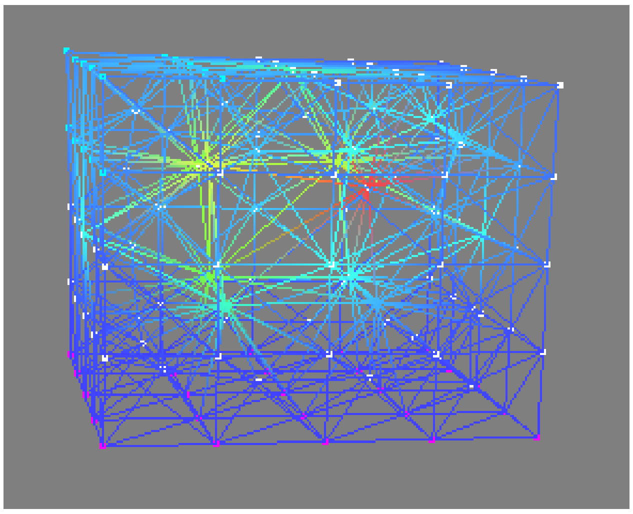


Figure A.7: Surface mesh of the cube model with nodal forces shown on the volume shader.

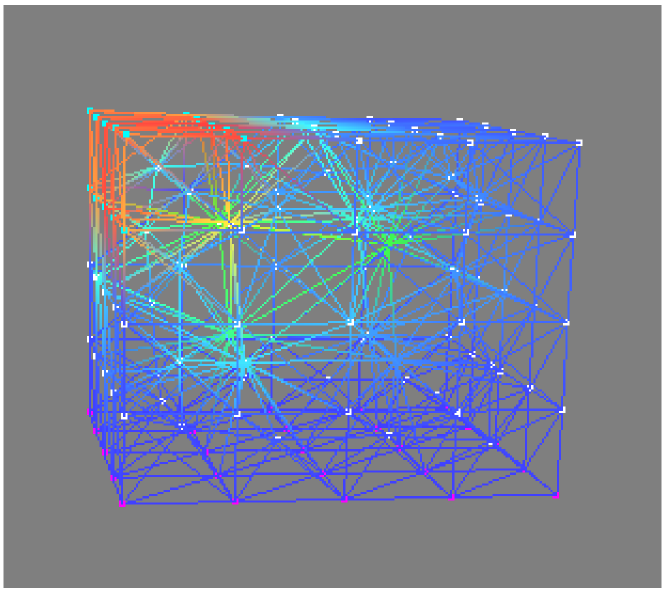


Figure A.8: Surface mesh of the cube model with nodal forces shown on the volume shader.

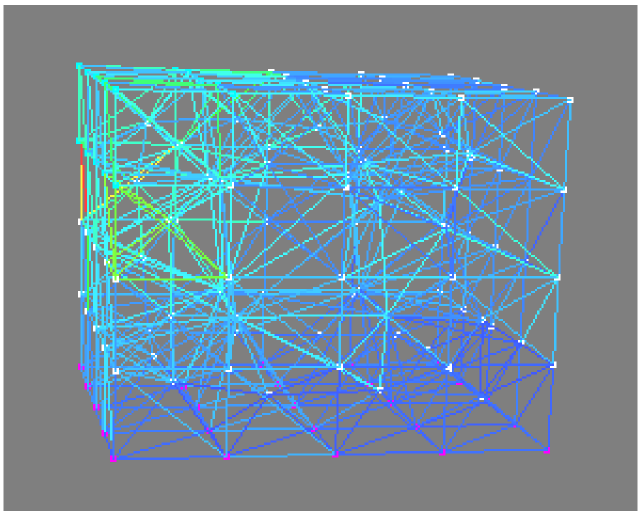


Figure A.9: Surface mesh of the cube model with element strains shown on the volume shader.

