JACOBS UNIVERSITY BREMEN

Detail-Preserving Animation

by

Gordan Ristovski

Guided Research Report
Supervised by: Prof. Dr. Lars Linsen

in the
School of Engineering and Science
Computer Science

May 2009

JACOBS UNIVERSITY BREMEN

Abstract

Computer Science
School of Engineering and Science

Guided Research Thesis

by Gordan Ristovski

Physics based deformable models have two decades of history in Computer Graphics and physics-based simulation has gained increasing importance in many fields of computer graphics, including 3D game engines, computer animation for feature films, surgery simulation, and virtual reality. But in many of them, precious details are lost over the course of object deformation. The proposed project aims to speed up the deformation process and save the details by removing them before the modifications and thus operating on objects with extremely simple meshes and returning them after the distortion.

Acknowledgements

I would like to express my gratitude to my supervisor Prof. Dr. Lars Linsen for making me passionate for topics from Graphics and Visualization as well as for the help, stimulating suggestions, encouragement and great guidance during the whole guided research.

Contents

Abstract i

Acknowledgements ii

List of Figures iv

1 Introduction 1
- 1.1 Motivation and Problem Statement . . . 1
- 1.2 Related Work . . . 1
- 1.3 Wrap up of Available Technologies . . . 3

2 Application Workflow 4
- 2.1 Object Choice and Surface Sampling . . . 4
- 2.2 Inside Point Sampling . . . 5
- 2.3 Physics Modeling . . . 9
- 2.4 Removing and Storing Details . . . 10
- 2.5 Return the Details . . . 13
- 2.6 GUI description . . . 14

3 Conclusion 16
- 3.1 Discussion . . . 16
- 3.2 Future Work . . . 16
  - 3.2.1 Further Improvements . . . 16
  - 3.2.2 Future Expansions . . . 17

Bibliography 19

List of Figures

1.1 Simple Deformation Process . . . 2

2.1 Model Cube . . . 5
2.2 Ray-Triangle Intersection . . . 7
2.3 Ray Intersection Border Cases . . . 8
2.4 Special Ray Penetration Cases . . . 8
2.5 Sampled Cube . . . 9
2.6 Making the Mesh Coarser . . . 10
2.7 Barycentric Coordinates . . . 12
2.8 Recursive Storing/Removing Details . . . 12
2.9 Less Detailed Cube . . . 13
2.10 Deformation Process . . . 14
2.11 Graphical User Interface . . . 15

3.1 Extreme Deformation . . . 17

Chapter 1

Introduction

1.1 Motivation and Problem Statement

Physics-based simulations are extremely important in many aspects such as gaming and film industry for creating 3D engines and special effects, and they are also becoming widely used in medicine. For 3D graphics visualization applications, objects and models are often represented using complex meshes in order to maintain a convincing level of realism. The cost of deforming such high resolution models can be significant.

Moreover, most methods for simulation of deformable objects lose the surface details after pressure has been applied. But elastic objects tend to return to their equilibrium position after the modifications, so whatever forces act on them, the details must be preserved and returned. A simple detail preservation is shown in Figure 1.1.

The approach presented solves these problems via multiresolution modeling [1] [2]. Namely, before the deformation the details are removed from the surface. It is then operated on a low detailed model and after the deformation, the details are mapped back to the body, thus avoiding expensive computations while keeping all the fine points. With this, not only is the overall process less expensive since the surface may become as simple as containing only the end points, but when the details are returned the model matches the expected form after such deformation, as discussed in Section 3.1.

1.2 Related Work

Many of the existing mesh-based methods for the simulation of deformable objects include [3]:

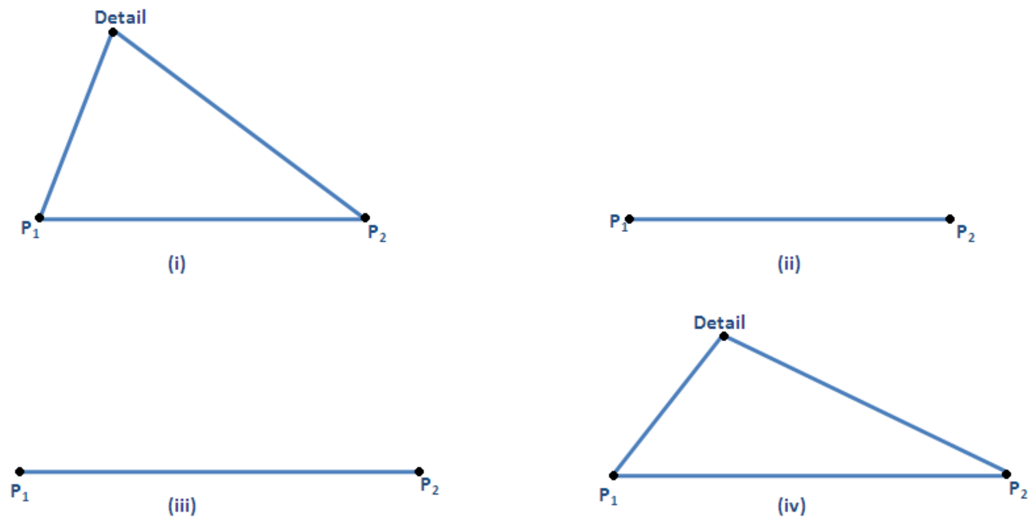

FIGURE 1.1: A general simple deformation process. From left to right, top to bottom: **(i)** Original Triangle, **(ii)** A detail has been removed. **(iii)** Line P_1P_2 is stretched. **(iv)** The detail is returned, but with a different position and distance then before. The height change of the detail is inversely proportional to P_1P_2's change of width.

- The Finite Element Method (FEM), where the object is viewed as a continuous connected volume which is discretized using an irregular mesh and then Partial Differential Equations obtained from Continuum Mechanics are solved.
- The Finite Differences Method, where the objet is sampled using a regular spatial grid and then certain equations are discretized using finite differences.
- The Finite Volume Method, where the forces acting on the nodes of an element are computed as the derivatives of the deformation energy with respect to the nodal positions.

However, in mesh-based approaches, complex physical effects such as melting, solidifying, splitting or fusion pose great challenges in terms of restructuring. Additionally, under large deformations the original meshes may become arbitrarily ill-conditioned.

