Master thesis

Procedural deformation and destruction in real-time

Christofer Stegmayr

Advisors:
Torbjörn Söderman   Joakim Lord

Examinor: Kenneth Bodin

July 17, 2008

Abstract

Special effects are almost exclusively done on computers today. Explosions, fires and destruction are most popular in action movies. These effects usually take several days or months to simulate on a computer in order to get the desired result. A field that is fairly new, in special effects, is solid mechanics. In other words, how materials deform and fracture. This is also a wanted effect in computer games. The aim of this thesis is to simulate solid mechanics in real-time (> 15 Hz). Several methods is evaluated and a primary method is implemented. The primary method implemented is the Element Free Galerkin (EFG) method. This is a meshless method which uses moving least square (MLS) approximations to calculate gradient of the deformation, which is used to approximate the strain of the material. The simulations, in this thesis, is parallelised and threaded in order to get better performance.

The EFG method yiels interactive framerates (30-150 Hz) depending on the resolution of the simulation. Because of limitations of the simulated speed of sound, only materials with a Young's modulus lower that 10^8 could be simulated using this method and the desired time step ($0.01 < \Delta t < 0.1$). This resulted in simulations of materials much like jelly. Real-time simulations of deformation and fractures are possible and easily parallelisable. Paralellisability is an advantage for GPU-simulations and future processor models. The problems with the low Young's modulus could possibly be solved using a for example constraint based regularisation of the simulation, to allow the simulated speed of sound to in theory approach infinity.

Contents

Chapter 1

Introduction

The field of computer animation has revolutionised the special effects industry for several decades back. Animators today use sophisticated physics simulations to enhance the reality of their work. The same techniques are now finding their way into computer games. The only restriction here is that the algorithms and methods need to be optimised and the level of detail brought down. This is because the game needs to run in real-time[1], not as in the film industry where a complex special effect can take days or even months to compute.

Many people are fascinated by special effects that involve large explosions and destruction. So much research has been done in this area. Explosions can now be simulated visually by a combination of particle systems that are affected by a heap of physical forces. The results are quite remarkable. But the destruction that the explosions cause are still often modelled without any real physical background. For a human eye this is quite visible since we have a connate sense of how the real world would look like, like a sixth sense of physical correctness[24, 37]. These flaws are often masked, with varying result, by for example adding even more smoke and fire to the explosions. What is needed to take this area further is a simulation of the fracturing and destruction using true physical properties.

These simulations are already done[4], in some sense, for the film industry and it is becoming the favoured way of creating these special effects. But it still has some limitations, which will be considered later in this thesis. For the computer games industry, the fact that these simulations does not have a well documented history of running in real-time is a problem.

1.1 Aim

The aim of the thesis is to implement a simulation of deformable and fracturable objects. The implementation should be done from the concept that it could be incorporated into a game. This means that the implementation has to run in real-time and the visual result is important. In the study different methods

[1]Real-time is defined as 15 frames per second (FPS), but a preferable framerate is the refresh rate of the display (often 60-85 Hz).

should be considered, including both CPU and GPU implementations. The models are evaluated and compared to each other. One primary method should be chosen and implemented in order to see if the method is indeed suitable for a game. The project is divided into two parts, one simulation part and one visualisation part. This thesis will focus on the simulation part.

Chapter 2

Background

2.1 Physical background

The area of *continuum mechanics* is one of the more complex branches of physics. Although most basic equations are formulated, these are often impossible to solve analytically. This may be the case with several field of physics, but the equations of continuum mechanics are mostly non-linear. This is why finding solutions might be difficult, even for special cases of symmetry. But with the introduction of computers many of these problems could be studied in new ways.

The key ideas of continuum mechanics are *conservation of momentum*, *conservation of mass* and of course *conservation of energy.* Continuum mechanics have two major branches which also must fulfill these "laws". These are *fluid mechanics* and *solid mechanics. Fluid mechanics* is the branch which studies the dynamics of fluids and gases, e.g. how fluids are affected by boundary conditions, external forces and temperature. *Solid mechanics* is the branch which studies the dynamics of solid matter, e.g. how solids are effected under force, temperature change and applied displacement. In this thesis solid mechanics will be considered, but sometimes similarities to fluid mechanics will be pointed out. The theory will be explained in more detail in Chapter 3 on p. 7.

2.2 Numerical background

Solving differential equations analytically is only possible for a small number of differential equations. These are often special cases of symmetry or simple linear differential equations. For non-linear equations chaotic behaviour is often experienced and there are no analytic solutions to those problems. These equations needs to be solved numerically instead.

Numerical integration has a long history. Ranging back several thousand years, where integrals of elementary functions could not be computed analytically but their derivatives could[50]. Integrals were calculated using geometric formulas where the slope was approximated by the derivative of the function (e.g. the midpoint method and the trapezoid method)[50]. More sophisticated methods

were later derived, e.g. Simpson's method and Romberg's method[50].
These methods were then evolved to handle approximation of differential equations. One of the simples forms of integrating differential equations is the explicit Euler method, which is unconditionally unstable[28]. In order to increase accuracy, higher order methods were invented (e.g. Runge-Kutta method, Bulirsch-Stoer and Richardson's extrapolation)[50]. Explicit higher order methods are usually not stable for large time steps or very many time steps[28], although these methods show a much higher accuracy within a reasonable amount of time steps than for example a lower order implicit method. As for implicit methods, these are unconditionally stable but they also damp out the solution in time.
The best approach, for long time simulations, seem to be the variational methods. Exampel of variational methods are the Verlet method[54] and the symplectic Euler[28].

What all the above methods have in common, are that they solve ordinary differential equations (ODE) in space-time. For partial differential equations (PDE) these schemes need to be refined. One way of doing this is to generalise the PDE to a system of ODEs and solve these using the proposed method above. This became more complex when the PDE was non-linear, the methods were also unable to handle complex geometries in an efficient way. This has been solved in some sense today[33] (using e.g. irregular grids) but there are still problems, which lead way to a new kind of numerical method. This method was presented 1950, based on theory of the mathematician Richard Courant[10] but also from the work of Boris Galerkin[16] and Walter Ritz[51]. The method presented was intended to solve the PDEs with complex geometry based on the non-linear equations of solid mechanics. The simulation in question was the stress and strain on a airplane wing[13, 14, 27, 29, 30], using methods which later were to be called the Finite Element Methods (FEM). More on the history of FEM can be read in the review article by Samuelsson and Zienkiewicz[52].

In modern computer simulation of solid mechanics, FEM is still used. Mostly in offline (non real-time) simulations, for both real physical simulations[34] and special effects[4, 46]. But there are even examples where it has been used in real-time simulations[42]. There are also advantages using FEM when simulating fractures[40], since the tough work of discretizing the volume is already done. But the FE-method have some disadvantages and new methods were developed to counter these "flaws". One of the "flaws" that FEM has is that it cannot handle large deformations very well. For those cases more dynamic mesh-free methods (or meshless) were developed and are becoming increasingly popular.

Using a meshless method, volumetric discretization is relatively simple. One of the earlier meshless methods was the smoothed particle hydrodynamics method (SPH). The idea of SPH came from astro-physical simulations back in 1977[17], where star dynamics could be modelled as a special case of fluid dynamics. The method was later generalised to work with any fluid dynamics problem, mainly using Navier-Stokes formulation of the fluid dynamics[38]. Today this method can be used to solve many more problems than fluid dynamics, one example is solid mechanics.
The SPH method is a so called meshless method in contrast to a mesh based

method. A mesh based method uses a grid to sample the space in some way. The grid does not have to be regularly spaced, but this simplifies things. A mesh method is most effective if there is some innate symmetry in the problem. If the problem is very asymmetric a meshless method may be more effective. A meshless method does not discretize space in the same way as a mesh method. Meshless methods often uses "particles" to sample the domain. These particles are free to move around in space and interact with other particles in the simulation. Physical quantities are calculated as weighted averages of the neighbouring particles.
SPH is currently used to simulated water and other fluids in real-time with moderate accuracy (and offline[1] with good accuracy). The method is quite parallel so it is suitable to be implemented on modern graphics cards. Real-time implementations on the graphics card have been able to simulate 4096 particles at 183 FPS[2] while the CPU implementation could simulate 4096 particles at 23 FPS[21]. Harada et. al[21] also showed that they can simulate up to 65536 particles in real-time (17 FPS).

SPH can also be used to simulate solid mechanics, but a more popular method is the Element Free Galerkin (EFG) method. It has many similarities to SPH but it evaluates derivatives (gradients) differently. Both SPH and EFG are meshless methods, which means that the vector field for the solution is in fact continuous. In ordinary FDM methods the functions are evaluated at discrete steps, in FEM the continuous function is approximated by linear interpolation between discrete steps. But for meshless methods the functions are in fact continuous (and must be treated that way). Li and Liu[31] goes through many of the differences of the meshless methods, as well as pros and cons for each method. Belytschko et. al[6] shows deeper mathematical differences and similarities of the meshless methods. It was in fact Belytschko et. al[7] who first used moving least square approximations to simulate solid mechanics and in that way developed the EFG method. Another advantage with the mesh-free methods is that the simulation geometry is not bound to a graphical mesh (but rather the other way around), and large deformations are therefore possible. Also, up-sampling and down-sampling of the simulation can be done in runtime relativly easily and effectively[48]. Although special care must be taken when resampling because simulations may become unstable.

There are many other types of messless simulations of continuum mechanics. There is a good overview of these in the review article by Li and Liu[31].

[1]Offline is when the simulation is not done in real-time
[2]Frames Per Second

Chapter 3

Theory

3.1 Physics

A brief overview of the physics of solid mechanics is covered in this section. Consider a one dimensional elastic rod, it may be modelled as a one dimensional massless spring. Most people that read some physics are well aware of *Hook's law* for massless springs:

$$f = -kx \quad \text{or} \quad U = \frac{1}{2}kx^2 \tag{3.1}$$

where f is the force that the spring exerts and U is the corresponding potential energy. This law is a good approximation of a spring in one dimension, and it can also be generalised to three dimensions by using vector notations. The only real flaw is that the forces in three dimesions are still in the direction of the spring, i.e. no shearing forces.

A simple model for a elastic solid material would therefore be a system of massless springs that are connected together. This may actually work, although it is tedious to do any calculations on a system like that since one spring is effected by all other springs in the system in a non-trivial way. This would lead to a system of equations the size of the number of springs. In a computer this can be solved "rather" effectively, but for computer calculations there are other issues, like numerical stability, that need to be considered. So in the age before computers a theory call solid mechanics was developed. In this theory the foundation was Hook's law, but instead of approximating the material as many springs, a tensor was used to represent the material. This tensor had as many dimesions as the domain of the simulation, i.e. for three dimensional simulations the tensor was a 3×3 matrix.

3.1.1 Strain

Let,

$$\mathbf{p} = \mathbf{x} + \mathbf{u}, \tag{3.2}$$

where $\mathbf{p} = \mathbf{p}(\mathbf{x})$ is the actual position, $\mathbf{x}$ is the undeformed position and the vector field $\mathbf{u} = \mathbf{u}(\mathbf{x}) = [u(\mathbf{x}), v(\mathbf{x}), w(\mathbf{x})]^T$ is the *displacement* at position $\mathbf{x}$. To specify the constitutive response of an elastic solid, a measure of deformation is needed. The gradient, or derivative, of the deformation (because we don't want the elasticity to be dependent on where in the world the object is) would be a good measurement of this deformation. This measurement is called strain, ε. Also take the gradient of the transpose of the deformation, because the strain needs to be a symmetrical tensor (the first part is linear strain and the other part is shear strain). Cauchy's infinitesimal strain tensor is defines as

$$\varepsilon_{\mathbf{c}} = \frac{1}{2}(\nabla\mathbf{u} + \nabla\mathbf{u}^T), \tag{3.3}$$

where $\nabla\mathbf{u} = [\nabla u, \nabla v, \nabla w]^T$. The tensor in eq. (3.3) is linear with respect to $\mathbf{u}$, this is very nice for numerical calculations but can cause unwanted artifacts if the deformations are large[40]. Since pure rotation should not induce any stress in the body, one can use a rotation-independent tensor. As a rotation can be represented as an orthogonal matrix (i.e. $\mathbf{A}^T = \mathbf{A}^{-1}$ or $\mathbf{A}^T\mathbf{A} = \mathbf{I}$), a rotation mutiplied with its inverse should lead to no rotation. This can be taken into account when constructing the strain tensor. By multiplying $\nabla\mathbf{u}^T\nabla\mathbf{u}$, the rotation is excluded. The strain tensor then becomes:

$$\varepsilon = \frac{1}{2}(\nabla\mathbf{u} + \nabla\mathbf{u}^T + \nabla\mathbf{u}^T\nabla\mathbf{u}) \tag{3.4}$$

This tensor is the non-linear Green strain tensor (or the Lagrangean Strain tensor). It measures local variations in the displacement field in an fairly accurate way. The Green strain tensor can also be expressed as a function of the Jacobian of the displacement field $\mathbf{J} = \nabla\mathbf{u} + \mathbf{I}$. The Green strain tensor then becomes $\varepsilon = \mathbf{J}^T\mathbf{J} - \mathbf{I}$.