Appendix B

Diagrams

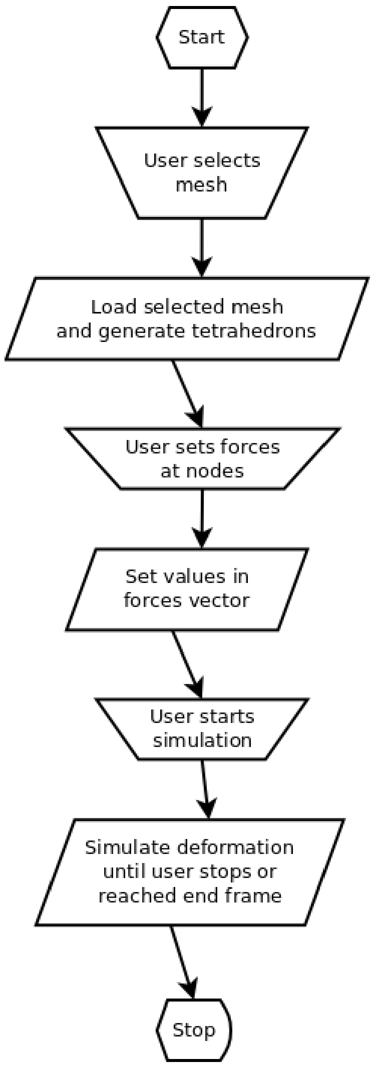

Start
User selects mesh
Load selected mesh and generate tetrahedrons
User sets forces at nodes
Set values in forces vector
User starts simulation
Simulate deformation until user stops or reached end frame
Stop

Figure B.1: Typical user workflow.