Thus, the combination of mesh-free physics with point-sampled surfaces [4] in so-called point-based animations have become popular. Müller et al. introduced to the graphics community a mesh-free continuum mechanics based framework for the animation of elastic, plastic and melting objects [5]. However, the surface animation presented is very costly and not very accurate. Namely, they start their model with a detailed surface (L_D) made of splat-like 2D elements (surfels). Underneath this, there is implicit surface (L_I) made up of the topmost layer of phyxels. It can only represent blobby exteriors,

and as the phyxels move, the surfels move too, so the detailed surface we see approaches L_I every step. Thus, we lose precious details, and the computations increase the overall complexity.

The framework presented by Müller at al. [5] is extended in [6] to cope with fracturing. Cracks are created at surfels where the main principal stress exceeds the threshold.

Horman et al. developed a way for replacing an arbitrary triangle mesh by a regular quadrilateral mesh [7], because a lot of algorithms require the mesh to be structured in a certain way and cannot be applied to an arbitrary shaped mesh. Chen et al. use this method to parametrize the original triangle mesh to a regular quadrilateral approximation [8], and then apply a wavelet transform to remove a large amount of correlation between neighboring vertices.

1.3 Wrap up of Available Technologies

- **OpenGL:**

  OpenGL (Open Graphics Library) is a standard specification defining a cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics. OpenGL was developed by Silicon Graphics Inc. and is widely used in CAD, virtual reality, scientific visualization, information visualization, and flight simulation.

- **Qt Creator:**

  Qt Creator is a complete integrated development environment (IDE) for creating applications with the Qt application framework. Qt is designed for developing applications and user interfaces once and deploying them across several desktop and mobile operating systems.

Chapter 2

Application Workflow

This section elaborates on how the algorithm for detail preservation works step by step. It explains the model choice, surface and volume sampling of the model, the removal of its details, the restoration of its details and finally it describes the interactive GUI designed.

2.1 Object Choice and Surface Sampling

The object used to model the deformation and detail preservation was chosen such that its mesh had simple yet very detailed initial structure - it is a cube with every face having wavy surface described by

$$z(x, y) = sin(x)sin(y) \tag{2.1}$$

using regular quadrilateral mesh. The points have big enough range, and the frequency is given as a global parameter which can be changed depending on how detailed the user wants the surface to be. The quadrilaterals which are adjacent to the edges of the cube are flat which guarantees consistent model without cracks or holes. Each quadrilateral's normal is calculated and passed to OpenGL for rendering. A screenshot can be seen on Figure 2.1.

As described in the related work, there is an algorithm that computes successive adaptive regular quadrilateral approximations from an arbitrary input triangle mesh. Thus, the choice of a quadrilateral mesh does not incur extreme limitations.

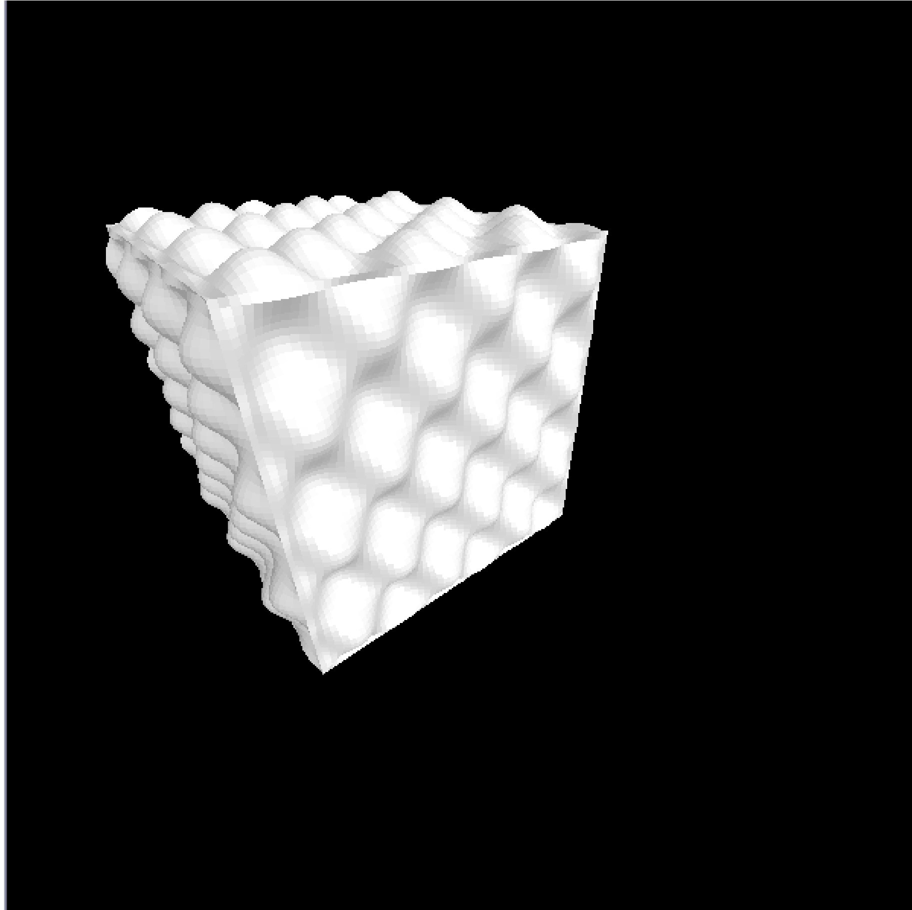

FIGURE 2.1: The object used throughout to show deformations, as rendered by OpenGL

2.2 Inside Point Sampling

For a realistic deformation, the volume needs to be sampled. The points that were determined to be inside the volume by the algorithm may then be passed to any physics simulation method and may be displaced according to the forces acting on the body.

The point sampling algorithm works as follows:

The sampled points are chosen from the view frustum in regular intervals with constant distance between them. This ensures that they are not randomized which may result in many points clustered at one place and consequently unrealistic deformation. Next, a ray is shot from a constant point (eye) outside of the frustum to each of the points. If this ray intersects our model odd number of times, then the point is inside the object, otherwise, it is outside.

