Dynamic volume deformation using surfels

Olafur Thor Gunnarsson

A dissertation submitted in partial fulfilment of the requirement for the award of the degree of MSc in Computer Game Technology

University of Abertay Dundee

School of Computing and Creative Technologies

October 2010

**University of Abertay Dundee**

Permission to copy

Author: Olafur Thor Gunnarsson

Title: Dynamic volume deformation using surfels

Qualification: MSc Computer Games Technology

Year: 2010

1. I certify that the above mentioned project is my original work
2. I agree that this dissertation may be reproduced, stored or transmitted, in any form and by any means without the written consent of the undersigned.

Signature …………………….........................

Date …………………………..

**Abstract**

**Acknowledgements**

Thanks to:

My beautiful wife, Hrafnhildur and our son, Mikael Máni. For their love and support throughout my studies.

**Table of contents**

[1. Introduction 1](#_Toc270670887)

[2. Literature Review 3](#_Toc270670888)

[2.1 Volume exterior 3](#_Toc270670889)

[2.1.1 Voxels 3](#_Toc270670890)

[2.1.2 Tetrahedron 4](#_Toc270670891)

[2.1.3 Surfels 4](#_Toc270670892)

[2.2 Volume interior 5](#_Toc270670893)

[2.2.1 Mass-spring system 6](#_Toc270670894)

[7. References 7](#_Toc270670895)

[8. Image references 8](#_Toc270670896)

[9. Bibliography 9](#_Toc270670897)

**List of tables**

[Table 1 – Material properties of various real world object 12](#_Toc270837713)

**List of figures**

[Figure 1 - Fallout 3 nuclear weapon launcher (Sigger 2008) 1](file:///C:\Dropbox\My%20Dropbox\Skoli\Programming\Masters\Dissertation%20-%20Olafur%20Thor%20Gunnarsson%20-%200900128.docx#_Toc270766342)

[Figure 2 - 15 Cube triangulations (Lorensen and Cline 1987, p165) 3](file:///C:\Dropbox\My%20Dropbox\Skoli\Programming\Masters\Dissertation%20-%20Olafur%20Thor%20Gunnarsson%20-%200900128.docx#_Toc270766343)

[Figure 3 - Voxel sphere of resolution 20, 40 and 80 respectively (Author’s student project) 3](file:///C:\Dropbox\My%20Dropbox\Skoli\Programming\Masters\Dissertation%20-%20Olafur%20Thor%20Gunnarsson%20-%200900128.docx#_Toc270766344)

[Figure 4 - Tetrahedral mesh. Left: Only external triangles shown. Right: Internal structure shown (O'Brien 2000, p.31) 4](file:///C:\Dropbox\My%20Dropbox\Skoli\Programming\Masters\Dissertation%20-%20Olafur%20Thor%20Gunnarsson%20-%200900128.docx#_Toc270766345)

[Figure 5 - A tetrahedron (Kunkel 2004b) 4](file:///C:\Dropbox\My%20Dropbox\Skoli\Programming\Masters\Dissertation%20-%20Olafur%20Thor%20Gunnarsson%20-%200900128.docx#_Toc270766346)

[Figure 6 –A surfel 5](file:///C:\Dropbox\My%20Dropbox\Skoli\Programming\Masters\Dissertation%20-%20Olafur%20Thor%20Gunnarsson%20-%200900128.docx#_Toc270766347)

[Figure 7 - Elliptical surfels covering a smooth 3D surface. The surfels draw over the adjacent surfels to cover up holes (Pajarola, Sainz And Guidotti 2004, p 599) 5](file:///C:\Dropbox\My%20Dropbox\Skoli\Programming\Masters\Dissertation%20-%20Olafur%20Thor%20Gunnarsson%20-%200900128.docx#_Toc270766348)

[Figure 8 – 3D model of Charlemagne (600.000 points) with 2.000, 10.000, 70.000 and 600.000 surfels respectively (Gross and Pfister 2007, p 133) 5](file:///C:\Dropbox\My%20Dropbox\Skoli\Programming\Masters\Dissertation%20-%20Olafur%20Thor%20Gunnarsson%20-%200900128.docx#_Toc270766349)

[Figure 9 - Hooke's law (Gross and Pfister 2007, p343) 8](file:///C:\Dropbox\My%20Dropbox\Skoli\Programming\Masters\Dissertation%20-%20Olafur%20Thor%20Gunnarsson%20-%200900128.docx#_Toc270766350)

# Introduction

Computer game realism has improved tremendously since the days of Doom and Wolfenstein 3D. Bill Gates said in a promotional video for Windows ’95 as he was superimposed into Doom 2: “These games are getting really realistic” (Matthewandrewtaylor 2006). Fifteen years later this statement is still true, games are getting really realistic. But what is video game realism? Is it the visual quality of the game world, or is it the behaviour of the world as the player expects it to be? Video game realism is a combination of the two, it needs to look well and behave properly. There is no reason for creating the most real looking video game when there is no logic to the world behaviour.