3.1.2 Stress and Energy

The strain tensor does not say much about internal forces of the body. To express the force Hook's law is again applied, but here in a more general form,

$$\sigma = \mathbf{C}\varepsilon, \tag{3.5}$$

where $\mathbf{C} = \mathbf{C}(E, \nu)$ is a symmetric, positive definite matrix (see eq. (3.14) on page 11) which is material dependant and σ is the stress tensor. The strain is a measure of force per unit area. Note the resemblance of eq. (3.1) and eq. (3.5), the latter is a generalisation of the first into three dimensional bodies which can also shear. The energy can also be expressed in a way similar to eq. (3.1):

$$W = \frac{1}{2}\mathbf{C}\varepsilon^T\varepsilon = \frac{1}{2}\mathbf{C}\varepsilon\sigma \tag{3.6}$$

Note that the energy in eq. (3.6) is an energy density, i.e. energy per unit volume. To get the entire potential energy for the body, the energy density needs to be multiplied with the volume of the body.

$$U = \frac{1}{2}V\varepsilon^T\mathbf{C}\varepsilon = \frac{1}{2}V\varepsilon\sigma \tag{3.7}$$

Remember that ε is symmetric, i.e. $\varepsilon = \varepsilon^T$.

3.1.3 Differential equations

A dynamic system can always be represented by a differential equation. The differential equation *is*, in some sense, the dynamics of the system. In the case of solid mechanics, or physics in general, the differential equation can be expressed as a function of the energy in the system. There are general methods for setting up these differential equations starting from the energy of the system. One of these methods are the Lagrange's equations, which can be derived from the variational principle (see Marion and Thornton[36]). Let $\mathcal{L} = T - V$, then:

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}_v} - \frac{\partial \mathcal{L}}{\partial q_v} = Q_e \tag{3.8}$$

where T is the kinetic energy, V is the potential energy, q_v are the generalised coordinates, Q_e is non-conservative applied forces expressed in generalised coordinates and $\mathcal{L}$ is known as the Lagrangian. Another popular method is Hamilton's equations (briefly formulated in Appendix A on page 41).

Using Lagrange formulations, the solid mechanics equations can be expressed as (use kinetic energy $T = \frac{1}{2}mv^2$ and potential energy U as in eq. (3.7)):

$$\begin{aligned}
&\mathcal{L} = T - U = \tfrac{1}{2}m\dot{\mathbf{u}}^2 - \tfrac{1}{2}V\varepsilon\mathbf{C}\varepsilon \\
&\tfrac{\partial \mathcal{L}}{\partial \mathbf{u}} \rightarrow \nabla_{\mathbf{u}}\mathcal{L} = \tfrac{1}{2}V\nabla_{\mathbf{u}}(\varepsilon\mathbf{C}\varepsilon) = V\mathbf{C}\varepsilon\nabla_{\mathbf{u}}\varepsilon \\
&\tfrac{\partial \mathcal{L}}{\partial \dot{\mathbf{u}}} = m\dot{\mathbf{u}}
\end{aligned} \tag{3.9}$$

and the differential equation then becomes:

$$\begin{aligned}
&\tfrac{d}{dt}\tfrac{\partial \mathcal{L}}{\partial \dot{\mathbf{u}}} - \tfrac{\partial \mathcal{L}}{\partial \mathbf{u}} = m\ddot{\mathbf{u}} - V\sigma\nabla_{\mathbf{u}}\varepsilon = \mathbf{F}_e \qquad \Rightarrow \\
&\tfrac{d^2\mathbf{u}}{dt^2} = \tfrac{V}{m}\sigma\nabla_{\mathbf{u}}\varepsilon + \mathbf{F}_e
\end{aligned} \tag{3.10}$$

where $\mathbf{F}_e$ is applied external forces (e.g. gravity and forces from other objects).

3.2 Element Free Galerkin

To simulate a deformation in a physical body a Partial Differential Equation (PDE) need to be solved. There are many ways to solve a PDE, one way is to solve it analytically. If the equations could be solved analytically a simple function evaluation could be done instead of a numerical integration. This may have some advantages like quicker evaluation and there are not any problems with numerical errors. In most cases where a PDE can be solved analytically, the domain of the solution is simplified in some ways. It could either be that the solution is symmetric in some sense, that the boundary conditions simplify the solution or the dimensions of the domain is low enough. But in most cases the boundary conditions are complex, there are seldom symmetries that simplify enough, and dimensions are sometimes even more than the three. One is then faced with an integral, for which there are no analytic solutions. This is the case for the domains that are considered in this thesis.

Other methods involve approximations like Taylor expansion or perturbation theory to get an approximate solutions close enough to the real analytic solution. Another way to approximate a solution is by using a numerical approach. Today this is done using computers and there are many theories and algorithms for this purpose. The method that will be covered in the following section is called Element Free Galerkin (EFG), and we applicate this method to solve equation (3.10).

EFG is a meshless, or mesh-free, method in contrast to the Finite Elements Method (FEM) which is a mesh-based method. In meshless methods one works with smaller "point" samples of the volume. Each point has a corresponding interaction radius in which it interacts with other nearby points. In this thesis we will call each point, *phyxel*, as an abbreviation of physical element. This name was suggested by Müller et. al[41].

3.2.1 Physics model

As mentioned earlier the dynamics of solid mechanics are calculated using strain (ε) and stress (σ). In three dimensions these quantities are symmetric 3x3 tensors (eq. (3.11)).

$$\varepsilon = \begin{bmatrix} \varepsilon_{11} & \varepsilon_{12} & \varepsilon_{13} \\ \varepsilon_{21} & \varepsilon_{22} & \varepsilon_{23} \\ \varepsilon_{31} & \varepsilon_{32} & \varepsilon_{33} \end{bmatrix}, \qquad \sigma = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{bmatrix} \tag{3.11}$$

Due to the fact that these tensors are symmetric, ε and σ can be parametrised as vectors of dimension six (see eq. (3.12)).

$$\varepsilon = \begin{bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ \varepsilon_{33} \\ \varepsilon_{12} \\ \varepsilon_{13} \\ \varepsilon_{23} \end{bmatrix}, \qquad \sigma = \begin{bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{33} \\ \sigma_{12} \\ \sigma_{13} \\ \sigma_{23} \end{bmatrix} \tag{3.12}$$

For larger deformations the Neo-Hookean solid model is a much better approximation of reality than a classical Hookean solid (see Appendix A.2. In these simulations only the classical Hookean solid will be considered, mostly due to the complexity of simulating a neo-Hookean material but also because the results are good enough for a game. Consider a classic Hookean material (i.e. obeys Hook's law), we have the relation:

$$\sigma = \mathbf{C}\varepsilon \tag{3.13}$$

where $\mathbf{C} = \mathbf{C}(E, \nu)$ is the stiffness tensor, approximating the constitutive law of the material. For isotropic material, $\mathbf{C}$ depends on the Young's Modulus,

$E \sim 10^6$–10^{12}, and Poisson's ratio, $0 \leq \nu < 0.5$, in the following way:

$$\mathbf{C} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & 1-2\nu & 0 & 0 \\ 0 & 0 & 0 & 0 & 1-2\nu & 0 \\ 0 & 0 & 0 & 0 & 0 & 1-2\nu \end{bmatrix}. \tag{3.14}$$

3.2.2 Initiation

Before the simulation can start, many constants needs to be reset and the volume which shall be simulated need to be discretized. After all phyxels have been sampled and created, matrix $\mathbf{A}$ can be calculated using eq. (3.23). Initially $\mathbf{u} = [0, 0, 0]$ for all phyxels.

Discretization

First, the volume needs to be discretized in some manner, i.e. put phyxels at the correct place in the volume. There are a couple of different ways to do that.

In computer graphics the objects are usually represented by a thin shell or mesh. This mesh is a collection of primitive geometric objects, most commonly triangles (shown in figure 3.1). The volume inside the triangles need to be discretized.

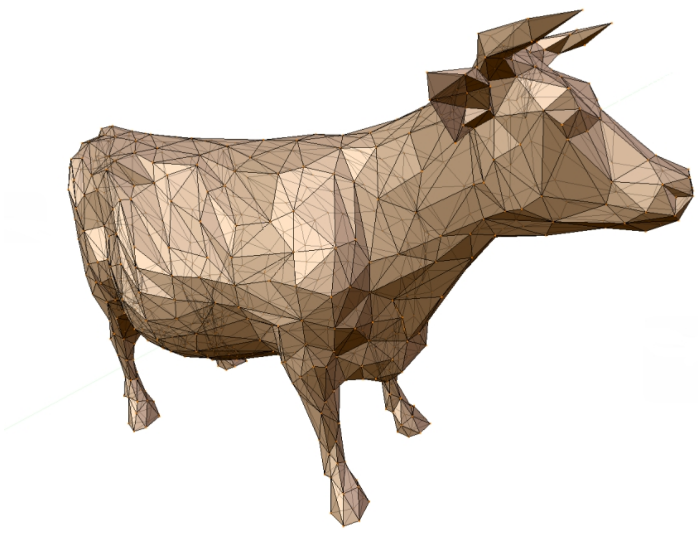

Figure 3.1: Mesh represented model of a cow. The mesh is represented by a collection of triangles.

Pauly et. al [48] suggested that an octree[1] data-structure of the model should

[1]The octree data-structure is described by Akenine-Möller and Haines [2]

be created. Start with the bounding box of the model. Construct the eight sub-boxes for the first octree level. If there are any triangles (surface of the model) in the box subdivide again. Repeat this step to a certain preset subdivision level. Then put a phyxel in the middle of each box that is inside the model (see table 3.1). In this way you get an adaptive sampling of the model, where surfaces have a higher resolution of phyxels than the centre of the model. The adaptive resolution is an advantage for this discretization algorithm although it is far more complex to implement than other more straight forward algorithms. One issue is how to determine the inside and the outside of the model.

Another method is to sample the model uniformly. An effective way to do this is by using the stencil buffer and the mesh representation already loaded by the application. The stencil buffer is like a texture but it is drawn to by a a series of operations. There are three modes that can alter the stencil buffer: `Depth Fail`, `Stencil Fail` and `Stencil Pass`. The `Depth Fail` operation will be called when the pixels depth test fails. `Stencil Fail` and `Stencil Pass` operations are called either if the stencil test is failed or passed, depending on a stencil test function which detemines pass or fail for that pixel. The user may specify whether the value in the stencil buffer at that specific pixel should be incremented, decremented, reset or set to a certain number for each of the three modes.

The algorithm can be seen in table 3.2, but a short description will follow. By altering the depth in steps (controlled by the resolution of the sampling) between 0 and 1 and then drawing the model in two passes. First clear the stencil then draw a pass by decrement the stencil for every back facing triangle and then by increment the stencil for every front facing triangle. These two passes can be done in one pass on new graphics cards that are checking front

Table 3.1: Algorithm for the sampling of a model using the octree method.

1. Initialize
   (a) Get bounding box of model.
   (b) Divide bounding box into first eight boxes of the octree, set $S_i = 0$ for each box i.
   (c) Set a maximum resolution level, R.
2. For each box i in the ocree
   (a) While $S_i \neq R$
      i. Check if any triangles are inside box i, if so subdivide box and add child boxes to octree. I.e. $S_i = S_i + 1$.
      ii. If no triangles are inside box i, goto 3.
3. Check each box i if it inside or outside the mesh
4. If inside add phyxel at box center $[x_i, y_i, z_i]$ otherwise check next box.

and back facing triangles at the same time. Lastly read out the values of the stencil buffer. If there is a non-zero value at a pixel in the stencil buffer, put a phyxel at that corresponding location. Say phyxel i is added then initiate $\mathbf{x}_i$ =`sampled position` and $\mathbf{u}_i = [0, 0, 0]$. Step the depth buffer and do it all over again (see figure 3.2). By utilising this scheme the volume can be sample fast and relatively simply. An arbitrary model can be sampled with ease as is shown in figure 3.3.

Table 3.2: Algorithm for the sampling of an arbitrary closed mesh using the stencil method.

1. Initialize
   (a) Set up the viewport to the desired sample resolution, R, in pixels (each pixel will sample one phyxel) and adjust scaling in x and y to make sure the model fits exactly.
   (b) Init camera to orthographic projection, center the model in x and y direction and place the cameras front clipping plane (z-axis) just before the smallest z-value of the model.
   (c) Init depth buffer to a value slightly further away than the smallest z-value of the model.
   (d) Set the stencil buffer to *increment* on pass for front-facing triangles.
   (e) Set the stencil buffer to *decrement* on pass for back-facing triangles.
   (f) Set the stencil pass-function to *always*
   (g) Put the step variable to $S = 0$
2. While $S \neq R$
   (a) Clear the stencil buffer
   (b) Draw the model
   (c) Analyze the stencil buffer, pixel by pixel
      i. If pixel i has non-zero value, add a phyxel at position $[x_i, y_i, z_i]$. The position can be caluclated by first deriving relative position for each pixel (e.g. $\Delta x = \frac{x_{max} - x_{min}}{R}$ for each pixel in x-direction).
   (d) Increment the depth buffer to $d = S\Delta z + \epsilon$, where $\Delta z = \frac{1}{R}$ and ϵ is the displacement which is slightly further away then the smallest z-value.
   (e) Increment $S = S + 1$.