To count the number of intersections, each quadrilateral is divided along the diagonal so it forms two triangles and for each triangle it is determined wether it is intersected

by the ray or not. For this, the intersection point of the ray and the plane in which the triangle lies is calculated, and then the algorithm checks if this point is inside the triangle or not. Let T_1, T_2 and T_3 be the vertices of the triangle (part of the mesh), E be the starting point for all the rays (the metaphoric eye), and P be the point we are sampling. Moreover, let I be the ray-plane intersection, as represented in Figure 2.2 which is calculated as follows:

- The normal of the plane N is the cross product of two vectors with a common starting point lying on it. As 3 points on the plane are already known (the vertices of our triangle), the normal is easily calculated by:

$$N = (T_2 - T_1) \times (T_3 - T_1) \tag{2.2}$$

- The equation of the plane is given by $d = p \cdot N$. As the normal and three points on the plane are known, we choose any of them, and calculate d by

$$d = T_1 \cdot N \tag{2.3}$$

- Finally, I is computed as the point at distance t along the vector $\overrightarrow{EP}$ with E being the initial point.

$$I = E + (P - E) \cdot t \tag{2.4}$$

  The only thing left is to find t. But this is not a problem because we know that I is a point on the plane, so the general plane equation holds for it ($I \cdot N = d$). If this is combined with Equation 2.3, we get the following

$$t = \frac{d - E \cdot N}{(P - E) \cdot N} \tag{2.5}$$

- When this result is substituted in Equation 2.4 we get the coordinates for I.

After the intersection point has been found, the algorithm checks if it is inside the triangle, meaning that the ray intersects the model, or not. If the point is in the triangle, then it is on the same side of the line passing through any combination of two points from the vertices as the third vertex. Without loss of generality, consider the line passing through T_1 and T_2. If I is on the same side of the line as T_3 then the cross products of $\overrightarrow{T_1T_2}$ with $\overrightarrow{T_1I}$ and with $\overrightarrow{T_1T_3}$ have the same direction. Namely, if

$$((T_2 - T_1) \times (I - T_1)) \cdot ((T_2 - T_1) \times (T_3 - T_1)) < 0 \tag{2.6}$$

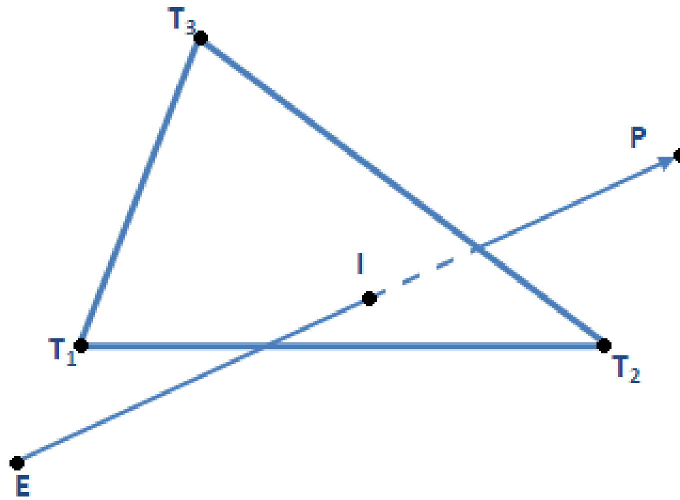

FIGURE 2.2: The ray $\overrightarrow{EP}$ intersects the plane defined by the triangle with vertices T_1, T_2 and T_3 at point I. It then has to be determined if I is in the triangle or not.

then the point is not in the triangle. Otherwise, it has to be checked for the other two lines.

Eventually, the ray penetrates the model only if I is between E and P. This is checked by simple comparison of their distances - if $||\overrightarrow{EI}|| < ||\overrightarrow{EP}||$ then the ray intersects the object, otherwise, it is too short.

The next challenge is the case when the point is found to be inside a triangle, but one or two of the dot products calculated by Equation 2.6 is zero. In this case, the ray intersects or touches the triangle's edge or vertex respectively, which is a very common case as the cube's faces are wavy and uneven. Figure 2.3 a) shows the first case where the ray penetrates the model, and these occurrences should be counted in the total number of intersections, but with extreme caution that they are considered only once, and not once for every triangle that shares the edge or vertex being intersected. In the second case, shown in Figure 2.3 a, the ray touches the figure, but does not intersect it, so these occurrences should be counted out of the total number of intersections.

This can be solved by checking the ray's direction with respect to the normals of all the triangles it intercepts. Namely, the dot product is calculated between the ray ($\overrightarrow{EP}$) and each of the triangle's normals, calculated according to Equation 2.2. If all the dot products have the same sign, then the vector intersects the objects, otherwise, it just touches it.

An edge or a vertex in a mesh is generally shared among several faces. So, if the intersection point lies on a special location, it must be determined how many faces share

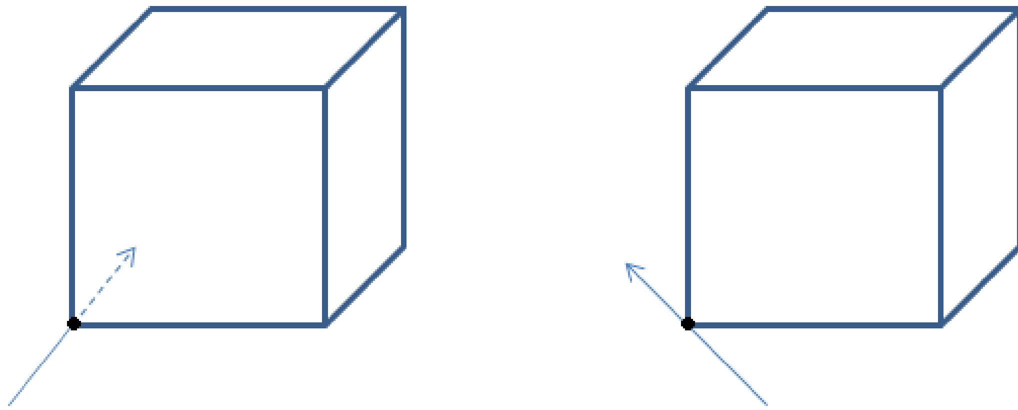

FIGURE 2.3: Border cases for ray-figure intersection: **(a)** ray penetrates the model, **(b)** ray touches the model

it, and this number will later be discounted from the total number of intersections. For my model, there are four cases in total with different number of faces sharing a common point, as shown in Figure 2.4.

