A particle-based volumetric sculpting tool

A Particle-Based Volumetric Sculpting Tool

Martin Ulrich Seiler

Master Thesis
March 2009

Prof. Dr. Mark Pauly

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Abstract

We rethink the early explorative design phase of today's modeling pipeline. A modeling tool based on a volumetric representation of material allows modeling paradigms which are different from conventional, surface based approaches. By using *Governed Design* in combination with classical sculpt and modeling tools, we study a method which is based on well known physical concepts to ensure predictability, but is flexible enough to enable the creation of a variety of shapes in a short time. We performed an evaluation of our ideas by implementing an experimental modeling tool.

Zusammenfassung

Wir überdenken die frühe Designphase der heutigen Modellierungspipeline. Ein Modellierungstool, welches auf einer volumetrischen Repräsentation basiert, erlaubt Modellierungsparadigmen, die sich von oberflächenbasierten Ansätzen unterscheiden. Wir kombinieren eine neue Methode, welche wir *Governed Design* nennen, mit klassischen Modellierungstechniken und schlagen eine System vor, welches auf bekannten physikalischen Konzepten aufbaut. Dadurch erreichen wir Vorhersagbarkeit, sind aber gleichzeitig flexibel genug, um eine Vielzahl von Formen in kurzer Zeit zu schaffen. Wir haben unsere Ideen in einem experimentellen Modellierungstool implementiert und evaluiert.

Contents

List of Figures

Acknowledgements

Special thanks to: My family for their tremendous support during my studies. My supervisor, Prof. Mark Pauly, for his guidance and wisdom. Thanks to: Tahmineh, for her love and patience. My fellow Urs, with whom I not only worked together on this project, but also spent hours learning for exams and accomplishing a lot of cool projects. Julian for his support and fruitful discussions. Thomas and Ruben for helping me with the revision of my thesis.

Chapter 1

Introduction

1.1 Motivation

Geometric modeling, a sub-category of Computer Aided Design (CAD), is the process of creating three dimensional shapes on a computer. The myriad of shapes that exist makes the creation of *geometric modeling tools* a task of managing complexity. Our motivation is to advance this creative process and to increase productivity. If the benefits of CAD could be brought to the early design phase, which is nowadays often done outside the digital domain, the increase in productivity would be a real benefit. Professional designers that want to quickly visualize a shape or artists who look for the next great idea would benefit from a tool which could be used to rapidly explore different shapes. Ordinary users could benefit from a system where they can easily define their idea of a shape, which then could be used for shape retrieval [33]. Many existing three dimensional (3D) modeling systems are rather complex and our motivation is to have a simple tool to reach a broad audience.

1.2 *Quiqshape*

For this thesis we developed a 3D modeling tool called *Quiqshape*. The tool was created in close collaboration with Urs Dönni, whose thesis [20] shares some parts with this one, while complementing other parts. Both theses give a complete overview of the system, but will go into more detail for selected topics.

1.3 Goals of the System

Our initial idea was to help artists and designers to find new shapes. Thus we defined the central and most abstract goal of this project to be:

- We want to provide artists with methods for shape finding.

This goal serves as the guiding handrail during the development of our tool. We want to support and improve the early explorative design phase of virtual objects. In order to achieve this, we defined different, increasingly refined sets of goals our system should meet and then made design decisions.

- We want to get closer to the behavior of real world objects since people have a good intuition about the real world.

- The system should be capable of representing as much detail as possible in order to allow for the modeling of more complex objects.
- Real-time user interaction is desired to ensure a fluent work experience.
- Predictability and flexibility of the outcome should be in healthy balance in order to gain maximum productivity.
- The form-finding process should be aided with intelligence/rules which govern the global properties of the shape being designed.

We believe that a design tool satisfying these goals would be a real benefit for the early design phase. In order to achieve these goals, many decisions have to be made regarding the technology to be used. Some of the goals are of competing interest: The less detail there is, the faster data can be processed by a computer, which in turn leads to faster user interaction.