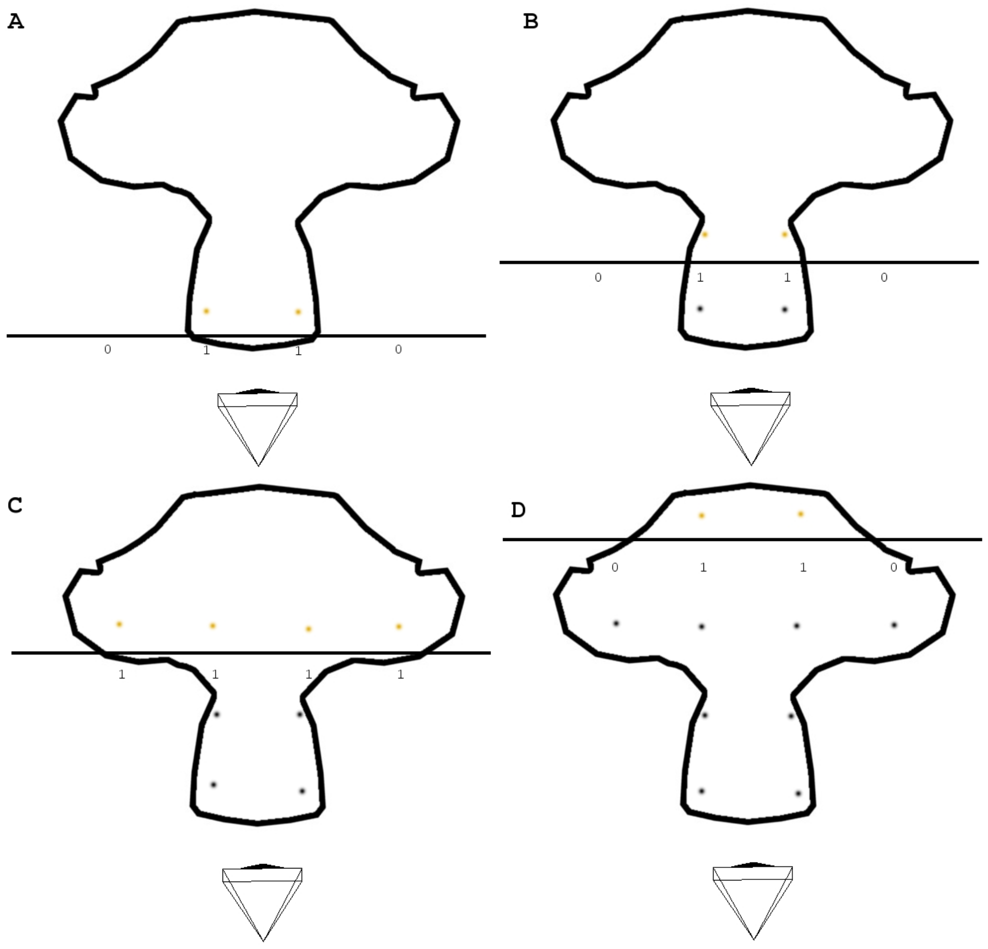

Figure 3.2: A schematic image of the stencil method using resolution 4. *Image A* shows the first depth step slightly inside the nearest polygon, stencil values is only non-zero for middle pixels. *Image B* shows the next depth step, the stencil values still show non-zero only for middle pixels. *Image C* shows the third depth step, now all four pixels show non-zero stencil values. *Image D* shows the last depth step, again only the two middle pixels have a non-zero stencil values.

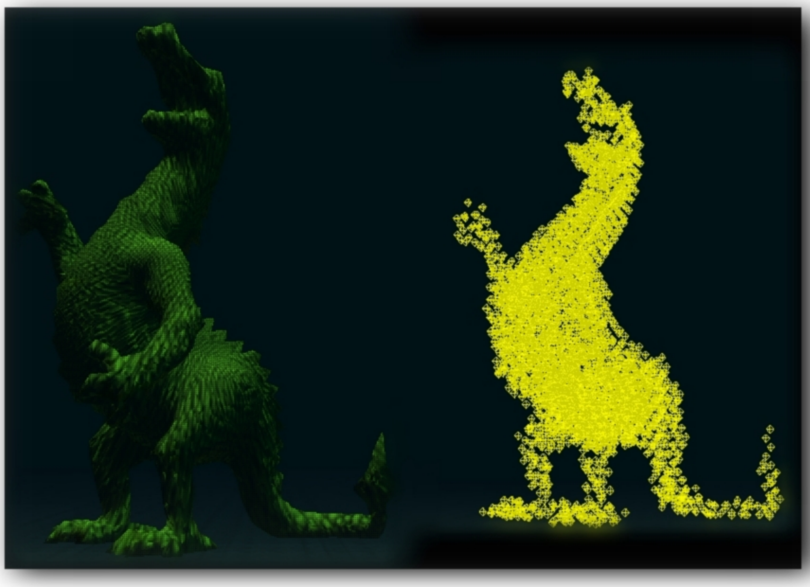

Figure 3.3: A dragon model sample using the stencil method.

3.2.3 Moving Least Square

In order to calculate the strain at each phyxel, $\nabla \mathbf{u}$ needs to be calculated. In EFG this is done using a method called Moving Least Square (or MLS). In theory this method calculates the derivative over an area by measuring differences between several points. The gradient of a vector can be seen as a function which needs to be estimated. Using least square methods one can fit a curve (or function) from a series of data points. MLS works in a similar fashion, it estimates the function using a weighted least square approximations biased towards the region around the centre phyxel. The weight should be a function (or weighting kernel), $w(r,h)$, depending on the distance, r, from the centre phyxel to the point at which the sample is taken (i.e. one of the centre phyxels neighbours) and the centre phyxels interaction radius, h. Also $\lim_{r\to h} w(r,h) = 0$, in other words the weight should go to zero as the radius approaches the interaction radius of the centre phyxel. Another criterion the weighting function needs to fulfill is that it must be normalised, i.e. $\int w(r,h)dr = 1$. More mathematically strict formulations of the criteria for a weighting kernel can be found in the article by Belytschko et. al[6].

There are several different types of weighting functions that can be used, but one of the more common in meshless simulations is the so called *poly-6* kernel[41] (see eq. (3.15) and figure 3.4).

$$w_{\text{poly6}}(r,h) = \begin{cases} \frac{315}{64\pi h^9}(h^2 - r^2)^3 & \text{if} \quad |r| < h \\ 0 & \text{otherwise} \end{cases} \tag{3.15}$$

This weighting function or kernel has the unit of $[1/m^3]$ and is previously used in both SPH[43] and EFG simulations[41]. From here on the weighting kernel will be abbreviated to $w_{ij} \equiv w(\| \mathbf{x}_j - \mathbf{x}_i \|_2, h_i)$.Using these formulations, the gradient $\nabla \mathbf{u}_i$, can be calculated for each phyxel i.

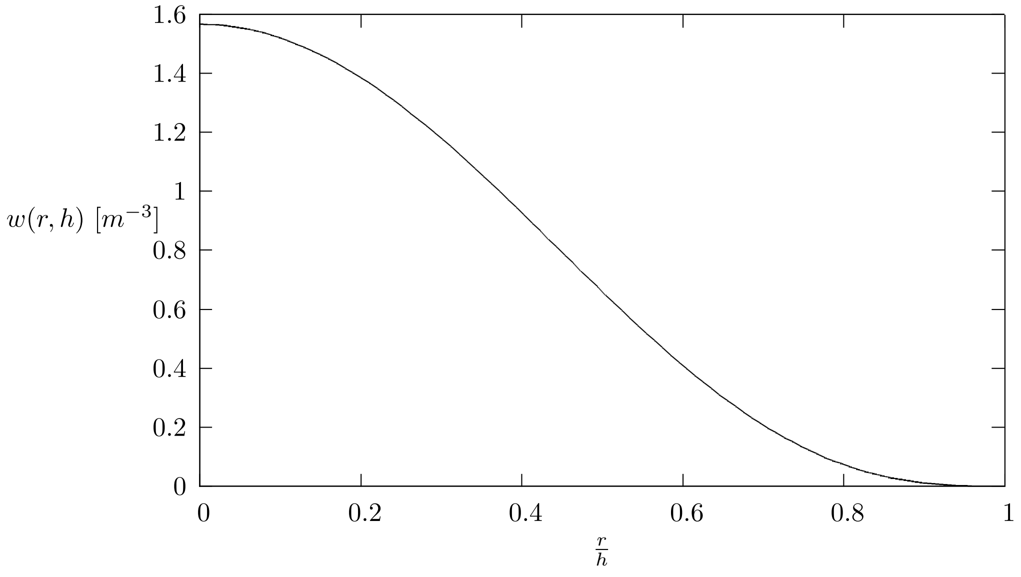

Figure 3.4: The Poly-6 function using normalised distance, r.

Moving Least Square for EFG simulations of solid mechanics

These derivations are based from other studies in this area [6, 41, 48].

The formal formulation of moving least square (MLS) approximation can be found in [6], where Belytschko et. al states the following.
Let,

$$u^h(\mathbf{x}) = \sum_{i=1}^{m} p_i(\mathbf{x}) a_i(\mathbf{x}) \equiv \mathbf{p}^T(\mathbf{x})\mathbf{a}(\mathbf{x}) \tag{3.16}$$

where $p_i(\mathbf{x})$ are monomial basis functions of the spatial coordinate $\mathbf{x}$ with the respective coefficients $a_i(\mathbf{x})$ and m is the number of terms in the basis.
The basis used in this study is linear in 3-dimensions:

$$\mathbf{p} = [1, x, y, z]^T = [1, \mathbf{x}]^T \tag{3.17}$$

There are also non-linear basis which can be used, for example the quadratic $\mathbf{p} = [1, x, y, z, x^2, y^2, z^2, xy, xz, yz]^T$. It can be shown that any function included in the basis can be reproduced by an MLS approximation.
Then define a local approximation by

$$u^h(\mathbf{x}, \widetilde{\mathbf{x}}) = \sum_{i=1}^{m} p_i(\widetilde{\mathbf{x}}) a_i(\mathbf{x}) = \mathbf{p}^T(\widetilde{\mathbf{x}})\mathbf{a}(\mathbf{x}). \tag{3.18}$$

To find the coefficients $a_i(\mathbf{x})$, a weighted least square fit for the local approximation is performed. This is obtained by minimising the difference between the local approximation and the basis function.

$$J = \sum_I w(\mathbf{x} - \mathbf{x}_I)(u^h(\mathbf{x}, \mathbf{x}_I) - u(\mathbf{x}_I))^2 = \sum_I w(\mathbf{x} - \mathbf{x}_I)\Big[\sum_i p_i(\mathbf{x}_I) a_i(\mathbf{x}) - u_I\Big]^2, \tag{3.19}$$

where $w(\mathbf{x} - \mathbf{x}_I)$ is the weighting kernel. Eq. (3.19) can be rewritten into the form

$$J = (\mathbf{Pa} - \mathbf{u})^T \mathbf{W}(\mathbf{x})(\mathbf{Pa} - \mathbf{u}), \tag{3.20}$$

where

$$\mathbf{u} = [u_1, u_2, \dots, u_n]^T$$

$$\mathbf{P} = \begin{bmatrix} p_1(\mathbf{x}_1) & p_2(\mathbf{x}_1) & \cdots & p_m(\mathbf{x}_1) \\ p_1(\mathbf{x}_2) & p_2(\mathbf{x}_2) & \cdots & p_m(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ p_1(\mathbf{x}_n) & p_2(\mathbf{x}_n) & \cdots & p_m(\mathbf{x}_n) \end{bmatrix} \tag{3.21}$$

$$\mathbf{W}(\mathbf{x}) = \begin{bmatrix} w(\mathbf{x} - \mathbf{x}_1) & 0 & \cdots & 0 \\ 0 & w(\mathbf{x} - \mathbf{x}_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & w(\mathbf{x} - \mathbf{x}_n) \end{bmatrix}$$

Differentiating J with respect to $\mathbf{a}$ and to find minimum of J, yields

$$\frac{\partial J}{\partial \mathbf{a}} = \mathbf{A}(\mathbf{x})\mathbf{a}(\mathbf{x}) - \mathbf{B}(\mathbf{x})\mathbf{u} = 0. \tag{3.22}$$

A is called the *moment matrix* and is given by,

$$\mathbf{A} = \mathbf{P}^T\mathbf{W}(\mathbf{x})\mathbf{P}, \tag{3.23}$$

and $\mathbf{B} = \mathbf{P}^T\mathbf{W}(\mathbf{x})$. This yields the coefficients

$$\mathbf{a}(\mathbf{x}) = \mathbf{A}^{-1}(\mathbf{x})\mathbf{B}(\mathbf{x})\mathbf{u}. \tag{3.24}$$

The approximation $u^h(\mathbf{x})$ can the be expressed as

$$u^h(\mathbf{x}) = \sum_{I=1}^{n} \phi_I^k(\mathbf{x})u_I, \tag{3.25}$$

where $\phi^k = [\phi_1^k(\mathbf{x}), \dots, \phi_n^k(\mathbf{x})] = \mathbf{p}^T(\mathbf{x})\mathbf{A}^{-1}(\mathbf{x})\mathbf{B}(\mathbf{x})$ is called the shape functions and the superscript k is the order of the polynomial basis.