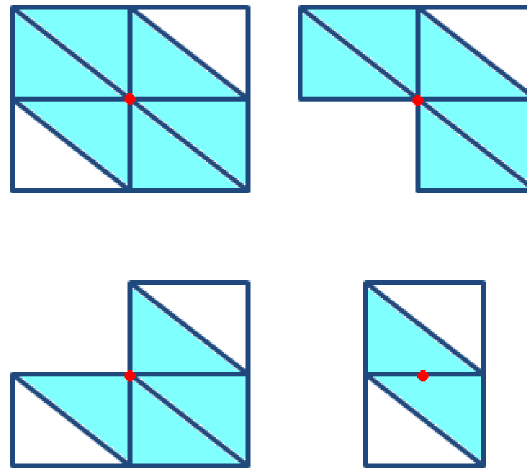

FIGURE 2.4: If the ray penetrates the cube, but the point is on triangle's borders there are four cases enumerated left-right, top-down: **(i)** The ray intersects an ordinary vertex, shared by four quadrilaterals. In this case 5 penetrations should be discounted. **(ii)** The ray intersects the cube at a point shared by 3 quadrilaterals only, for example, the bottom left corner of the front face. In this case 4 penetrations should be discounted. **(iii)** The ray intersects the cube at a point shared by 3 quadrilaterals only, for example the upper left corner of the front face. In this case 3 penetrations should be discounted. **(iv)** The ray intersects an edge. Since every edge is shared by only 2 quadrilaterals, only 1 penetration should be discounted here.

The result of the inside point sampling process can be seen on Figure 2.5. The red points here represent the surface samples, and the blue points make the volume of the

figure. The front surface was intentionally left unrendered, so the inside could be clearly visible.

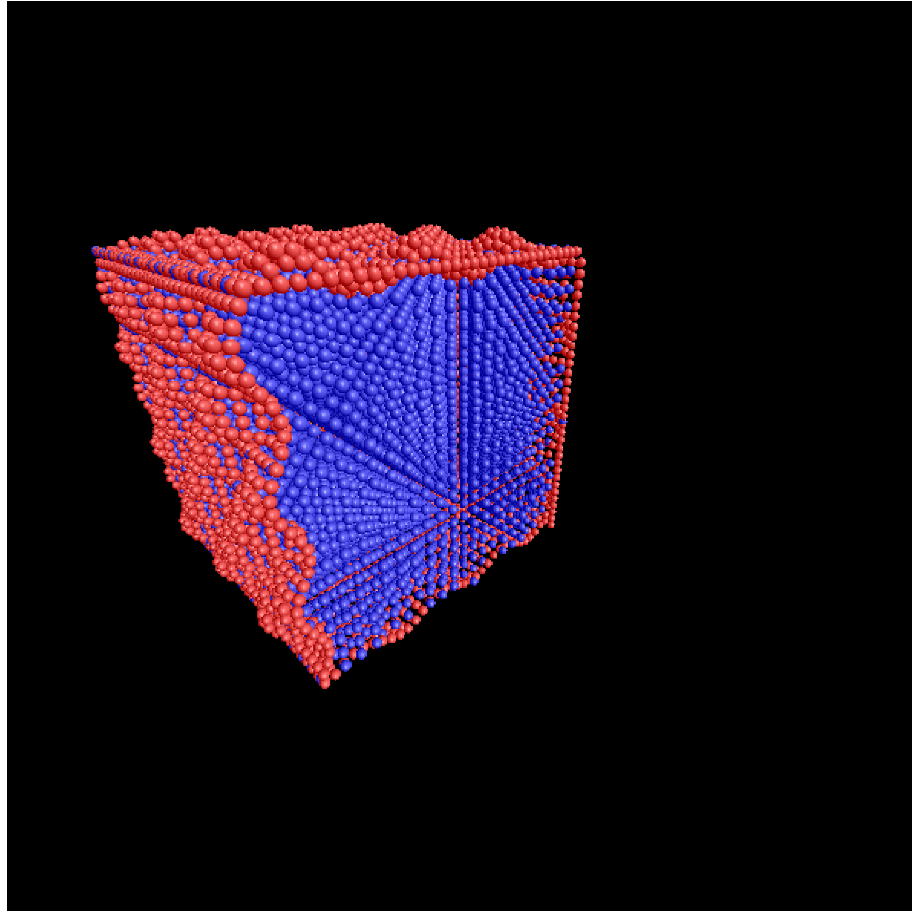

FIGURE 2.5: Inside Point Sampling of the model.

2.3 Physics Modeling

In the Related Work Section 1.2, I have listed many physics based deformable models. Once the process in the previous section is completed and all the inside points have been found, they can be passed to a physics simulator and displaced according to forces acting on the body (ranging from gravity to complex forces acting on the body in many directions to deform it). For example, if the points are initialized with certain weight, and a spring is inserted between neighbors, a deformation based on the mass-spring system is possible. If the points are passed without connectivity information, but with initial density, strain, body force and velocity, one of the many existing mesh free approaches to deformable modeling can be used. The displaced points realistically deform the surface, but the connectivity information is not lost, so it can be easily rerendered after the process is finished, thus obtaining realistic real-life deformation.

2.4 Removing and Storing Details

When a less detailed surface is created for a regular quadrilateral mesh, at each step every second point is removed in horizontal and vertical direction, as shown in Figure 2.6. This will basically result in decrease of the surface detail by half. But the detail information must be preserved so after the less detailed object has been deformed they could be added to it again. After the process is finished there should be no noticeable difference between deformations with the details on, or low-detail fast deformation where details are restored after the process has finished. A point to be removed has 3 other reference points with respect to which the information regarding details will be stored. They are chosen to be the left neighbor, right neighbor, and the middle of the upper left point closest to it and upper-right point closest to it that are not going to be removed. In Figure 2.6, the reference points for P are going to be C_1, C_2 and $C_m = (C_3 + C_4)/2$

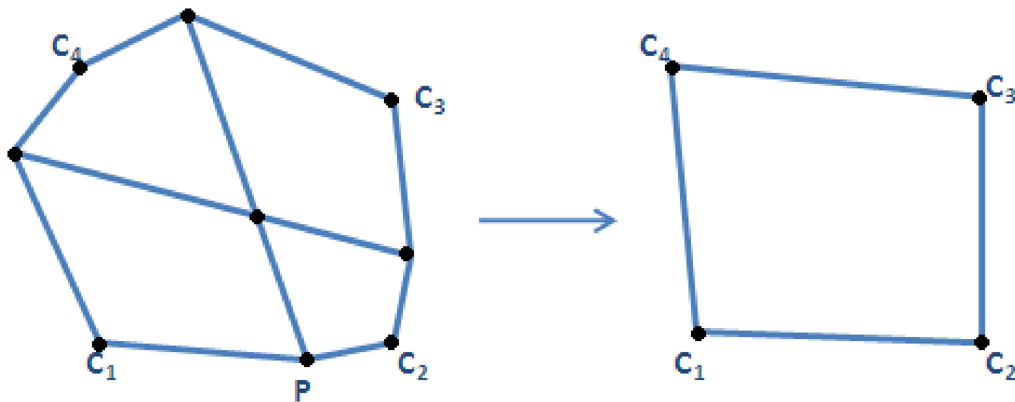

FIGURE 2.6: Removing the details in a quadrilateral mesh. On the left side is part of the original mesh, and on the right side is how the mesh will look like when one level of details is removed.