The game world behaviour has many layers, from physics to night and day cycles. The behaviour that will be covered in this project is the volumetric deformation of walls. This feature is more often than not excluded from modern big budget games. Fallout 3 (Bethesda Game Studios 2008) is a perfect example of this discrepancy. In this game the player is given a rocket launcher that fires mini nuclear weapons. While this weapon works wonders on killing mutants and any other computer generated villain, it does no damage to structures. In a computer generated world that is so rich of interesting characters and beautiful graphics, this omission makes the game world feel like its lacking something. The behaviour of the game world therefore plays an important role in maintaining the user immersion.



Figure - Fallout 3 mini nuclear weapon launcher (Sigger 2008)

A few reasons exist why this is not incorporated into every 3D realistic game. The first one is because static environments simplify the game development by a large margin. In a game world where structures are static, a single wall’s vertices can be pre-calculated and placed into a vertex buffer that will not change throughout the game. This vertex buffer is in turn used when the wall needs to be drawn. If the wall is dynamic and destructible there are many systems that need to be updated. First of all there needs to be an internal mechanism that controls how the wall deforms, which needs to be updated every time a force acts on the wall. In addition to the interior, the exterior needs updating so the user can see the wall deform. And since the exterior changes, the collision detection system needs to be updated, so collisions continue to be realistic. In addition to these steps, the deformation needs to be realistic, which is something which can be hard to do.

This project will aim to find a realistic and yet efficient method of deforming 3D volumes dynamically in video games, such as metal walls. The methods used in this project are a combination of surface and physical elements (surfels and phyxels respectively). Where the surfels are used to simulate the exterior of the model, but the phyxels simulate the interior.

# Literature Review

This chapter is dedicated to the immense research and knowledge that exists in the field of volumetric deformation. Each method will be summarized and discussed with special focus on this project.

## Volume exterior

This sub-chapter will cover the various mesh-free methods for surface visualization.

### Voxels

The Marching Cubes algorithm was introduced in 1987 by Lorensen and Cline for 3D visualization of medical data, such as magnetic resonance (MR) and computed tomography (CT) scans. This algorithm takes eight voxels (points) that make up a cube and checks if the eight points are inside or outside of the model. Based on the states of the points, the algorithm approximates the surface by placing triangles inside the cube. Since each point can be in two states the triangulation possibilities are 28 = 256. Through intuitive thinking, Lorensen and Cline reduced the number of possible triangulation from 256 down to 15 which can be seen in Figure 2. This reduction comes from the use of inverses and rotation. As can be seen from case 0 no triangulation occurs when all or none of the points are defined as “inside”. Alternatively, a single triangle is inserted when a single point is defined as “inside“ or “outside” as case 1 depicts. When a single cube has been triangulated, the algorithm marches to the next cube (Lorensen and Cline 1987).

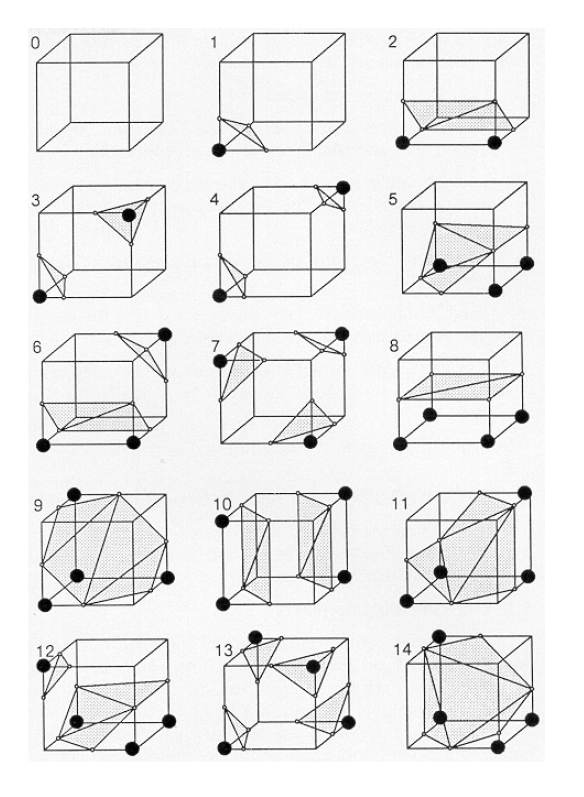


Figure - 15 Cube triangulations (Lorensen and Cline 1987, p165)