Müller et. al[41] applied this theory directly by using solid mechanics notations:
Consider the $u(\mathbf{x})$-component of $\mathbf{u}(\mathbf{x}) = [u(\mathbf{x}), v(\mathbf{x}), w(\mathbf{x})]^T$, where $\mathbf{x} = [x, y, z]^T$. Taylor expand the scalar field, $u(\mathbf{x})$, around $\mathbf{x}_i$:

$$u(\mathbf{x}_i + \Delta\mathbf{x}) = u_i + \nabla u|_{\mathbf{x}_i} \cdot \Delta\mathbf{x} + \mathcal{O}(\| \Delta\mathbf{x} \|^2) \tag{3.26}$$

where $\nabla u|_{\mathbf{x}_i} = [\frac{\partial u_i}{\partial x}, \frac{\partial u_i}{\partial y}, \frac{\partial u_i}{\partial z}]^T \equiv [u_{,x}, u_{,y}, u_{,z}]^T$ and i denotes the phyxel for which the gradient is taken. Using this we can approximate the values u_j at neighbouring phyxel j.

$$\tilde{u}_j = u_i + \nabla u|_{\mathbf{x}_i} \cdot \mathbf{x}_{ij} = u_i + \mathbf{x}_{ij}^T \nabla u|_{\mathbf{x}_i} \tag{3.27}$$

where $\mathbf{x}_{ij} = \mathbf{x}_j - \mathbf{x}_i = [x_{ij}, y_{ij}, z_{ij}]^T$. Measure the error in the approximations as the weighted squared differences between the approximated and known values of u_j.

$$e_i = \sum_j (\tilde{u}_j - u_j)^2 w_{ij} \tag{3.28}$$

Substituting eq. (3.27) into eq. (3.28) yields,

$$e_i = \sum_j (u_i + [x_{ij}, y_{ij}, z_{ij}][u_{,x}, u_{,y}, u_{,z}]^T)^2 w_{ij} = \sum_j (u_i + u_{,x}x_{ij} + u_{,y}y_{ij} + u_{,z}z_{ij} - u_j)^2 w_{ij} \tag{3.29}$$

To minimise the error we set the derivative of e with respect to $u_{,x}$, $u_{,y}$ and $u_{,z}$ to zero. This yields three equations:

$$\left.\begin{array}{l} \frac{\partial e}{\partial u_{,x}} = \sum_j 2x_{ij}w_{ij}(u_i + \mathbf{x}_{ij}^T\nabla u|_{\mathbf{x}_i} - u_j) = 0 \\ \frac{\partial e}{\partial u_{,y}} = \sum_j 2y_{ij}w_{ij}(u_i + \mathbf{x}_{ij}^T\nabla u|_{\mathbf{x}_i} - u_j) = 0 \\ \frac{\partial e}{\partial u_{,z}} = \sum_j 2z_{ij}w_{ij}(u_i + \mathbf{x}_{ij}^T\nabla u|_{\mathbf{x}_i} - u_j) = 0 \end{array}\right\} \Rightarrow \sum_j \mathbf{x}_{ij}^T \nabla u|_{\mathbf{x}_i} = \sum_j (u_j - u_i) \tag{3.30}$$

By multiplying both sides of eq. (3.30) by $\mathbf{x}_{ij}w_{ij}$ we get the final expression:

$$\left(\sum_j \mathbf{x}_{ij}\mathbf{x}_{ij}^T w_{ij}\right) \nabla u|_{\mathbf{x}_i} = \sum_j (u_j - u_i)\mathbf{x}_{ij}w_{ij} \tag{3.31}$$

here we identify the *moment matrix* $\mathbf{A} = \sum_j \mathbf{x}_{ij}\mathbf{x}_{ij}^T w_{ij}$, which in 3-dimensions is a 3x3 symmetric matrix. This matrix only depends on the initial position, $\mathbf{x}$.

To finally calculate $\nabla u|_{\mathbf{x}_i}$, we take the inverse of the moment matrix and multiply from the left.

$$\nabla u|_{\mathbf{x}_i} = \mathbf{A}^{-1}\left(\sum_j (u_j - u_i)\mathbf{x}_{ij} w_{ij}\right). \tag{3.32}$$

The only restriction here is that $\mathbf{A}$ is invertible, i.e. $\mathbf{A}$ in non-singular. The moment matrix can be singular if the number of phyxels are less than 4 or if the neighbouring phyxels are co-planar or co-linear[41]. A “safe” inversion using Singular Value Decomposition[50], as proposed by Müller et al[41], is used here.

Also note that in all sums over the centre phyxel's neighbours the centre phyxel must also be considered a neighbour, i.e. the phyxel is considered a part of its own neighbourhood.

Calculation of gradient

In summation, the calculation of $\nabla \mathbf{u}_i$ is done by,

$$\nabla \mathbf{u}_i = \begin{bmatrix} \nabla u|_{\mathbf{x}_i}^T \\ \nabla v|_{\mathbf{x}_i}^T \\ \nabla w|_{\mathbf{x}_i}^T \end{bmatrix} = \begin{bmatrix} (\mathbf{A}^{-1}\left(\sum_j (u_j - u_i)\mathbf{x}_{ij} w_{ij}\right))^T \\ (\mathbf{A}^{-1}\left(\sum_j (v_j - v_i)\mathbf{x}_{ij} w_{ij}\right))^T \\ (\mathbf{A}^{-1}\left(\sum_j (w_j - w_i)\mathbf{x}_{ij} w_{ij}\right))^T \end{bmatrix}, \tag{3.33}$$

where $\mathbf{A} = \sum_j \mathbf{x}_{ij}\mathbf{x}_{ij}^T w_{ij}$. From this the Jacobian $\mathbf{J}$, strain ε and stress σ can be calculated for each phyxel.

$$\begin{aligned} \mathbf{J}_i &= \nabla \mathbf{u}_i + \mathbf{I} \\ \varepsilon_i &= \nabla \mathbf{u}_i + \nabla \mathbf{u}_i^T + \nabla \mathbf{u}_i^T \nabla \mathbf{u}_i = \mathbf{J}_i^T \mathbf{J}_i - \mathbf{I} \\ \sigma_i &= \mathbf{C}\varepsilon_i \end{aligned}\ , \tag{3.34}$$

where $\mathbf{I}$ is the identity matrix and $\mathbf{C}$ is the stiffness tensor from eq. (3.14).

3.2.4 Force calculations

From the differential equation derived earlier (eq. (3.10)), Newton's second law, $F = ma$ or $a = \frac{F}{m}$, can be identified:

$$\mathbf{a} = \ddot{\mathbf{u}} = \frac{1}{m} V \sigma \nabla_{\mathbf{u}} \varepsilon + \mathbf{F}_e. \tag{3.35}$$

Eq. 3.35 can be derived from the strain energy stored in a volume element. In the case of EFG simulations this volume element is a phyxel. By simply adding the subscript i for the corresponding phyxel, we have an updating scheme:

$$\ddot{\mathbf{u}}_i = \frac{1}{m_i} V_i \sigma_{\mathbf{i}} \nabla_{\mathbf{u}_i} \varepsilon_i + \mathbf{F}_{e,i}. \tag{3.36}$$

To utilise this scheme, we need to identify each of the variables above to a variable we can calculate.

Start from Newton's third law, the force acting on phyxel i is the negative sum of forces that each of its neighbours influence phyxel i with.

$$\mathbf{f}_i = -V_i \sigma_i \sum_j \nabla_{\mathbf{u}_j} \varepsilon_i \tag{3.37}$$

Now recall how $\nabla \mathbf{u}_i$ is calculated (eq. (3.33)), that $\mathbf{J}_i = \nabla \mathbf{u}_i + \mathbf{I}$ and that $\varepsilon_i = \mathbf{J}_i^T \mathbf{J}_i - \mathbf{I}$. Taking the derivative of the strain with respect to u_j we get

$$\frac{\partial}{\partial u_j}(\mathbf{J}_i^T \mathbf{J}_i - \mathbf{I}) = \frac{\partial \mathbf{J}_i^T}{\partial u_j}\mathbf{J}_i + \mathbf{J}_i^T \frac{\partial \mathbf{J}_i}{\partial u_j}, \tag{3.38}$$

where

$$\frac{\partial \mathbf{J}_i}{\partial u_j} = \frac{\partial}{\partial u_j}(\nabla \mathbf{u}_i + \mathbf{I}) = \begin{bmatrix} \frac{\partial}{\partial u_j}\nabla u|_{\mathbf{x}_i} \\ \frac{\partial}{\partial u_j}\nabla v|_{\mathbf{x}_i} \\ \frac{\partial}{\partial u_j}\nabla w|_{\mathbf{x}_i} \end{bmatrix} = \begin{bmatrix} \mathbf{A}^{-1}(\mathbf{x}_{ij} w_{ij}) \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}. \tag{3.39}$$

Then define $\mathbf{A}^{-1}(\mathbf{x}_{ij} w_{ij}) \equiv \mathbf{d}_j = [d_j|_u, d_j|_v, d_j|_w]^T$. This works analogously for $\frac{\partial}{\partial v_j}\nabla \mathbf{u}_i$ and $\frac{\partial}{\partial w_j}\nabla \mathbf{u}_i$ where the final results are

$$\begin{aligned} \frac{\partial}{\partial u_j}\nabla \mathbf{u}_i &= \begin{bmatrix} \mathbf{A}^{-1}(\mathbf{x}_{ij} w_{ij}) \cdot \hat{u} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} \\ \frac{\partial}{\partial v_j}\nabla \mathbf{u}_i &= \begin{bmatrix} \mathbf{0} \\ \mathbf{A}^{-1}(\mathbf{x}_{ij} w_{ij}) \cdot \hat{v} \\ \mathbf{0} \end{bmatrix} \quad \Rightarrow \nabla_{\mathbf{u}_j} \nabla \mathbf{u}_i = \begin{bmatrix} d_j|_u & 0 & 0 \\ 0 & d_j|_v & 0 \\ 0 & 0 & d_j|_w \end{bmatrix}. \\ \frac{\partial}{\partial w_j}\nabla \mathbf{u}_i &= \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{A}^{-1}(\mathbf{x}_{ij} w_{ij}) \cdot \hat{w} \end{bmatrix} \end{aligned} \tag{3.40}$$

Then the gradient of the strain with respect to $\nabla \mathbf{u}_j$ becomes

$$\begin{aligned} \nabla_{\mathbf{u}_j} \varepsilon_i = \nabla_{\mathbf{u}_j}(\mathbf{J}_i^T \mathbf{J}_i - \mathbf{I}) = \nabla_{\mathbf{u}_j} \mathbf{J}_i^T + \mathbf{J}_i^T \nabla_{\mathbf{u}_j} \mathbf{J}_i = \\ (\nabla_{\mathbf{u}_j} \nabla \mathbf{u}_i)^T \mathbf{J}_i + \mathbf{J}_i^T (\nabla_{\mathbf{u}_j} \nabla \mathbf{u}_i) = \mathbf{d}_j (\mathbf{J}_i + \mathbf{J}_i^T) \end{aligned}. \tag{3.41}$$

The final force equation

Finally, putting together eqs. (3.37) and (3.41), the force on phyxel i from all it's neighbours can be expressed as (here we assume that the jacobian is symmetric, $\mathbf{J} = \mathbf{J}^T$):

$$\mathbf{f}_i = -2 V_i \sigma_i \mathbf{J}_i \sum_j d_j. \tag{3.42}$$

Also remember that the force on phyxel i from phyxel j also effect phyxel j but with negative sign,

$$\mathbf{f}_{ji} = 2 V_i \sigma_i \mathbf{J}_i d_j. \tag{3.43}$$

3.2.5 Time integration

The time integration can be done in several different ways. In this thesis two of those ways will be presented.

Verlet integration

Time stepping can be done by enforcing the Verlet algorithm[54]. In contrast to the naive Euler algorithm, the Verlet algorithm is symplectic which means that it does not accumulate the errors for oscillatory differential equations. This makes it substantially more stable than the Euler algorithm. The Verlet algorithm is also more accurate than a symplectic Euler (or Leap-Frog) algorithm.

The Verlet algorithm uses implicit velocity, which makes it optimal when the velocities are not needed explicitly,

$$\mathbf{u}_i(t+\Delta t) = 2\mathbf{u}_i(t) - \mathbf{u}_i(t-\Delta t) + \frac{\mathbf{f}_i}{m_i}\Delta t^2 + \mathcal{O}(\Delta t^4). \tag{3.44}$$

For more numerical stability, the equation is usually altered to include a small dampening

$$\mathbf{u}_i(t+\Delta t) = 1.99\mathbf{u}_i(t) - 0.99\mathbf{u}_i(t-\Delta t) + \frac{\mathbf{f}_i}{m_i}\Delta t^2 + \mathcal{O}(\Delta t^4). \tag{3.45}$$

If the velocities are needed they can be calculated from the central difference of the positions[50]

$$\dot{\mathbf{u}}_i(t) = \frac{\mathbf{u}_i(t+\Delta t) - \mathbf{u}_i(t-\Delta t)}{2\Delta t} + \mathcal{O}(\Delta t^2). \tag{3.46}$$

The Verlet integration method can be practical in some types of simulations. The problem here is that if the velocities are needed in the calculation of forces, they are one timestep behind. This might not be a big issue if the timestep is small enough, but for interactive simulations the Leap-Frog method is used more frequently.