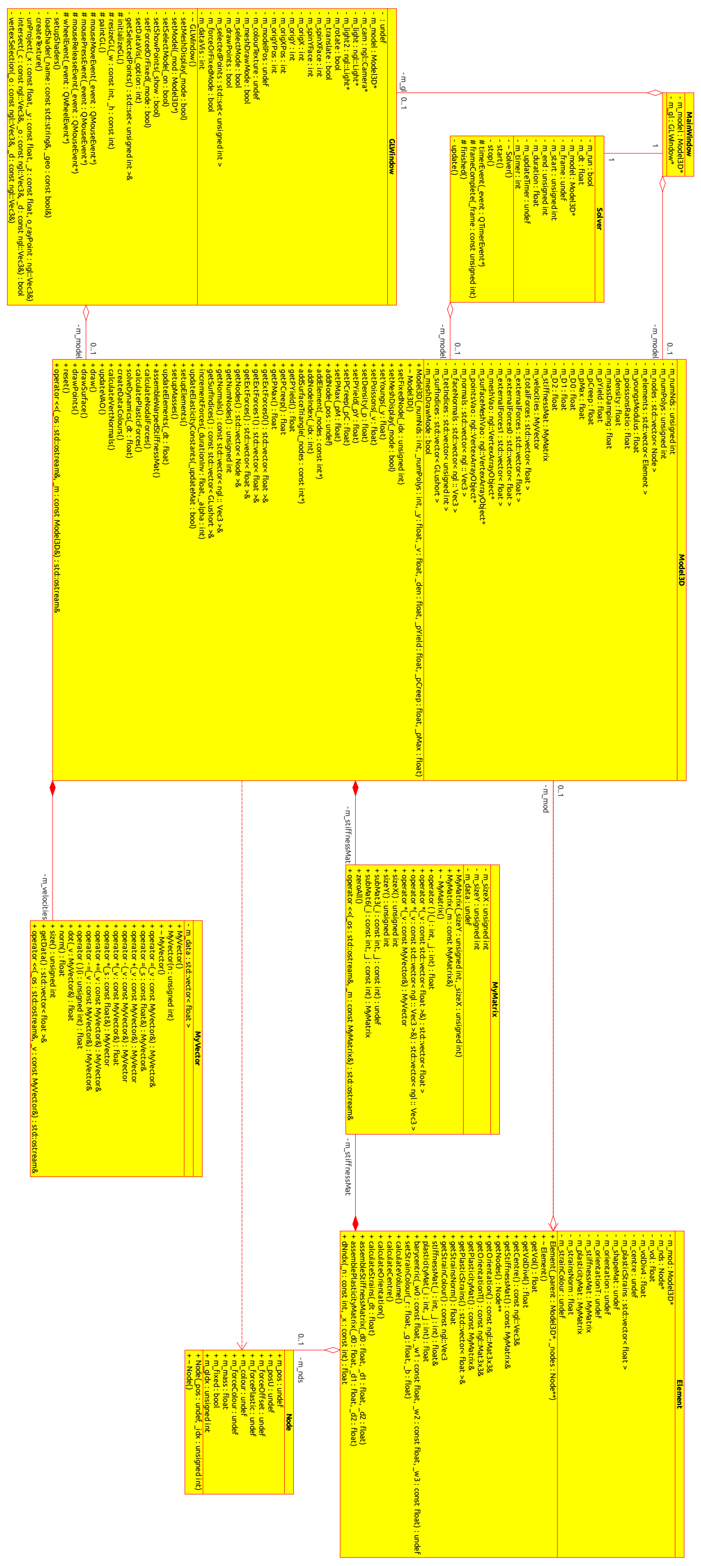


Figure B.2: Class diagram of final code.

Bibliography

Bargteil, A. W., Wojtan, C., Hodgins, J. K. and Turk, G., 2007. A finite element method for animating large viscoplastic flow. *In: ACM SIGGRAPH papers*, 2007. San Diego, CA. New York, NY: ACM.

Erleben, K., Sporring, J., Henriksen, K. and Dohlmann, H., 2005. *Physics-Based Animation.* Hingham, MA, USA: Charles River Media.

Gould, P. L., 1994. *Introduction to Linear Elasticity.* New York, NY: Springer-Verlag 2nd ed.

Henwood, D. and Bonet, J., 1996. *Finite Elements A Gentle Introduction.* London: Macmillan Press.

Hilf, B., 1997. Don't believe your eyes: It is real or is it animation. *Animation World Magazine*, 2(5), 17–19.

MacDonald, B. J., 2011. *Practical Stress Analyis with Finite Elements.* Dublin, Ireland: Glasnevin 2nd ed.

Müller, M. and Gross, M., 2004. Interactive virtual materials. *In: Proceedings of Graphics Interface*, May 2004. London, Ontario, Canada. Waterloo, Ontario, Canada: Canadian Human-Computer Communications Society School of Computer Science, 239–246.

Müller, M., Julie, D., McMillan, L., Jagno, R. and Cutler, B., 2002. Stable real-time deformations. *In: Proceedings of ACM SIGGRAPH*, July 2002. San Antonio, TX. New York, NY: ACM, 49–54.

O'Brien, J. F. and Hodgins, J. K., 1999. Graphical modeling and animation of brittle fracture. *In: Proceedings of ACM SIGGRAPH*, Aug 1999. New York, NY: ACM Press/Addison-Wesley Publishing Co., 137–146.

Parker, E. G. and O'Brien, J. F., 2009. Real-time deformation and fracture in a game environment. *In: Proceedings of ACM SIGGRAPH*, 2009. New Orleans, LA. New York, NY: ACM, 165–175.

Saad, Y., 2003. *Iterative Methods for Sparse Linear Systems.* Philadelphia, PA: Society for Industrial and Applied Mathematics 2nd ed.

Terzopoulos, D., Platt, J., Barr, A. and Fleischer, K., 1987. Elastically deformable models. *In: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, 1987. New York, NY: ACM, 205–214.

Wicke, M., Botsch, M. and Gross, M., 2007. A finite element method on convex polyhedra. *Computer Graphics Forum*, 26(3), 355–364.

Zhang, Y., Hughes, T. J. R. and Bajaj, C. L., 2010. An automatic 3d mesh generation method for domains with multiple materials. *Computational Geometry and Analysis*, 199(5-8), 405–415.