However, this method does have its downside because the point set needs to be dense enough so the volume does not look triangulated. In Figure 3 three voxel spheres were created using the marching cube algorithm. These spheres are all of equal size, but vary in their resolution and voxel size. The leftmost sphere is 20 voxels in height, depth and width and has a voxel size of 1. This results in a very course representation of the sphere. When the resolution becomes higher than 40, the voxelized object starts to resemble a sphere. At this resolution the sample point count is 64000. As a reference the sphere of resolution 80 has a sample point count of 512000. It is therefore apparent that a detailed model will need a lot of sample points. Because of this reason, this method was not chosen. Also when the volume is deformed, the marching cube algorithm needs to be run again. If the point set is very large, the marching cube algorithm takes a while to finish.



Figure - Voxel sphere of resolution 20, 40 and 80 respectively (Author’s student project)

### Tetrahedrons

Just like triangles can represent any surface, tetrahedrons can represent any volume (O’Brien 2000). A tetrahedron is defined as four vertices and four triangles, where three triangles meet at each vertex (Kunkel 2004). The reason why tetrahedrons are useful when it comes to volumetric deformation is because tetrahedrons provide both the surface representation as well as the volumetric representation. The triangles of the tetrahedron that are on the boundary of the mesh are drawn to display the surface of the mesh. The tetrahedron’s vertices can be used as the volume’s physical elements, where the edges between the vertices can be used to simulate the volumes internal structure. The single most importance of tetrahedrons, when it comes to deformation, is the fact that when a tetrahedron is split by any arbitrary plane, the resulting pieces can be decomposed into more tetrahedrons (O’Brien 2000). This plays an important part in deformation since it guarantees a water tight structure. Tetrahedrons were considered as a surface and volume representation for this project, but because of the fact that a lot of tetrahedrons are needed to produce realistic effects, this method was not chosen.

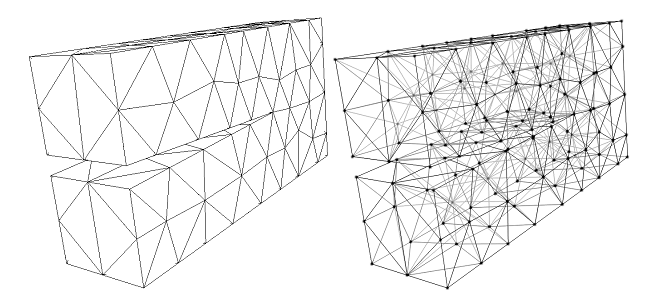


Figure - Tetrahedral mesh. Left: Only external triangles shown. Right: Internal structure shown (O'Brien 2000, p.31)

### Surfels

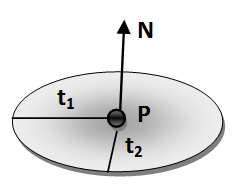


Figure –A surfel

Over the last decade, research on point based graphics has been ever increasing. Surface elements (surfels) have become very popular when it comes to point based graphics, mainly because of their efficiency to approximate surface (Gross and Pfister 2007). A surfel is defined as a point with a normal and two tangent axes that define an ellipsoidal plate. This plate can in addition contain surface information, such as texture coordinates or colour. Figure 5 shows an image of a surfel with normal **N**, two tangents **t1** and **t2** and position **P**. The tangents are tangent to each other as well as to the normal.

A surfel based surface is created by splatting surfels onto a approximated surface based on a set of input points. If the point set is sufficiently dense and the surfels are correctly set up, the surfel surface can resemble the expected model (Figure 6). It is the author’s opinion that this method is more efficient than using voxels since only a set of exterior points is used to approximate the surface. However, this method does have its drawbacks. Firstly this method has considerable overdraw, because of the way surfels are used to create a watertight surface. It is apparent from Figure 7 that surfel based rendering techniques do draw over pixels that have been drawn already. Secondly, edges and corners pose a problem with surfels, since they are elliptical plates. One solution to the edge problem is to detect intersecting surfels and divide into smaller surfels until the intersection is below a certain threshold (Adams and Dutré, 2003). This method does however increase the number of surfels by a large margin and might result in visual defects if the structure is looked at closely. Another solution has been to use an algorithm that goes through the point set and reconstructs the 3D object based on these points. This solution is very popular for systems that receive a large set of points obtained by a laser scanner, but is beyond the scope of this thesis. Those interested in such reconstruction algorithms are pointed to a paper by Öztireli, Guennebaud and Gross (2009).