Leap-Frog integration

The Leap-Frog algorithm (also called the Symplectic Euler with some modifications) can be derived from the Verlet method as follows.
Begin with the Position Verlet equation (eq. (3.47)).

$$\mathbf{u}_i(t+\Delta t) = 2\mathbf{u}_i(t) - \mathbf{u}_i(t-\Delta t) + \frac{\mathbf{f}_i(t)}{m_i}\Delta t^2 \tag{3.47}$$

Then rearrange the equation a bit.

$$\frac{\mathbf{u}_i(t+\Delta t) - \mathbf{u}_i(t)}{\Delta t} = \frac{\mathbf{u}_i(t) - \mathbf{u}_i(t-\Delta t)}{\Delta t} + \frac{\mathbf{f}_i(t)}{m_i}\Delta t \tag{3.48}$$

Now a numerical derivative can be expressed as

$$\mathbf{v}_i(t-\frac{\Delta t}{2}) = \frac{\mathbf{u}_i(t) - \mathbf{u}_i(t-\Delta t)}{\Delta t}. \tag{3.49}$$

And the same derivative time-shifted can be expressed as

$$\mathbf{v}_i(t+\frac{\Delta t}{2}) = \frac{\mathbf{u}_i(t+\Delta t)-\mathbf{u}_i(t)}{\Delta t}. \tag{3.50}$$

Inserting eq. (3.49-3.50) into eq. (3.48) gives,

$$\mathbf{v}_i(t+\frac{\Delta t}{2}) = \mathbf{v}_i(t-\frac{\Delta t}{2}) + \frac{\mathbf{f}_i(t)}{m_i}\Delta t \tag{3.51}$$

and the position updates are derived from eq. (3.50).

$$\mathbf{u}_i(t+\Delta t) = \mathbf{u}_i(t) + \mathbf{v}_i(t+\frac{\Delta t}{2})\Delta t. \tag{3.52}$$

To summarize the Leap-Frog method is formulated as

$$\begin{cases} \mathbf{v}_i(t+\frac{\Delta t}{2}) = \mathbf{v}_i(t-\frac{\Delta t}{2}) + \frac{\mathbf{f}_i(t)}{m_i}\Delta t + \mathcal{O}(\Delta t^2) \\ \mathbf{u}_i(t+\Delta t) = \mathbf{u}_i(t) + \mathbf{v}_i(t+\frac{\Delta t}{2})\Delta t + \mathcal{O}(\Delta t^2) \end{cases} \tag{3.53}$$

The integrator needs to be initialized with half an Euler step backwards, $\mathbf{v}_i(-\frac{\Delta t}{2}) = \mathbf{v}_i 0 - \frac{\mathbf{f}_i(0)}{2m_i}\Delta t$.

The Leap-Frog method has the advantages that it does not accumulate the error as the Explicit Euler does, it is relativly cheap to calculate and it knows the velocity at this time step (even better if halv step correction is done before the velocities are used).

Half step correction is used to compensate for the half time step lag in the velocities. Simple do a half step integration (as in eq. (3.54)) on the velocities before using them in any force calculation to get a more stable solution.

$$\mathbf{v}_i(t+\Delta t) = \mathbf{v}_i(t+\frac{\Delta t}{2}) + \frac{\mathbf{f}_i(t)}{2m_i}\Delta t \tag{3.54}$$

The symplectic Euler algorithm is presented in more detail in the publication of Hairer et. al[20] or Donnely and Rogers[11]. It can also be shown that the symplectic Euler algorithm is stable under the CFL condition[39].

Implicit integration

A way to stabilise the simulations is to implement an implicit integrator instead of the explicit (or symplectic) integrator. The implicit integrators use velocities and accelerations implicitly (i.e. for the current time step) to update the positions. In this manner a stiffness matrix **K** will be formed for the problem. The linear system needs to be solved in order to get the next position. In this way one phyxels position is dependant of the force on every other phyxel. For high oscillatory problems, not many implicit integrators are suitable. The most common implicit integrator is the implicit Euler method. This algorithm will be stable, but will fatally damp out all high frequency oscillations. This is a major problem with this method, although some people might think this is a desired side product. For full derivation of the implicit Euler method in conjuction with EFG see Müller et. al[41]. We only present the implicit Euler method in short,

$$(\mathbf{M} - \Delta t^2 \mathbf{K}|_{\mathbf{u}(t)})\mathbf{v}(t + \Delta t) = \mathbf{M}\mathbf{v}(t) + \mathbf{f}(\mathbf{u}(t))\Delta t, \tag{3.55}$$

where the stiffness matrix (size $3N \times 3N$), $\mathbf{K} = \nabla_{\mathbf{u}}\mathbf{f}(\mathbf{u})$, is the gradient of the force vector, $\mathbf{f}(\mathbf{u})$, and $\mathbf{M}$ is the mass matrix (size $3N \times 3N$). The force vector, of size $3N \times 1$, is the sum of all internal and external forces acting on each of the N phyxels.

For position updates, solve eq. (3.55) for $\mathbf{v}(t+\Delta t)$ and then update the positions using an explicit scheme,

$$\mathbf{u}(t + \Delta t) = \mathbf{u}(t) + \mathbf{v}(t + \Delta t)\Delta t. \tag{3.56}$$

3.2.6 Stability

A stability analysis of the EFG method is rather complex and will not be shown here in full. Belytschko et. al[5] have nontheless written a quite extensive article on the matter. However some parts of the stability of the method will still be presented here.

In real world materials forces propagate through the material with the speed of sound. If in numerical simulations, the forces are not propagated fast enough through the material, the simulation will be unstable. In a solid, the speed of sound can be approximated from Young's modulus and the density of the material

$$c = \sqrt{\frac{E}{\rho}}, \tag{3.57}$$

where E is the Young's modulus and ρ is the density. An approximate speed of sound for the simulation is

$$c_{\text{sim}} = \frac{h}{\Delta t}, \tag{3.58}$$

where h is the interaction radius of a phyxel. So the time step Δt is critical for numerical simulations. If the time step is too large the simulation will become unstable because the simulated speed of sound will be too low for that material. Equation (3.58) is analogous with the CFL condition[2] ($c\frac{\Delta t}{\Delta x} \leq 1$), which is derived from the Verlet algorithm as well as many others. By putting $c = c_{\text{sim}}$ from eq. (3.57) and (3.58) we get,

$$\Delta t \leq h\sqrt{\frac{\rho}{E}}. \tag{3.59}$$

The thing that will happen, if the time step is too large, is that there will be unwanted high frequency oscillations in the material which will add a small (or sometimes large) error at every time step.

However there are ways of "cheating" this limit. By either implementing a

[2]Courant-Friedrichs-Levy Condition

integration method that dampens all high frequencies, like the implicit Euler method, or by implementing a method which propagates forces instantly through the material independent of the time step (i.e. infinite speed of sound). The first method might sometime work but it is expensive to calculate and it also dampens wanted oscillations and motions. The other method is preferable, with small or neglectable disadvantages, but it is difficult to implement a good method. In the Optimisations and Improvements section (Section 4.2.2) a constraint based method for achieving infinite speed of sound is presented.

A quick fix for the stability issue is to use a lower Young's modulus and then penalise volume change by adding a volume conserving force.

3.2.7 Volume conserving force

By introducing a volume conserving force, that penalises volume change, the material will seem stiffer than it actually is. It will also help prevent volume inverting displacement field, for which the Green's strain tensor is zero[41]. This kind of constraint or force was suggested by Servin et. al[53], but they never showed it in any calculations. Müller et. al[41] also recommended this kind of control and they implemeted it in their algorithms. Their energy term for it was,

$$U_v = \frac{1}{2}k_v(|\mathbf{J}| - 1)^2, \tag{3.60}$$

where k_v is a user controlled parameter similar to a spring constant and $|\mathbf{J}|$ is the determinant of the jacobian. So it penalises deviations of the jacobian from unity. The corresponding forces (which will not be derived here) will then be,

$$\mathbf{f}_i^{(\mathrm{v})} = -\nabla U_v = -k_v(|\mathbf{J}_i| - 1)\nabla_{\mathbf{u}_i}|\mathbf{J}|, \tag{3.61}$$

where it can be shown that $\nabla_{\mathbf{u}_i}|\mathbf{J}|$ can be expanded into $[(\mathbf{J}_i|_v \times \mathbf{J}_i|_w)^T, (\mathbf{J}_i|_w \times \mathbf{J}_i|_u)^T, (\mathbf{J}_i|_u \times \mathbf{J}_i|_v)^T]^T \mathbf{A}^{-1}(\mathbf{x}_{ij}w_{ij})$ for the force acted on phyxel j from phyxel i. Using Newton's third law we get,

$$\mathbf{f}_i^{(\mathrm{v})} = k_v(|\mathbf{J}_i| - 1)\begin{bmatrix} (\mathbf{J}_i|_v \times \mathbf{J}_i|_w)^T \\ (\mathbf{J}_i|_w \times \mathbf{J}_i|_u)^T \\ (\mathbf{J}_i|_u \times \mathbf{J}_i|_v)^T \end{bmatrix} \mathbf{A}^{-1}(\sum_j^{N_i} \mathbf{x}_{ij}w_{ij}), \tag{3.62}$$

where $\mathbf{J}_i = \begin{bmatrix} \mathbf{J}_i|_u \\ \mathbf{J}_i|_v \\ \mathbf{J}_i|_w \end{bmatrix}$.

3.3 Surface representation

One of the major problems with meshless simulations is the graphical representation, since at every time step of the simulation a surface representation of the point cloud need to be created. For fluid like behaviour there are algorithms like Marching Cubes[35] or Metaballs[8], but these create a very smooth surface, and it is not suitable for elasticity simulations. Below, two methods will be described in short. The Surfel method is a point-splatting method, which is popular for meshless elasticity simulations. The other method is a novelty method which we tried. This method reuses the original mesh and deforms it directly, in an attempt to both make the visualisation quicker and without the artifacts which may occur in point-splatting methods.

3.3.1 Surfels

The surface representation of using surfels is used in the meshless simulations of Müller et. al[41]. Also Pauly et. al[48] implements this method, they even uses the surfels in the fracture simulations.

A surfel is a abbreviation for Surface Element, and is a small surface sheet that represents a part of the surface. The method does not use a mesh, as is common in computer graphics. Therefore no explicit connectivity information needs to be stored between the phyxels and the surfels, which makes dynamic surface sampling simple and efficient[49]. However the surfels method is in short a point splatting method, which means that each surfel is splatted (projected) to screen space with a certain colour. This might produce artifacts which are not visible for ordinary mesh based shading. Sharp creases might also be difficult to visualise using the surfels method. There are certain algorithms to improve these flaws. For example, the CSG method[55] could be used to get more detail and sharp creases. The results are quite remarkable, but the sampling resolution needs to be high. This is usually seen by the poor performance of the method. But Guennebaud and Gross[19] developed a GPU-based method that could handle more than 40 million fitting points per second.

3.3.2 Mesh based

When the model is imported to the simulation, it is represented as a surface mesh (see figure 3.1). Why not use the original shape and connectivity to construct deformations? The mesh based method uses this surface representation (the mesh) of the 3D object directly, to visualise the deformations and eventual fractures. The method automatically subdivides edges and faces when necessary. It also tries to snap new vertices to pre-existing faces and edges instead of creating new ones. If there is a fracture, the vertices at which the fracture starts will be split into two. And new faces will propagate in the fracture direction (see figure 3.5), given by the eigenvectors of the stress tensor.

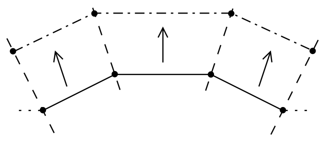

Figure 3.5: Propagation of the edge its fracture direction. The fracture direction is approximated from the fracture direction given by the simulation and the position of the neighbouring edges.

More on this method can be read in the thesis by Jonas Lindmark, University of Linköping[32].

Chapter 4

Methods

4.1 Simulation

By utilising the theory presented earlier (chapter 3), everything boils down to these algorithms.