The algorithm for storing the details first projects the point to be removed to the plane defined by the three references that are not abstracted, meanwhile checking if the point was above or below the plane since the models are in 3D. It then calculates the barycentric coordinates of the projection point with respect to the references. The barycentric coordinates are extremely powerful for this problem because as the body deforms, the reference points will inevitably change, so when the details are to be restored, the new point's position on the plane will be recalculated according to its relative distance to the vertices of the distorted triangle.

The first part of the algorithm is as described in Section 2.2. If t is determined to be less then 0, then we are moving backwards along the vector, so the point is below the plane. Otherwise, it is above it.

Let P be the projection point on the plane defined by the reference points C_1, C_2 and C_m as shown in Figure 2.7. The barycentric coordinates of P with respect to the references are $[u, v, w]$ such that

$$P = uC_1 + vC_2 + wC_m, \text{ where } u + v + w = 1 \tag{2.7}$$

They can be calculated as the ratios of the areas formed by P and any two of the references to the area of the triangle formed by them. Therefore

$$u = A_1/A, \; v = A_2/A \text{ and } w = A_3/A \text{ where } A = A_1 + A_2 + A_3 \tag{2.8}$$

Since the points are in 3D, let the vector $\mathbf{m} = P_2 - P_1$, $\mathbf{n} = P_3 - P_1$ and $\mathbf{r} = \mathbf{m} \times \mathbf{n}$. The area of the parallelogram formed by $\mathbf{m}$ and $\mathbf{n}$ is

$$A = ||\mathbf{r}|| = |\mathbf{m}||\mathbf{n}|\sqrt{1 - (\hat{\mathbf{m}} \cdot \hat{\mathbf{n}})^2} \tag{2.9}$$

where $\hat{\mathbf{m}}$ and $\hat{\mathbf{n}}$ represent unit vectors obtained by dividing each vector by its magnitude. In this equation, all the areas give positive result, but the barycentric coordinates may as well be negative if the point is outside of the triangle. For this, the concept of signed areas is introduced. When the vectors $\mathbf{m}$ and $\mathbf{n}$ are calculated, the points must be passed in the same direction (clockwise or counterclockwise) as if the projection point was inside the triangle. Thus if it is inside as in Figure 2.7, all cross products $\mathbf{r}$ will have the same direction. Otherwise, one or more of them will differ hence resulting in negative barycentric coordinate. In the algorithm used, points are passed counterclockwise, so

$$\text{to calculate } A,\; \mathbf{r} = (C_2 - C_1) \times (C_m - C_1) \tag{2.10}$$

$$\text{to calculate } A_1,\; \mathbf{r_1} = (C_2 - P) \times (C_m - P) \tag{2.11}$$

$$\text{to calculate } A_2,\; \mathbf{r_2} = (P - C_1) \times (C_m - C_1) \tag{2.12}$$

$$\text{to calculate } A_3,\; \mathbf{r_3} = (C_2 - C_1) \times (P - C_1) \tag{2.13}$$

Eventually the values are substituted in Equation 2.8 to get the barycentric coordinates. Note that although the calculated area is twice the area of the triangle, there is no need for this number to be divided since the ratio is computed. To decide the sign of the coordinate, the dot product of $\mathbf{r_i}$ and $\mathbf{r}$ is computed. If it is negative, then the point is outside of the triangle, and the corresponding barycentric coordinate calculated according to Equation 2.8 will be negated.

For the central point, whose neighbors are all to be eliminated at the same time as itself, the references are the closest lower-right, closest lower-left, and the middle of the closest

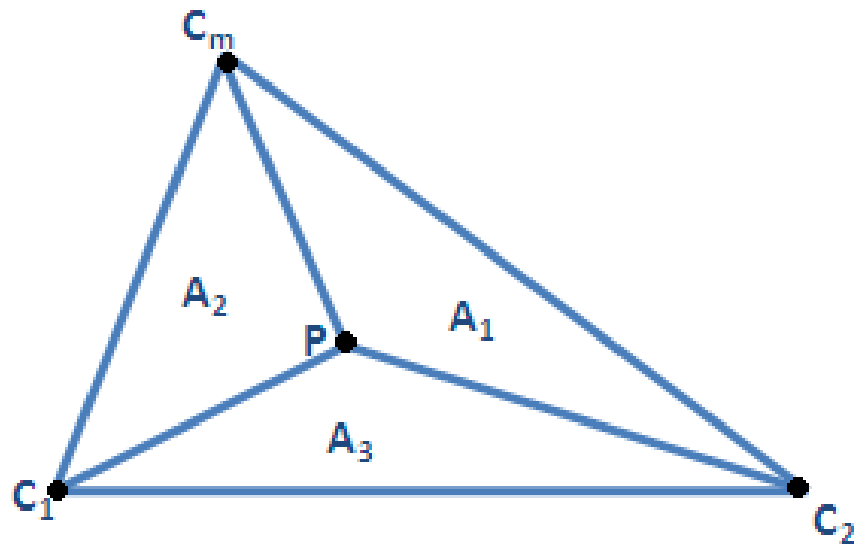

FIGURE 2.7: The barycentric coordinates of P with respect to C_1, C_2 and C_m will be determined. A_1, A_2 and A_3 are the respective areas of the point with the sides of the triangle. A is the area of the triangle formed by C_1, C_2 and C_m

upper-left and upper-right points. In Figure 2.6, the references for C_c are C_1, C_2 and C_m (defined earlier).

The algorithm calculates/removes details recursively, as shown in Figure 2.8.