Figure - 3D model of Charlemagne (600.000 points) with 2.000, 10.000, 70.000 and 600.000 surfels respectively (Gross and Pfister 2007, p 133)

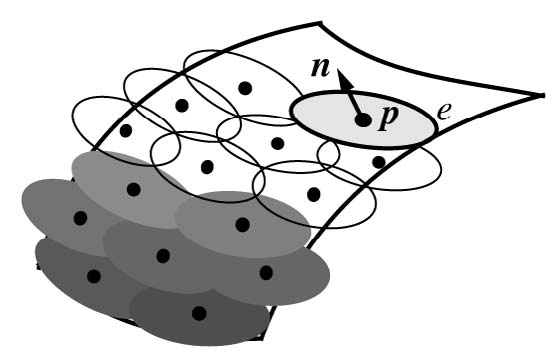


Figure - Elliptical surfels covering a smooth 3D surface. The surfels draw over the adjacent surfels to cover up holes (Pajarola, Sainz And Guidotti 2004, p 599)

## Volume interior

For volumetric deformation to be applied to a volume, the volume needs to be converted into a form that is easy to use in computer graphics. There exist a number of different methods to represent how force is distributed through the volume, which will be covered in this sub-chapter.

### Mass-spring system

The simplest way to simulate deformable objects in computer graphics is to use a mass-spring based system. It is set up by using particles with masses that are connected by springs (Müller et al. 2008). In the simplest mass-spring systems the points have only position and velocity attributes. It can also be extended by adding acceleration to the points. With these attributes it is possible to add forces to the system and distribute them through the model. Appendix A shows a short pseudo code to emphasize the simplicity of mass-spring systems.

This method does however have its drawbacks. The first problem is that the deformable behaviour of the object is very dependent on how the mass-spring system is set up. For example if a wall is implemented with a mass spring system, the mass of the points as well as the springs stiffness and damping constants need to be correctly set up so the correct behaviour is simulated. It is therefore inevitable to encounter some variable fiddling. The final drawback is due to this system being only stable for relatively small time steps, from 0.1ms – 1ms (Müller et al. 2008).

### Phyxels

Another particle based volumetric representation is the physical element (phyxels) systems. Similar to mass spring particles, phyxels contain body location (***x***) and force (***f***). Additionally, phyxels contain attributes that model density (), deformation (***u***), volume (*v*), strain () and stress () (Müller et al. 2004). With these additional attributes the model’s material can be simulated relatively realistically using Hook’s law (see chapter 2.3). Figure 8 shows a modelled head of the German physicist Max Planck, simulated by surfels and phyxels. The leftmost image depicts the Max Planck model, where the surfels are scaled down so the internal structure is showing. In this image the phyxels are coloured yellow and the surfels are blue.



Figure - Max Planck represented with surfels and phyxels. Two leftmost heads are undeformed. The four right heads are deformed elastically, plastically, melted and solidified (respectively) (Müller et al. 2004, p 356)

Instead of springs between phyxels, the phyxels incorporate a weighing function () which is a scalar-kernel and is radially symmetric. To implement this function the phyxels need to have a support radius, that is the phyxel’s area of effect. Müller et al. (2004) defined the weighing function as , where *r* is the distance from the position of the phyxel and *h* is the phyxel’s support radius. This weighing function can be used to calculate how much of a force is distributed from one phyxel to its neighbours. Another functionality of the weighing function is to calculate the phyxel’s density , where (Gross and Pfister 2008). This volumetric representation was chosen as the project’s representation because this method does provide a reasonably realistic representation of volumetric deformations. Because of this, the phyxel method will be used throughout this thesis as the interior representation of the volumes.