4.1.1 Elasticity

The algorithm for the elasticity simulation is:

1. Initialisation
   - (a) Discretize the model and set $\mathbf{x}_i$
   - (b) For each phyxel i do
     - i. Set $\mathbf{u}_i^{(+)} = \mathbf{0}$, $\mathbf{u}_i^{(-)} = \mathbf{0}$, $\mathbf{v}_i^{(+/2)} = \mathbf{0}$, $\mathbf{v}_i^{(-/2)} = \mathbf{0}$ and $\mathbf{f}_i^{(\mathrm{i})} = \mathbf{0}$
     - ii. Find the k nearest neighbours and calculate $h_i = \frac{\sum_j^k(\|\mathbf{x}_{ij}\|_2)}{k}$ (we use $k = 9$) or use a fixed radius using a scale of the lattice distance as mentioned in the results
     - iii. Calculate $\mathbf{A}_i = \sum_j \mathbf{x}_{ij}\mathbf{x}_{ij}^T w_{ij}(h_i)$ and invert (preferably using Singular Value Decomposition, to avoid problems with singular matrices) to get $\mathbf{A}_i^{-1}$
     - iv. Calculate $m_i = \frac{3\pi\rho h_i^3}{4}$ (where ρ is the global density)
     - v. Set external force to gravity force only, $\mathbf{f}_i^{(\mathrm{e})} = m_i\mathbf{g}$
   - (c) For each phyxel i do
     - i. Find all N_i phyxels within the radius h_i.
     - ii. Calculate local density $\rho_i = \sum_j^{N_i} m_j w_{ij}(h_i)$
     - iii. Calculate local volume $V_i = \frac{m_i}{\rho_i}$
2. Run loop
   - (a) For each phyxel i do
     - i. Calculate $\nabla\mathbf{u}_i = \begin{bmatrix} \mathbf{A}_i^{-1}\left(\sum_j^{N_i}(u_j - u_i)\mathbf{x}_{ij}w_{ij}\right) \\ \mathbf{A}_i^{-1}\left(\sum_j^{N_i}(v_j - v_i)\mathbf{x}_{ij}w_{ij}\right) \\ \mathbf{A}_i^{-1}\left(\sum_j^{N_i}(w_j - w_i)\mathbf{x}_{ij}w_{ij}\right) \end{bmatrix}$

```
        ii. Calculate the Jacobian $\mathbf{J}_i = \nabla \mathbf{u}_i + \mathbf{I}$
        iii. Calculate strain $\varepsilon_i = \frac{1}{2}(\nabla \mathbf{u}_i + \nabla \mathbf{u}_i^T + \nabla \mathbf{u}_i^T \nabla \mathbf{u}_i)$
        iv. Calculate stress $\sigma_i = \mathbf{C}\varepsilon$
        v. Calculate the force matrix $\mathbf{F}_i^{(\mathrm{i})} = -2V_i \mathbf{J}_i \sigma \mathbf{A}^{-1}$
        vi. If volume conserving force is used, calculate the force matrix for
            this, $\mathbf{F}_i^{(\mathrm{v})} = k_v(|\mathbf{J}_i - 1) \begin{bmatrix} (\mathbf{J}_i|_v \times \mathbf{J}_i|_w)^T \\ (\mathbf{J}_i|_w \times \mathbf{J}_i|_u)^T \\ (\mathbf{J}_i|_u \times \mathbf{J}_i|_v)^T \end{bmatrix} \mathbf{A}^{-1}$. Otherwise put
            $\mathbf{F}_i^{(\mathrm{v})} = \mathbf{0}$.
        vii. For each phyxel $j$ of the neighbourhood $N_i$ do
            A. Calculate force on phyxel $j$ as $\mathbf{f}_j^{(\mathrm{i})} = \mathbf{f}_j^{(\mathrm{i})} + (\mathbf{F}_i^{(\mathrm{i})} + \mathbf{F}_i^{(\mathrm{v})})\mathbf{x}_{ij} w_{ij}$
            B. Add reaction force to phyxel $i$, $\mathbf{f}_i^{(\mathrm{i})} = \mathbf{f}_i^{(\mathrm{i})} - (\mathbf{F}_i^{(\mathrm{i})} + \mathbf{F}_i^{(\mathrm{v})})\mathbf{x}_{ij} w_{ij}$
    (b) For each phyxel $i$ do
        i. Set $\mathbf{u}_i^{(\mathrm{tmp})} = \mathbf{u}_i^{(+)}$
        ii. Time integrate,
            $\mathbf{v}_i^{(+/2)} = \mathbf{v}_i^{(-/2)} + \frac{\Delta t}{m_i}(\mathbf{f}_i^{(\mathrm{i})} + \mathbf{f}_i^{(\mathrm{e})})$
            $\mathbf{u}_i^{(+)} = \mathbf{u}_i^{(-)} - \frac{\mathbf{v}_i^{(+/2)} \Delta t}{m_i}$
        iii. Set $\mathbf{u}_i^{(-)} = \mathbf{u}_i^{(\mathrm{tmp})}$
        iv. Reset forces, $\mathbf{f}_i^{(\mathrm{i})} = \mathbf{0}$ and $\mathbf{f}_i^{(\mathrm{e})} = m_i \mathbf{g}$
    (c) Goto step 2
```

The time integration here is the Leap-Frog algorithm, but it can easily be replaced by your favourite time integrator.

4.1.2 Plasticity

Plasticity occurs when the material is subjected to so much strain that its internal structure collapses and the shape of the object is permanently deformed. An intuitive example could be a chunk of clay, which has a very low threshold of plasticity. The clay is easily deformed by strain and does not go back to its original shape when the strain is lowered. On the other side of the scale is a piece of rubber, which is also easily deformed but it has much higher threshold for plasticity. When the strain that causes deformation is lowered, the piece of rubber goes back to its original shape.

There are ways to incorporate plasticity into the simulation. For small plastic deformations it is usually enough to use the method of strain state variables proposed by O'Brien et. al[45]. Their method is based on physical properties of the material, where elastic strain is absorbed by plastic strain if it reaches a certain threshold[18]. If the deformations are large, a method needs to be incorporated in order to keep the simulation stable. This method continuously absorbs the strain into a plastic strain variable and in that way allow much larger plastic deformations without additional instability[41], but at the cost of a potentially slower and more inaccurate simulation.

Small Plastic Deformations

The method of strain state variables was introduced by O'Brien et. al[45]. They stored the plastic strain in a state variable based on von Mises yield condition. Müller[18] presented a direct use of the strain state variable for EFG simulations of plastic behaviour. The elastic strain considered in every phyxel i is expressed as,

$$\varepsilon_i^{\text{elastic}} = \varepsilon_i - \varepsilon_i^{\text{plastic}}, \tag{4.1}$$

where ε_i is the measured strain and $\varepsilon_i^{\text{plastic}}$ is the plastic strain state variable. Every phyxel i should initialise the plastic strain to zero, $\varepsilon_i^{\text{plastic}} = \mathbf{0}$. The plastic strain is updated, at every time step, as follows:

1. Calculate ε_i as in only elastic simulations
2. $\varepsilon_i^{\text{elastic}} = \varepsilon_i - \varepsilon_i^{\text{plastic}}$
3. If $\| \varepsilon_i^{\text{elastic}} \|_2 > c_{\text{yield}}$ then
   (a) $\varepsilon_i^{\text{plastic}} = \varepsilon_i^{\text{plastic}} + c_{\text{creep}} \varepsilon_i^{\text{elastic}}$
4. If $\| \varepsilon_i^{\text{plastic}} \|_2 > c_{\text{max}}$ then
   (a) $\varepsilon_i^{\text{plastic}} = c_{\text{max}} \frac{\varepsilon_i^{\text{plastic}}}{\|\varepsilon_i^{\text{plastic}}\|_2}$

There are three scalar parameters in the algorithm. c_{yield} is the yield criterion, where the material starts to plastically deform due to the high strain, $0 < c_{\text{creep}} < 1$ is the speed at which the elastic strain is absorbed into plasticity (small values gives a slow plastic flow in the material) and c_{max} is the maximum amount of strain that can be stored in the material before it fractures (i.e. the *fracture limit*). Here fractures are not incorporated in the model, the solution for that flaw is to scale down the plastic strain to the maximum amount.

For the stress calculation, we assume that the plastic strain is constant during one time step. We can then use the elastic strain instead of measured strain to calculate the stress,

$$\sigma_i = \mathbf{C}\varepsilon_i^{\text{elastic}}. \tag{4.2}$$

This is then used to update all forces.

Larger Plastic Deformations

If the deformations are too large the reference positions of two phyxels, $\mathbf{x}_i$ and $\mathbf{x}_j$, might be within each others support radius while the deformed positions $\mathbf{x}_i + \mathbf{u}_i$ and $\mathbf{x}_j + \mathbf{u}_j$ are far away from each other. This leads to very large strains, stresses and elastic forces which will crash the simulation. The method proposed to solve this was presented by Müller et. al[18, 41]. They suggested that after each time step, the measured strain is absorbed into the plastic strain state variable. At the same time the reference positions moves to the deformed positions and a new neighbourhood of phyxels is found. In this way both reference shape and deformed shape are identical after each time step, which yields a much more stable simulation. Strain is not lost, but saved in the plastic strain state variable. On the other hand, the original shape information will be lost and the object will be permanently deformed. Also small errors might sum up over time which gives a somewhat inaccurate result, but not visually noticeable.

The algorithms proposed by Müller et. al, in a chapter of Gross and Pfister's book on Point based graphics[18], are

1. For each phyxel i
   (a) Absorb strain into the plastic strain variable, $\varepsilon_i^{\text{plastic}} = \varepsilon_i^{\text{plastic}} - \varepsilon_i$
   (b) Set reference position to the deformed position, $\mathbf{x}_i = \mathbf{x}_i + \mathbf{u}_i$
   (c) Reset displacement vector, $\mathbf{u}_i = \mathbf{0}$
2. For each phyxel i
   (a) Find all neighbours, N_i, within support radius h_i
   (b) Update m_i, ρ_i, V_i and $\mathbf{A}_i^{-1}$

4.1.3 Fracture

The fracture part was not implemented in this thesis, but the method will be covered here in theory. The methods are based on the simulations of Müller et. al[42] and Pauly et. al[48].

Based on fracture mechanics theory, a crack could be initialised when the stress reaches a certain threshold. To find this threshold, the eigenvalues of the stress tensor is calculated for each phyxel. A new crack is then initialised if the largest eigenvalue d_{max} exceeds the limit for tensile fracture (material parameter). This corresponds to mode `I` fracture (opening mode fracture)[3]. The fracture direction will then be defined by, the to the corresponding eigenvector of the largest eigenvalue, perpendicular vector, α.

Given the fracture vector and position of the phyxel, the surface model is split and propagated a small amount in the fracture direction. How far the slit will be propageted depends on a material dependant propagation speed variable c_{prop}. While propagating the crack new faces are created in the surface mesh. The phyxel at which the split occurred is duplicated, the two phyxels are placed on one side each of the split. New neighbourhoods are calculated for all phyxels before the next simulation step.

In order to take fractures into account a visibility criterion is inserted. The criterion, introduced by Belytschko et. al[7], states that a phyxel only recognises neighbours if they are visible. I.e. if a ray can be shot from one phyxel to the other (within the support radius) without hitting a face of the mesh, that phyxel is considered a neighbour and is added to the neighbourhood. This may cause discontinuities in the simulation which may contribute to incorrect results or instability. A solution to this issue is to implement a transparency function, proposed by Organ et. al[47]. The idea of the transparency method is to let phyxels "see" through faces if they are close enough. In practice the weight function used in the EFG simulations is modified to add the distance from the point where the ray intersects the face to the crack front[48]. This method effectively removes discontinuities in the simulation volume.

4.2 Optimisations and Improvements

There are many optimisations that can be done to the implementations. All from the most fundamental programming tactics, like implementing vectorized operations (e.g. SSE) to more use complex and effective data-structures. A place where large speedups can be achieved is the neighbourhood search for each phyxel. By implementing spatial hashing[12] this could yield a significant speed increase. Caching values that are reused often, a small speed increase might also be gained. Other improvements are more algorithm based. By implementing schemes to use larger time steps, the simulation can be run in real-time with lower framerates. Using dynamic sampling of phyxels yield fewer phyxels in total but no loss in simulation accuracy.

The main EFG simulation can be parallelised on several processors by using three sync passes. This can also be utilised in GPU simulations of the simulation.

4.2.1 Parallelisation and simulation on GPU

The graphics processing unit or GPU uses hardware that is highly parallel. The unit can consist of, at the time of writing, as much as 256 processors working at 1.5 GHz each[1]. The processors are much less complex than a CPU on a computer but can perform many simple tasks effectivly. The GPU processors are optimised to carry out vector calculations for graphics applications. In other words, operations like projections of vectors, scalar/vector multiplication and read/write operations. But recently the graphics processors are used for more general purposes. For example, more complex rendering algorithms[9] and real-time physics simulations[22].

In order to run physics on the GPU the algorithms must be able to be parallelised. This means that different tasks of the algorithms should be able to run simultaneously, preferably without depending on each other. The EFG simulation is quite suitable for this kind of simulation since each phyxel only reads data from its neighbouring phyxels. But the phyxel needs to save its results at some point, that is why synchronisation is needed.

A general method for parallelisation will be presented here (for more GPU specific algorithms see [21, 22, 25, 26, 44]).

For parallel simulation of EFG first assume all phyxels are synchronised, and know which its neighbours are. Then the first pass will be to update m_i, ρ_i, V_i and $\mathbf{A}_i^{-1}$ if the phyxels neighbourhood has changed since the last step. After each step, each process must wait for all other processes to finish before starting the next step. The waiting is referred to as synchronisation.

The next pass will be to calculate strain, stress and forces. Since each force contribution from phyxel j to phyxel i will give negative contribution back to phyxel j, safe methods to write to another phyxel are needed. Safe methods

[1]Data taken from nVidia graphics card 9800 GX2

mean that read/write conflicts must be avoided. This may be the most diffucult step when simulating on a GPU using ordinary shaders. New libraries, like nVidia CUDA[2], can give developers access to the whole graphics memory shared between the processors and also use a programming language that is more general than most shader languages available. For example nVidia CUDA uses a C compiler to build shaders.