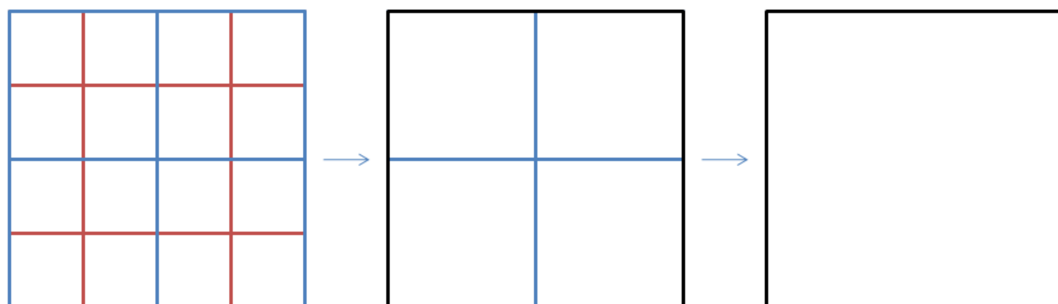

FIGURE 2.8: Recursive storing of details is as shown. Red details are stored with the blue details as reference. Later, blue details are stored with respect to black and so on until the coarsest level possible is reached.

The result of this algorithm, a less detailed cube may be seen in Figure 2.9.

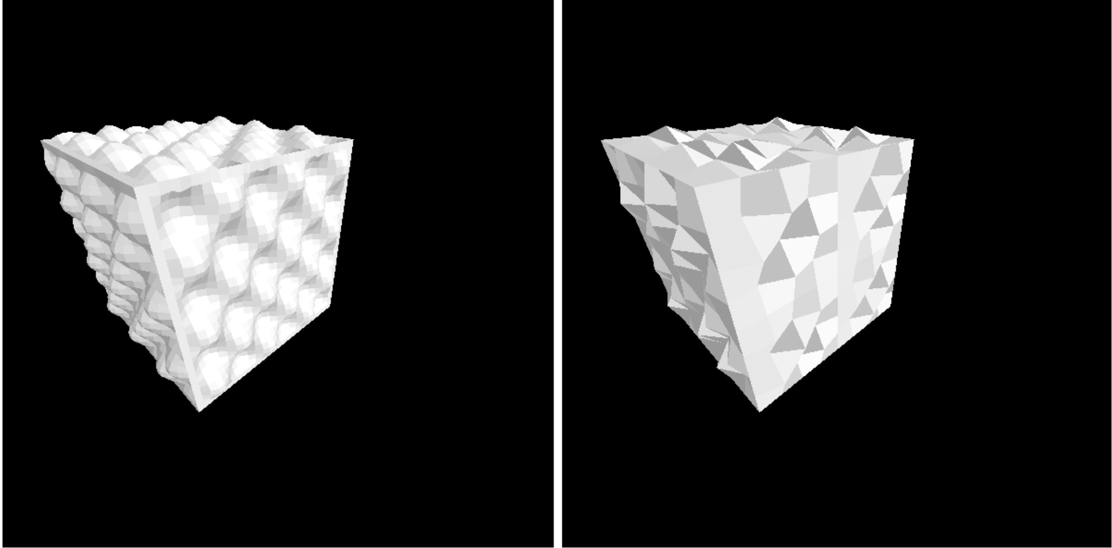

FIGURE 2.9: The model with removed details after: **(left)** 1 iteration, **(right)** 3 Iterations

2.5 Return the Details

When returning the details of the cube, the procedure is opposite of the algorithm described in Section 2.4. Going one detail up from the current position means adding one more point between two neighboring points and adding one central point for four interconnected points in a quadrilateral if such a point was present in the beginning of the model. Since the particle's positions may have changed during the course of model deformation, for each new point to be added, first it's corresponding projection is calculated with respect to three reference points on a plane according to Equation 2.7, and then it is moved above or below the plane based on previously stored information.

For this, it is considered that the model might have been stretched or squeezed while the detail was gone. So, the new height of the point might not necessary be the same as the old height. In fact, the height is inversely proportional to the change of the distance between the reference points as shown in Figure 1.1. If the points were moved apart, the height is decreased, and the other way around. So, the new height is calculated according to:

$$\frac{\text{new height}}{\text{old height}} = \frac{\text{old width}}{\text{new width}} \tag{2.14}$$

The only unknown in this equation is the new height, so it can be easily computed. After that, the previously calculated projection is moved in the same direction or the opposite direction of the plane's normal depending on previously stored data for a total distance "new height".

Figure 2.10 shows the result of returning details after stretching the model cube. It is clearly visible in the last part of the figure that the height is not equal everywhere. On the stretched faces it is almost flat, while on the other faces it remains the same.