## Continuum Mechanics

In continuum mechanics a single law, created by Robert Hooke in the 17th century, has been extensively used to represent deformation of 3D computer generated objects. Hooke’s law can be used on so called Hookean materials, which are materials where stress and strain are linearly related. Figure 9 shows a beam with a cross section that has an area *A*. Force ***f****n* is applied perpendicular to the area, resulting in an elongation of the beam by ∆*l*. The beam’s stress  is defined as and therefore has the unit . The strain  however has no unit as it is defined as . The strain to stress ratio does depend on *E*, called the Young’s modulus which represents the elastic stiffness of the material. Various Young’s modulus’s for a variety of real world materials can be seen in Table 1 in Appendix B.



Figure - Hooke's law (Gross and Pfister 2007, p343)

The object in Figure 9 is only deformable in one dimension, along the force and therefore its stress and strain variables are one dimensional (1D). However if the object is deformable in 3D, these variables get more complicated. According to Müller et al. (2008) and Gross and Pfister (2007), 3D materials that behave equally in every direction can express the stress and strain variables as 3x3 symmetric matrices (or tensors):

|  |  |  |  |
| --- | --- | --- | --- |
|  | ( .) |  | ( .) |

Since the stress and strain tensors are symmetric, they only contain 6 individual variables. Because of this, the stiffness modulus can be written as a 6x6 symmetric matrix:

|  |  |
| --- | --- |
|  | ( .) |

In equation (1.3) *E* is the Young’s modulus and *v* called Poisson’s ration, which describes how much of the volume is conserved within the material and is limited between 0 and .

When a deformation object is represented in computer graphics, it is typically defined by two states: its undeformed state and by a set of parameters that define its deformed state. As forces are applied to phyxels, the material coordinate (***x*** *=* [*x, y, z*]T) is moved to its world coordinates (***p****(****x****)**=* [*x’, y’, z’*]T), both of which are defined as vector fields. The deformation of the object can be defined by a displacement field . This displacement can be seen in Figure 10 where the undeformed position is moved to its deformed position by applying the material properties.

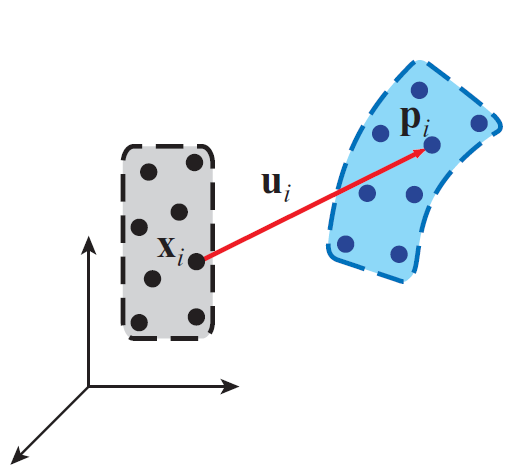


Figure - Object's undeformed state (left) and deformed state (right) (Gross and Pfister 2007, p345)

When updating the stress and strain a few things need to be calculated. Firstly a moment matrix (also known as a system matrix) is used to calculate the displacement derivative of each phyxel. This moment matrix is defined in equation 2.4 and the derivative of the displacement matrix is defined in equation 2.5:

|  |  |
| --- | --- |
|  | (2.4) |
|  | (2.5) |

This displacement field is used to update the strain, which can in turn be used to update the stress. Each spatial derivative of the displacement field is calculated by:

|  |  |
| --- | --- |
|  | (2.6) |
|  | (2.7) |
|  | (2.8) |

where *u* is defined as the displacement along the x-axis, *v* is the displacement along the y-axis and *w* is along the z-axis. Using the displacement derivative the strain can be updated with the Green’s nonlinear strain tensor:

|  |  |
| --- | --- |
|  | (2.9) |

Finally, a single matrix (***B***) is used to distribute internal forces throughout the volume:

|  |  |
| --- | --- |
|  | (2.10) |

The following pseudo code shows how a complete system of phyxels is initialized and updated:

**forall** phyxel i

initialize i,i = 0, = 0, , i, ;

calculate using equation (2.4);

**endfor**

**loop**

**forall** phyxels i **do endfor**

**forall** phyxels i **do**

calculate and using equations (2.6, 2.7 and 2.8);

calculate using equation (2.9);

calculate using equation (2.3);

calculate using equation (2.10);

**forall** neighboring phyxels j

;

**endfor**

**endfor**

**forall** phyxels i

; **;**

**endfor**

render system at ;

**endloop**

# Appendices

## Appendix A

Pseudo code for a simple mass-spring system (Müller et al. 2008)

// initialization

**forall** particles i

initialize **x**i, **v**i and mi

**endfor**

// simulation loop

**loop**

**forall** particles i

(Eq. A.5)

**endfor**

**forall** particles i

(Eq. A.7)

(Eq. A.8)

**endfor**

display the system every nth time

**endloop**

Before the simulation begins each particle is initialized with a position (**x**), velocity (**v**) and mass (*m*). During each simulation step the gravity force (***f****g*) and collision force (***f****coll*) are accumulated to the force of the particle. The particle’s spring forces are calculated from its adjacent particles (S) in equations A.1 and A.2.