But parallelism can be used when simulating on the CPU as well. Today it is not uncommon to have four or even eight cores on the CPU and the trend indicates that the number of cores will grow[23]. For more information on parallel programming and the issues that come with it see Fosters book on this subject[15].

The last pass is to time integrate, using the forces that were accumulated in the previous pass.

The neighbourhood search could be difficult to parallelise, especially on a GPU where memory read/write operations are different from that of a CPU implementation, however there are studies that have shown success in this area[21, 26].

4.2.2 Regularized Constraint Based Simulation

Lacoursière[28] presented a regularize constraint based method to improve stability of many types of simulations. Servin et. al[53] used those formulations in a method to allow stiffer materials without taking finer time steps. The method was implemented on FEM simulations of solid mechanics. Their results were remarkable, but there are no direct formulations on how to utilize their method on EFG simulations. The difficulies in EFG simulations is to find the right constraints without loosing the actural core of the method.

4.3 Material

The equipment used was a 64-bit quad core Intel® Xeon® CPU at 2.33 MHz per core, 4096 MB of RAM and a nVidia® GeForce® 8800 GTS graphics card with 640 MB of dedicated memory.

The work station ran 64-bit Microsoft® Windows Vista™ Enterprise. The implementation was done on Microsoft® Visual Studio® 2005 and utilised Direct3D 10 graphics. Also OpenMP was used for thread handling and Havok™ 4 for basic rigid body physics.

Other software used in this project:
Blender 3D, Emacs, GIMP, LaTeX, GNUPlot.

[2]http://www.nvidia.com/object/cuda_home.html (visited July 17, 2008)

Chapter 5

Results

An implementation of the theory and algorithms mentioned in earlier chapters was done. This implementation showed that it was possible to run EFG simulations in real-time (see table 5.1). The materials that could be simulated with some measure of stability, were mostly jelly like. This was due to the relatively low Young's modulus used. The Young's modulus used in the simulations was of the order of, $E \sim 10^8$. As real rubber has a Young's modulus of roughly, $E_{\text{rubber}} \sim 10^{11}$, one can easily see that the simulated material will be far too soft. In order to get a material with some form of rigidity, the volume conserving force was adjusted to a appropriate level. In simulations with a simple model of a cube, a spring constant for the volume conserving force was chosen as $k_v \sim 10^7$. During the simulation we kept the interaction radius of each phyxel to approximatly one BCC[1] cells in radius for interactive simulations (corresponds to 9 phyxels) and two BCC cells in radius for more accurate simulations. We use the BCC formation both because it represents a common formation in solids but mostly because we need some measurement of how many phyxels to include in the simulation (i.e. what interaction radius to choose).

Table 5.1: Results from the implemented EFG simulation ($\Delta t = 0.01$)

Number of phyxels	Mean number of neighbours	Frames per second $[s^{-1}]$
48	37.9	227.8
100	74.9	83.5
180	104.4	44.0
294	133.5	21.6
448	127.7	12.3
448	248.3	8.2
1210	66.6	7.2

[1]BCC: Body Centered Cubic formation (a crystal formation used in e.g. Solid State Physics)

5.1 Benchmarks

The benchmarks taken from the implemented simulation show that the simulation scale linearly with both the amount of phyxels in the simulation (figure 5.1) and the number of neighbours each phyxel has (figure 5.2).

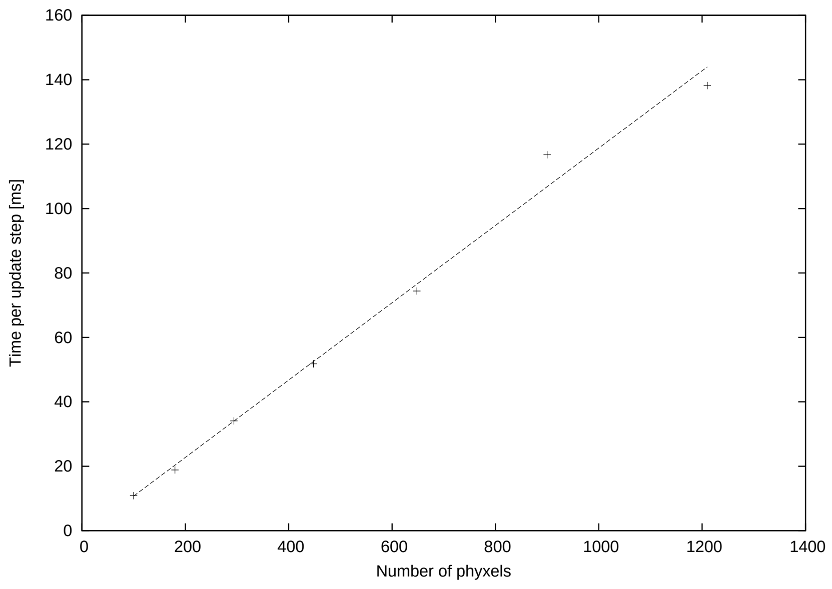

Figure 5.1: The figure shows the linear scaling between the number of phyxels in the simulation and the time for one update step ($R^2 = 0.997$). Here the mean number of neighbours each phyxel had was kept constant.

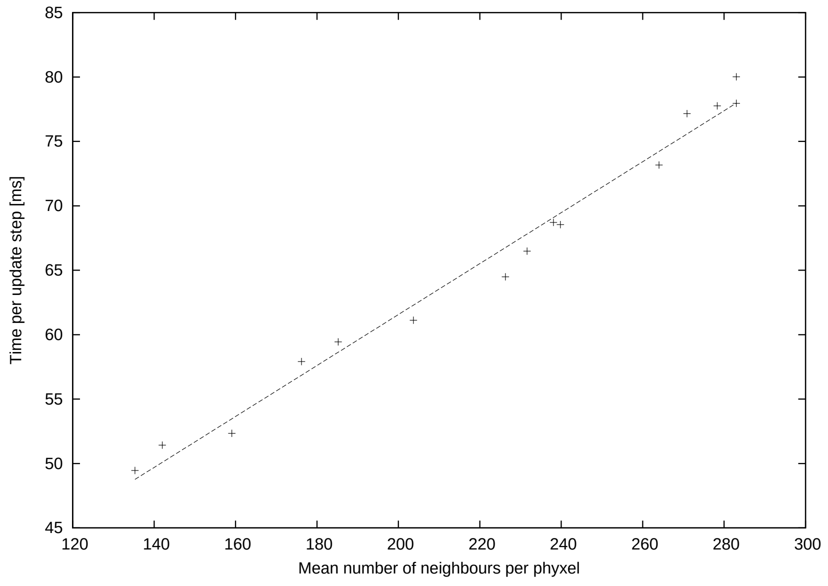

Figure 5.2: The figure shows the linear scaling between the mean number of neighbour of each phyxel in the simulation and the time for one update step ($R^2 = 0.994$). Here the total number of phyxels in the simulations was kept constant.

5.2 Screenshots

Below are a few screenshots of the simulation of a cube. The first set of screenshots (fig. 5.3) show a simple deformation of the cube. The second set of screenshots (fig. 5.4) show the plastic deformation of the cube, where some phyxels are far from their original position due to a large force that pulled them towards the ground plane.

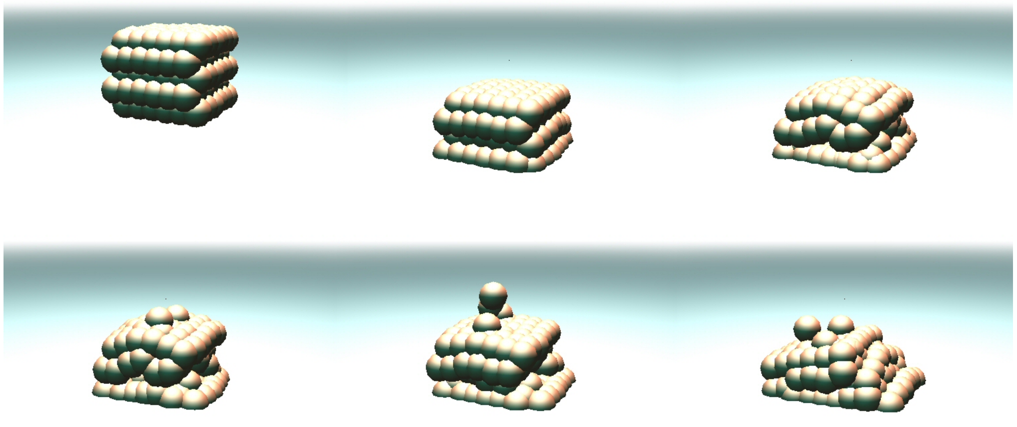

Figure 5.3: Simulation of a soft deformable cube.

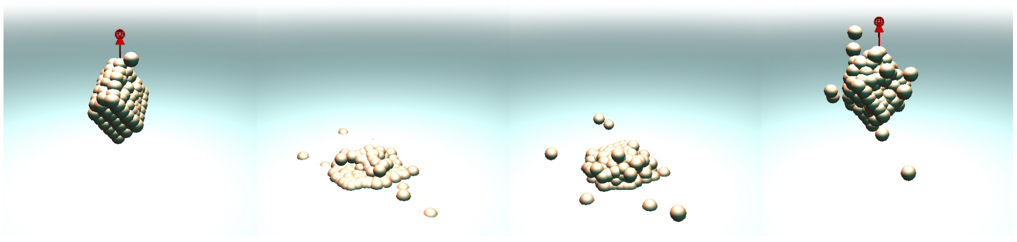

Figure 5.4: Plasticity simulation of the cube.

Chapter 6

Discussion

For games and other interactive appliances the real-time aspect is very important. The physics simulations are but a small part of the entire game, every part has a certain amount of processing time allocated for each time step. If the task could be put on another processor or on the graphical processing unit, the allocated time will be larger, but the simulation must still run in at least 15 frames per second, in order to look good enough. This thesis has shown that it is possible to run simulations of deformable materials in real-time, and it is possible to split the workload on several processors in an intuitive way. This is a sign that the method has great potential on future gaming consoles and computers, which strives towards more cores and more processors rather then higher clock rates[23]. The plasticity simulation showed that large deformations were possible without losing overall stability. The volume conserving force, introduced, also helped to achieve a more rigid material than the Young's modulus would suggest. This is important for the visual experience for the end user. In games we could sacrifice much of the physical correctness as long as it "looks good". In the case of EFG simulations we get correct simulations perhaps at the cost of numerical instability for the low sampled objects, which we need in real-time simulations.

The problems with numerical instability for high values of Young's modulus is disconcerting. The current implementation is limited by the CFL condition which yields material properties much like jelly, for the time steps needed for real-time simulations. And it is difficult to get a stable simulation at all for some models with complex geometry. The implementation of a regularized constraint based simulation of EFG seems promising, although we have not tried to implement in full scale yet. If the speed of sound could be ignored in the simulations, it should be possible to simulate materials of any stiffness. Another way might be to simply lower the time step and instead increase the framerate. This might be done in a GPU implementation of the simulation, where the computing power potential is high. The only problem with GPU simulations is how to handle memory transfers. The solution might be to do the simulation entirely on the GPU, both simulation and visualisation. The new field of GPGPU[1] where the GPU is programmed using a kind of C-code and the devel-

[1]General Purpose Graphical Processing Unit

oper has to think less about hardware, might simplify these tasks.

The visualisation of the simulation was not complete for our implementation, in the sense that we had no explicit connection between simulation and visualisation. We worked on these implementations separately and had no time to try our theories in practice and join together the simulation and visualisation parts. The method of altering the surface mesh directly, instead of generating a new surface each time step, showed great potential though. The ability to simply change the mesh instead of rebuilding it each time step, should give higher framerates. At the moment the algorithm is still un-perfected and needs a lot more work in order to function in real-time applications. For example, the algorithms searches for shortest routes in the mesh, could be speeded up by implementing more effective data-structures and search algorithms. Also the method should be possible to parallelise, so that it can be run on the GPU. However due to the fact that there was no connection between the visualisation and simulation, fractures were near impossible to model using the proposed method.

For fracture simulation, a surface mesh is needed to do ray-triangle test between two phyxels, this must be done in order to determine whether or not there was a gap/crack between the two phyxels. Since we did not have a valid surface representation, fracture simulations were never implemented. However the theory was explored and the implementation was adjusted to be able to incorporate fracture simulations in the future.

For fracture simulations FEM might be more straight-forward to implement, since the domain that is simulated is already discretized in a manner that simplifies production of new cracks. As for the stability issue, it is uncertain which method is best suited. For FEM there is documented success with a regularized constraint based method[53], but it is a tedious work to tessellate a surface mesh into a volumetric tetrahedral mesh in a general way. The EFG simulations offer higher possibilities for dynamic sampling and simulation. So perhaps in the long run it is best to put more effort on EFG simulations, mostly because of its dynamic properties and absence of discontinuities in the simulation volume. Another advantage for the EFG model could be its ability to easily be parallelised. As for FEM there are not many publications on FEM simulations on the GPU (apart from a US patent[1]).