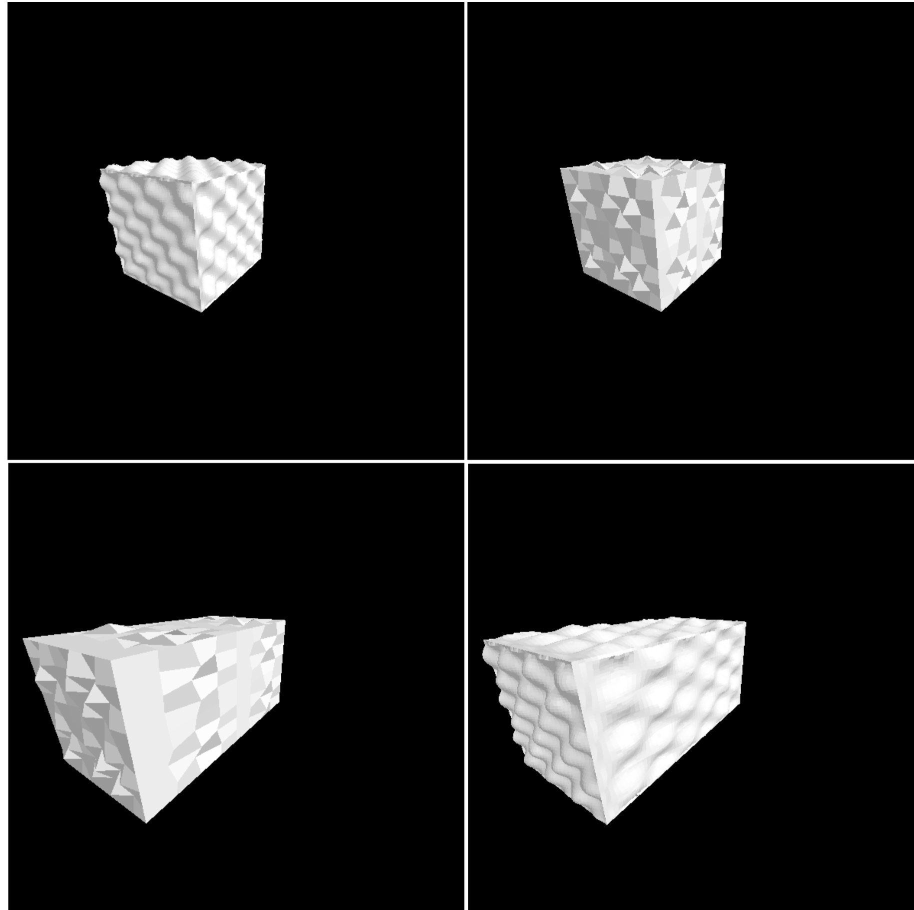

FIGURE 2.10: The deformation process of the model cube. From left to right, top to bottom: **(i)** Original Cube, **(ii)** Low Detailed Cube **(iii)** Stretched cube with few details **(iv)** Details returned to the stretched cube

2.6 GUI description

As a final touch of the project, a very user-friendly Graphical User Interface (GUI) was designed using Qt. The greater part of the GUI, as seen on Figure 2.11 is occupied by a window designed for rendering the output of the application. This part is interactive, the user may play around with the object using the mouse and examine it from all sides.

On the right side is a slide bar for zooming in and out of the window. Below it are several buttons used to remove and add detail or deform the cube. There is also an "Animate" button which produces a hard-coded animation displaying the cube bouncing from some imaginary ground. Note that this animation is not realistic since the physical

modeling was not done in this project, but is there with the purpose of showing the detail preservation of the cube.

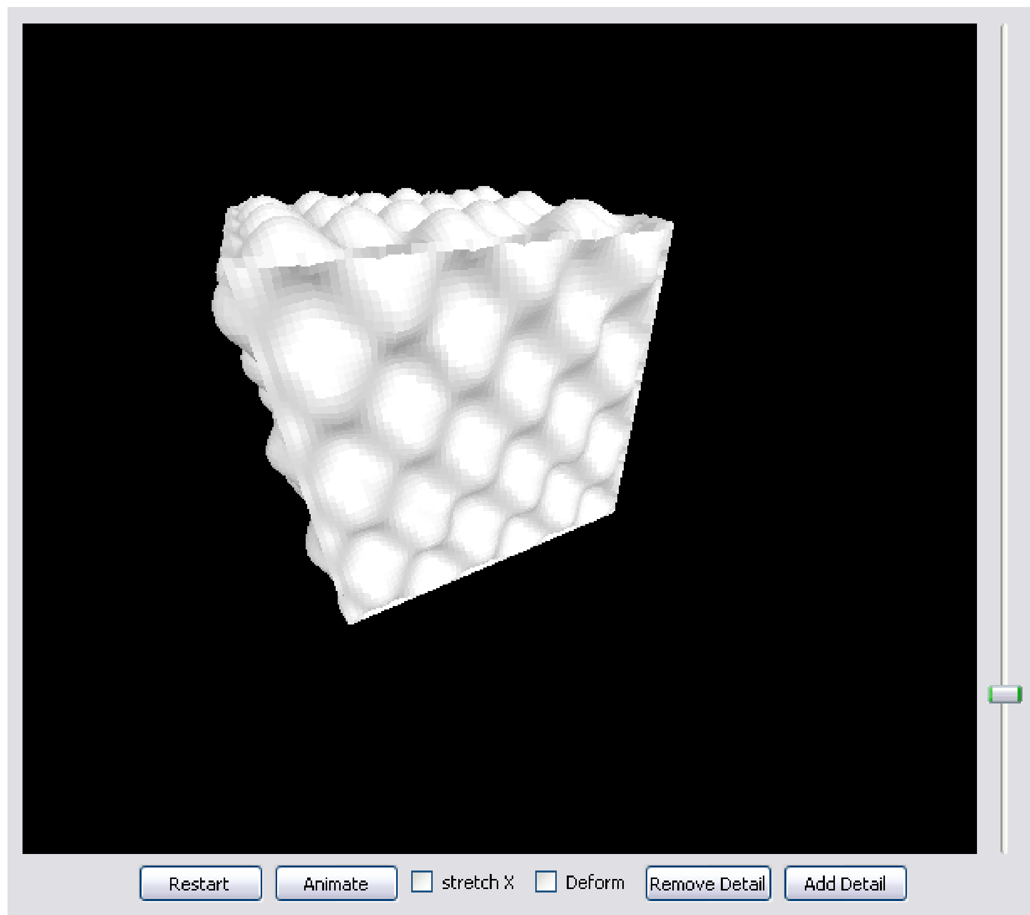

FIGURE 2.11: The User-Friendly Graphical User Interface done with Qt

Chapter 3

Conclusion

3.1 Discussion

The method developed results with fast and realistic deformations of our model. As a proof, Figure 3.1 presents removing of all details from our model and then performing extreme deformations to it. This is extremely fast as it only needs to update 4 points (as opposed to deforming 64×64 that were present in the original mesh). After the details have been returned there is no indication that the figure was ever simplified. The animation, in which the cube bounces from the ground meanwhile being squeezed and stretched exhibits the expected behavior as well.

The results from this project might be applied to computer games. Nowadays, objects in the games contain thousands of polygons. But if the user wants to interact with far-away objects this might be a problem because even today's graphics cards can not hold infinitely many information. Suppose in a first-person shooter game, the player shoots the cube presented throughout the paper. When he is far-away, he will see the low-detail version of the broken cube without even realizing this since it is very small. But as the distance between him and the cube decreases, the coarseness level decreases as well. So, when he eventually approaches it he will see it with the original number of polygons. Thus, the gameplay value would significantly increase.