(A.1)

(A.2)

Here attributes with the *ith* and *jth* indices represent adjacent particles *i* and *j*, *k****­****s* stands for the stiffness of the spring and *l0* represents the initial length between the two particles. Similarly the damping force between the two particles needs to be calculated using equations A.3 and A.4:

where *k*d represents the spring’s damping constant. If equations A.1 and A.3 are combined, the complete spring force is:

(A.5)

(A.3)

(A.4)

Finally when all forces have been calculated for each particle and its adjacent particles, the particles need to be moved. By combining Newton’s second law of motion and two kinematic equations, the equations for updating the particle’s velocity and position are constructed:

(A.6)

(A.7)

(A.8)

## Appendix B

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Material | Young’s Modulus | Poisson’s ratio |  |  |
| Clay |  |  |  |  |
| Concrete |  |  |  |  |
| Steel |  |  |  |  |

Table – Material properties of various real world objects

# References

Adams, B. and Dutré, P. 2003. Interactive Boolean operations on surfel-bounded solids*.* In: *International Conference on Computer Graphics and Interactive Techniques, San Diego, 27-31 July*. New York: ACM. pp 651-656. [Online]. Available from: <http://portal.acm.org/citation.cfm?id=1201775.882320> [Accessed 1 August 2010]

Bethesda Game Studios. 2008. *Fallout 3.* [Disk]. Windows XP. Ubisoft.

Gross, M. and Pfister, H. eds. 2007. *Point based graphics.* Burlington, MA: Morgan Kaufman.

Lorensen, W.E. and Cline H.E. 1987. Marching cubes: a high resolution 3D surface construction algorithm. *Computer Graphics.* 21(4), pp. 163 – 169

Kunkel, P. 2004a. *The Tetrahedron.* [Online]. Available from: <http://whistleralley.com/polyhedra/tetrahedron.htm> [Accessed 27 August 2010]

Matthewandrewtaylor. 2006. *Bill Gates in Doom 2*. [Online]. YouTube. Available from: <http://www.youtube.com/watch?v=xh0JM7pD4qM> [Accessed 24 August 2010]

O’Brien, J.F. 2000. *Graphical Modeling and Animation of Fracture*. [M.Phil. thesis]. Georgia Institute of Technology.

O’Brien, J.F., Bargteil, A. W. and Hodgins, J.K. 2002. Graphical modelingand animation of ductile fracture. *ACM Transactiona on Graphics (TOC).* 21(3), pp 291-294.

Müller, M. et al. 2004. Point based animation of elastic, plastic and melting objects. In: *Proceedings of the 2004 ACM SIGGRAPH, Grenoble 27 – 29 August 2004*. New York: ACM. pp 141-151.

Müller, M. et al. 2008. Real time physics: class notes. *International Conference on Computer Graphics and Interactive Techniques, Los Angeles, 11 – 15 August 2008*. New York: ACM

Nealen, A. et al. 2006. Physically based deformable models in computer graphics. *Computer Graphics Forum.* 25(4), pp. 809 – 836.

Pfister, H., et al. 2000. Surfels: surface elements as rendering primitive. In: *Proceedings of the 27th annual conference on Computer graphics and interactive techniques.* New York: ACM. pp. 335 – 342. [Online]. Available from: <http://portal.acm.org/citation.cfm?id=344779.344936> [Accessed 21 April 2010]

Pajarola, R., Sainz, M. And Guidotti, P. 2004. Confetti: Object-Space Point Blending and Splatting. *IEEE transactions on visualization and computer graphics.* 10 (5), pp 598 – 608

Öztireli, A. C., Gennebaud, G., And Gross. M. 2009. Feature preserving Point set surfaces based on non-linear kernel regression. In: *Proceedings of Eurographics, Munich, 30 March – 3 April.* 28(2), pp. 493-501.

# Image references

Sigger, J. 2008. *No fat man for you!*. [Online image]. Available from: <http://armchairgeneralist.typepad.com/my_weblog/2008/11/no-fat-man-for.html> [Accessed 24 August 2010]

# Bibliography