6.1 Future Work

A comparative study showing the actual differences between FEM and EFG simulations, with respect to simulation speed, visual accuracy and stability, would be most interesting. The field of constraint based EFG simulation should also be explored further. To be able to simulate any kind of material, independent of the size of the time step, might be done using the constraint method. Perhaps is is possible to construct an algorithms using a constraint based method with EFG simulations. A valid GPU simulation of elastic materials, using the EFG method, is also proposed. To run the simulation on the GPU will not only free up important CPU-time, but might also speed up the actual simulation. If the visualisation could be done on the GPU as well the need for costly data transfer could be reduced. Then of course the field of surface representation, that is a large research field already, must find algorithms that are more suitable for real-time simulations and GPU implementations. Work with mesh based deformation in conjunction with meshless simulations should also be explored further, especially the possibility to run the surface representation on the GPU, so that no data transfer between the CPU and the GPU are needed for GPU simulations.

6.2 Conclusion

It was shown that the EFG simulation can be run in real-time. Although there are issues in what size of time step is allowed. This is decided from $\Delta t \leq h\sqrt{\frac{\rho}{E}}$, where h is the mean interaction radius (or support radii), ρ is the density and E is Young's modulus. If the time step is too large the simulation will quickly become unstable. The three other parameters can then be altered in order to get the desired time step, but often with the result that the material will look like jelly (for $\Delta t \sim 0.01$). Further work is suggested on finding ways to keep the time step large while also increasing the stiffness of the material (i.e. increasing Young's modulus).

6.3 Acknowledgements

Thanks to EA Digital Illusion CE AB who have supplied the nesseccary equipment and support to fulfill the thesis. A special thanks to my advisors at EA DICE AB, Joakim Lord and Torbjörn Söderman. I also like to acknowledge my co-worker on this project, Jonas Lindmark from the University of Linköping, who worked on the visualisation of the project and gave me support in many programming issues.

Appendix A

Math and Physics

A.1 Hamilton's equations

A method of deriving a system of first order differential equations from the energy of the system, is Hamilton's equations.

Let, $\mathcal{H} = T + V - \sum_v \frac{\partial V}{\partial \dot{q}_v}\dot{q}_v$, then:

$$\begin{cases} \dot{p}_v = -\frac{\partial \mathcal{H}}{\partial q_v} \\ \dot{q}_v = \frac{\partial \mathcal{H}}{\partial p_v} \\ \frac{\partial \mathcal{H}}{\partial t} = -\frac{\partial \mathcal{L}}{\partial t} \end{cases}, \tag{A.1}$$

where q_v are the generalised coordinates, $\dot{q}_v = \frac{dq_v}{dt}$, $p_v = \frac{\partial \mathcal{L}}{\partial \dot{q}_v}$ are the generalised momenta, $\dot{p_v} = \frac{dp_v}{dt} = \frac{\partial \mathcal{L}}{\partial q_v}$, $\mathcal{H}$ is the Hamiltonian, $\mathcal{L}$ is the Laplacian, T is the kinetic energy and V is the potential energy. The term $\sum_v \frac{\partial V}{\partial \dot{q}_v}\dot{q}_v$ is zero if there are only monogenic forces.

A.2 Neo-Hookean solid

The strain, σ, for a neo-Hookean solid is expressed using the Finger tensor, $\mathbf{B}$, and the pressure, ρ, as

$$\sigma = -\rho\mathbf{I} + \mu\mathbf{B}, \tag{A.2}$$

where μ is the shear modulus of the material. The Finger tensor can be computed as the inverse of the strain tensor, $\mathbf{B} = \varepsilon^{-1}$. And the strain energy stored in the material is defines as

$$U = \frac{1}{2}\mu \cdot \mathrm{tr}(\mathbf{B}), \tag{A.3}$$

where the tr() operator is the trace and U is the potential energy per volume.

List of Tables

List of Figures

References

[1] Shmuel Aharon. GPU-based finite element. US Patent, No. US 2005/0243087 A1, 2005.

[2] Tomas Akenine-Möller and Eric Haines. *Real-Time Rendering.* A K Peters, Ltd., 2nd edition, 2002.

[3] Ted L. Anderson. *Fracture Mechanics, fundamentals and applications.* CRC Press, 1995.

[4] Zhaosheng Bao, Jeong-Mo Hong, Joseph Teran, and Ronald Fedkiw. Fracturing rigid materials. *Transactions on Visualization and Computer Graphics*, 13(2):370–378, 2007.

[5] Ted Belytschko, Yong Guo, Wing Kam Liu, and Shao Ping Xiao. A unified stability analysis of meshless particle methods. *Int. J. Numer. Meth. Engng*, 48:1359–1400, 2000.

[6] Ted Belytschko, Yuri Krongauz, Daniel Organ, Mark Fleming, and Petr Krysl. Meshless methods: An overview and recent developments. *Computer Methods in Applied Mechanics and Engineering*, 139(1–4):3–47, 1996.

[7] Ted Belytschko, Yun Yun Lu, and Lei Gu. Element-free galerkin methods. *International Journal for Numerical Methods in Engineering*, 37:229–256, 1994.

[8] James F. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, 1(3):235–256, 1982.

[9] Andrew Corrigan, John Wallin, and Matej Vesenjak. *Progress on Meshless Methods*, chapter Visualization of meshless simulations using Fourier volume rendering. Springer, 2008.

[10] Richard Courant. Variational methods for the solution of problems of equlibrium and vibration. *Bulletin of the American Math Society*, 49:1–61, 1943.

[11] Denis Donnely and Edwin Rogers. Symplectic integrators: An introduction. *American Journal of Physics*, 73(10):938–945, 2005.

[12] Kenny Erleben, Jon Sporring, Knud Henriksen, and Henrik Dohlmann. *Physics-Based Animation.* Charles River Media, INC., 1st edition, 2005.

[13] Helmut Falkenheiner. Calcul systématique des charactéristiques élastiques des systémes hyperstatiques. *Rech. Aero.*, (17), 1950.

[14] Helmut Falkenheiner. La systématisation du calcul hyperstatiques d'aprés l'hypothèse du “schéma du champ homogène.”. *Rech. Aero*, (2), 1951.

[15] Ian Foster. *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering.* Addison Wesley, 1995.

[16] Boris Grigoryevich Galerkin. Series solution of some problems in elastic equilibrium of rods and plates. *Vestnik inzhenerov i tekhnikov*, 19:897–908, 1915.

[17] Robert A. Gingold and Joseph J. Monaghan. Smoothed particle hydrodynamics: Theory and application to non-spherical stars. *Monthly Notice of the Royal Astronomical Society*, 181:375–389, 1977.

[18] Markus Gross and Hanspeter Pfister, editors. *Point-Based Graphics.* Elsevier Inc., 2007.

[19] Gaël Guennebaud and Markus Gross. Algebraic point set surfaces. In *ACM SIGGRAPH 2007*, 2007.

[20] Ernst Hairer, Christian Lubich, and Gerhard Wanner. *Geometric Numerical Integration*, volume 31 of Spring Series in Computational Mathematics. Springer Verlag, 2001.

[21] Takahiro Harada, Seiichi Koshizuka, and Yoichiro Kawaguchi. Smoothed particle hydrodynamics on GPUs. In *CGI '07: Proceedings of Computer Graphics International 2007*, 2007.

[22] Mark J. Harris, Greg Coombe, Thorsten Scheuermann, and Anselmo Lastra. Physically-based visual simulation on graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics Hardware*, pages 109–118. Eurographics Association, 2002.

[23] Jim Held, Jerry Bautista, and Sean Koehl. From a few cores to many: A tera-scale computing research overview. Technical report, Intel Corporation, 2006. Intel White Paper. http://download.intel.com/research/platform/terascale/terascale_overview_paper.pdf link validated Jan. 14 2008.

[24] Victoria Interrante, Daniel Kersten, David Brainard, Heinrich H. Buelthoff, James A. Ferwerda, and Pawan Sinha. How to cheat and get away with it: what computer graphics can learn from perceptual psychology. In *SIGGRAPH '99: ACM SIGGRAPH 99 Conference abstracts and applications*, pages 119–121, New York, NY, USA, 1999. ACM.

[25] Andreas Kolb and Nicolas Cuntz. Dynamic particle coupling for GPU-based fluid simulation. pages 722–727, 2005.

[26] Andreas Kolb, Lutz Latta, and Christof Rezk-Salama. Hardware-based simulation and collision detection for large particle systems. In *HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 123–131. ACM, 2004.

[27] Gabriel Kron. Tensorial analysis and equivalent circuits of elastic structures. *Journal of the Franklin Institute*, 238(6), 1944.

[28] Claude Lacoursière. *Ghosts and Machines: Regularized Variational Methods for Interactive Simulations of Multibodies with Dry Frictional Contacts.* PhD thesis, University of Umeå, 2007.

[29] A. L. Lang and Raymond L. Bisplinghoff. Some results of swept-back wing structural studies. *J. Aero. Sci.*, 18(11), 1951.

[30] Börje Langefors. Analysis of elastic structures by matrix transformation with special regard to semimonocoque structures. *J. Aero. Sci.*, 19(7), 1952.

[31] Shaofan Li and Wing Kam Liu. Meshfree and particle methods and their applications. *Applied Mechanics Reviews*, 55(1):1–34, 2002.

[32] Jonas Lindmark. Fracturable surface model for particle-based simulation. Master's thesis, University of Lindköping, To be published in spring 2008. LITH-ISY-EX–08/4083–SE.

[33] Tadeusz J. Liszka and Janusz Orkisz. The finite difference method at arbitrary irregular grids and its application in applied mechanics. *Comput. Struc.*, 11:83–95, 1980.

[34] Wing Kam Liu, Ted Belytschko, and Herman Chang. An arbitrary lagrangian-eulerian finite element method for path-dependent materials. *Comput. Methods Appl. Mech. Eng.*, 58:227–246, 1986.

[35] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *ACM Computer Graphics*, 21(4), 1987.

[36] Jerry B. Marion and Stephen T. Thornton. *Classical Dynamics of Particles and Systems.* Thomson Brooks/Cole, 5th edition, 2003.

[37] George Mather. *Foundations of Perception.* Psychology Press (UK), 2006.

[38] Joseph J. Monaghan. An introduction to SPH. *Computer Physics Communications*, 48:89–96, 1988.

[39] Keith W. Morton and David F. Mayers. *Numerical Solution of Partial Differential Equations.* Cambridge University Press, 2nd edition, 2005.

[40] Matthias Müller and Markus Gross. Interactive virtual materials. In *GI '04: Proceedings of Graphics Interface 2004*, pages 239–246, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2004. Canadian Human-Computer Communications Society.

[41] Matthias Müller, Richard Keiser, Andrew Nealen, Mark Pauly, Markus Gross, and Marc Alexa. Point based animation of elastic, plastic and melting objects. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 141–151, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.

[42] Matthias Müller, Leonard McMillan, Julie Dorsey, and Robert Jagnow. Real-time simulation of deformation and fracture of stiff materials. In *Proceedings of the Eurographic workshop on Computer animation and simulation*, pages 113–124, New York, NY, USA, 2001. Springer-Verlag New York, Inc.

[43] Matthias Müller, Simon Schirm, and Matthias Teschner. Interactive blood simulation for virtual surgery based on smoothed particle hydrodynamics. *Journal of Technology and Health Care*, Accepted 2003.

[44] Hubert Nguyen. *GPU Gems 3.* Addison Wesley, 2007.

[45] James F. O'Brien, Adam W. Bargteil, and Jessica K. Hodgins. Graphical modeling and animation of ductile fracture. In *Proceedings of ACM SIGGRAPH 2002*, pages 291–294. ACM Press/Addison-Wesley Publishing Co., 2002.

[46] James F. O'Brien and Jessica K. Hodgins. Graphical modeling and animation of brittle fracture. In *Proceedings of ACM SIGGRAPH 1999*, pages 137–146. ACM Press/Addison-Wesley Publishing Co., 1999.

[47] Daniel Organ, Mark Fleming, T. Terry, and Ted Belytschko. Continuous meshless approximations for nonconvex bodies by diffraction and transparency. *Computational Mechanics*, 18:1–11, 1996.

[48] Mark Pauly, Richard Keiser, Bart Adams, Philip Dutré, Markus Gross, and Leonidas J. Guibas. Meshless animation of fracturing solids. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 957–964, New York, NY, USA, 2005. ACM Press.

[49] Mark Pauly, Richard Keiser, Leif P. Kobbelt, and Markus Gross. Shape modeling with point-based geometry. *ACM Transactions on Graphics*, 22(3):641–650.

[50] William H. Press, Saul A. Tuekolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C++: The Art of Scientific Computing.* Cambridge University Press, 2nd edition, 2002.

[51] Walter Ritz. Über eine neue methode zur lösung gewisser variationsproblem der mathematischen physik. *Journal für die reine und angewandte Mathematik*, 135:1–61, 1908.

[52] Alf Samuelsson and Olgierd C. Zienkiewicz. History of the stiffness method. *Int. J. Numer. Meth. Engng*, 67:149–157, 2006.

[53] Martin Servin, Claude Lacoursière, and Niklas Melin. Interactive simulation of elastic deformable materials. In *Proceedings of SIGRAD Conference 2006 in Skövde, Sweden*, pages 22–32. Linköping University Electronic Press, Linköping, 2006.

[54] Loup Verlet. Computer experiments on classical fluids. *Phys. Rev.*, 159(98), July 1967.

[55] Martin Wicke, Matthias Teschner, and Markus Gross. CSG tree rendering of point-sampled objects. In *Proceedings of Pacific Graphics*, 2004.

Index