3.2 Future Work

3.2.1 Further Improvements

- Currently, when a central point is removed, its information is stored with respect to three closest points, as described in Section 2.4. But in some cases this may

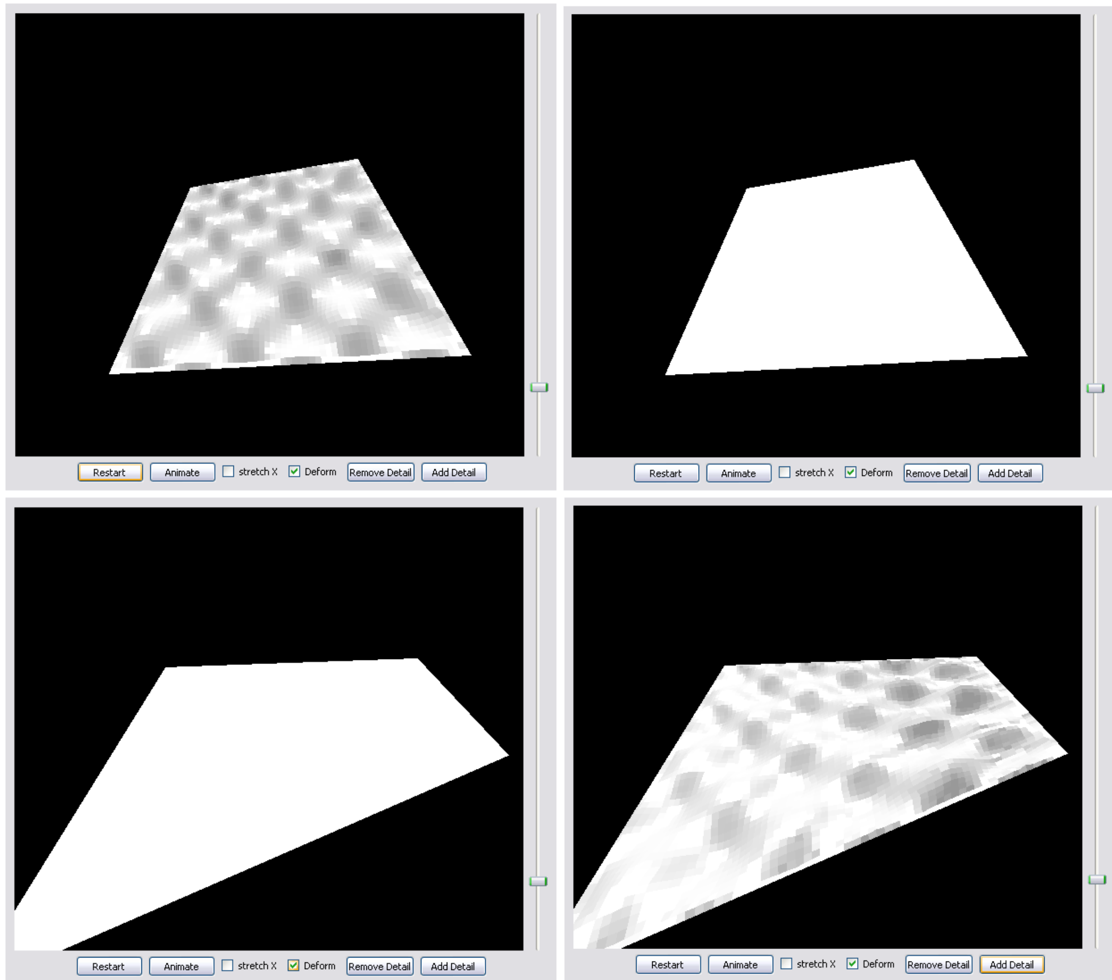

FIGURE 3.1: Extreme Deformation of one of the cube's faces. From left to right, top to bottom: **(i)** Original face, **(ii)** The face with no details at all, **(iii)** Custom Deformation, **(iv)** Previously stored details are returned.

lead to glitches, so it would be better if the algorithm is extended so the point store information with reference to closest 4 or even 6 points.

- At each step of detail-removing, the algorithm deletes every second point horizontally and vertically. This may not always be practical since sometimes distances between points may be large, and sometimes many points may be in the same place. If this is to be considered, and instead of patterned removal only clustered details are removed, it would most certainly lead to better results.

3.2.2 Future Expansions

A full extension of the algorithm would make realistic deformation on any mesh. To do this, the algorithm may be extended in the following way:

- Horman et al. developed a way for replacing an arbitrary triangle mesh by a regular quadrilateral mesh [7]. So the extended algorithm may ideally take any arbitrary mesh, run the method described by Horman on it, and then work with the mesh outputted as described in Chapter 2.

- For realistic deformation, one may use some of the existing methods described in Section 1.2 once the inside sampling of the points is done. This will result with real-world deformation. An extension to the existing GUI may result with user induced forces acting on the body.

Bibliography

[1] Leif Kobbelt, Swen Campagna, Jens Vorsatz, and Hans-Peter Seidel. Interactive multi-resolution modeling on arbitrary meshes. In *Proceedings of SIGGRAPH 98*, Computer Graphics Proceedings, Annual Conference Series, pages 105–114, 1998.

[2] Igor Guskov, Wim Sweldens, and Peter Schröder. Multiresolution signal processing for meshes. In *SIGGRAPH*, pages 325–334, 1999.

[3] Andrew Nealen, Matthias Müller, Richard Keiser, Eddy Boxermann and Mark Carlson. Physically Based Deformable Models in Computer Graphics. The Eurographics Association, 2005.

[4] Pfister H., Zwicker M., Van Baar J., Gross M. Surfels: Surface elements as rendering primitives. In *Siggraph 2000, Computer Graphics Proceedings*, volume 14, pages 335–342, 2000.

[5] Müller M., Keiser R., Nealen A., Pauly M., Gross M., Alexa M. Point based animation of elastic, plastic and melting objects. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer Animation*, volume 14, pages 141–151, 2004.

[6] Pauly M., Keiser R., Adams B., Dutre P., Gross M., Guibas L. J. Meshless animation of fracturing solids. In *Proceedings of ACM Siggraph*, volume 14, 2005.

[7] K. Hormann and G. Greiner. Quadrilateral remeshing. In B. Girod, G. Greiner, H. Niemann, and H.-P. Seidel, editors, *Proceedings of Vision, Modeling, and Visualization 2000*, pages 153–162, Saarbrücken, Germany, November 2000. infix.

[8] Ren Chen, Xiaonan Luo, and Hao Xu. Geometric compression of a quadrilateral mesh. *Comput. Math. Appl.*, 56(6):1597–1603, 2008. ISSN 0898-1221. doi: http://dx.doi.org/10.1016/j.camwa.2008.03.023.