1.3.1 Material Representation

Ordinary 3D content creation software only represents the surface of an object, justified by the fact that only the surface is visible to human eyes. If one deletes triangles from a triangle mesh, a hole to the empty interior is created. This is because a triangle mesh is nothing more than a thin shell. This is clearly counterintuitive. In order to reach our goal of a more realistic representation of virtual objects, our system needed some sort of physically-based simulation. We found that the following properties of real world objects would be desirable in our system. We expect to achieve not only an increase in productivity from a more intuitive behavior, but also the possibility to shift to a new modeling paradigm:

- Volumetric representation
- Mass and density
- Different material types
  - rigid, elastic, plastic, and fluid
- Inter-object collisions

This allows certain operations to be more efficient. For example, the designer can let such a system do some work for him, as it now has its own dynamics. Topological changes should be particularly easy to achieve: one can simply delete matter from the model to create a hole. New surface is implicitly generated and does not have to be added explicitly by the user. The material can melt and freeze, be sensitive to gravity, or not. We imagine that these properties yield a very dynamic and creative playground for new ideas and explorations.

1.3.2 Extended Tool Intelligence: Governing Rules

Even though some artists are incredibly creative beings and can produce new art simply out of the corners of their mind, the everyday reality proves to be different. New shapes are created by experimenting, playing and combining existing concepts. The creative process is supported with clay, plasticine and/or other tools. Our system should allow an artist to express his general ideas about the shape to be designed: What kind of symmetries should the shape have? How smooth is it? Is there a hole in the middle? Which regions in space have to be covered by material? Our tool should then enable the artist to find an appealing shape satisfying his initial abstract design ideas and constraints.

1.4 System Overview

Now that we have defined our goals and provided a rough overview of the desired features of our tool, it is a good time to give a first impression of how our specific implementation looks. Unlike most tools in widespread use, our 3D modeling tool *Quiqshape* operates on an approximation of real matter. The core of our system is a physics simulation of deformable matter. We created a state of the art graphical user interface which allows simple access to the different tools and operations that can be applied to the 3D objects. A storage system ensures that during each stage of the design process backups can be made and stored to disk. In order to integrate our tool into an existing tool chain we added an exporter to a widely known format used to store 3D triangle meshes. The set of tools one can use to model a shape can be divided into two categories:

- Classic modeling tools
- Governing rules and constraints

The next two sections describe what we mean by this. A short description of the user interface is given in the appendix, so that readers who have not seen the tool yet get at least some idea of how it looks.

1.4.1 Classic Modeling

The artist is given a set of tools which are closely related to real-world tools one would use for clay sculpturing. We implemented tools to add and remove material, and others to push or pull the surface of the sculpture. All these actions are performed by the artist with a clear intention. The shape to be modeled exists at least roughly in his mind. To go one step further we enable the user to change the state of matter between rigid, elastic, plastic and fluid. We have tools to add and remove heat to objects and we implemented the underlying physical properties such as heat diffusion and freezing/melting of matter.

1.4.2 Governed Design

In contrast to the *classic modeling* approach *Governed Design* is a totally different approach to shape finding. This paradigm requires the artist to give the system *guidelines*, *design constraints* and *rules* on what properties a final shape should have. For a given setting the shape itself is then computed as a dynamic solution of the defined rules. New material added to the system tries to satisfy the constraints while the user is still allowed to interact. As this process is very dynamic, an appealing shape might exist only a limited amount of time. Therefore the user needs to have control over time while the system evolves, meaning that he can stop time and go to a previous state.

The guiding rules were implemented as actual objects which are part of the scene: So called *tool objects*. Their position, orientation and scale control the properties of the tools. One property very central to design is symmetry; therefore we have a set of tools which govern the symmetry of the final shape. Other abstract properties are global smoothness or the topology of an object. We also provide tools to control these properties. For example we implemented a way to specify where holes should be. The overall process of shape finding can be described as putting a few rules in place which govern the macroscopic form of the shape followed by the use of the *classic tools* (see section 1.4.1) to guide the material into new shapes.

1.5 Target Audience

As stated earlier, we want to support the early form-finding process. Therefore our target audience is artists and our tool can be seen as a creativity stimulator for finding prototypes. Our tool is not geared towards the exact engineering type of CAD but it is used to find new forms, to prototype an exciting shape, or to create quickly the basic shape of sculptures and characters. As it is aimed at the early design stages, we have ensured that our tool integrates well with other existing methods which complement the work flow.

Chapter 2

Related Work

This section discusses work related to our main goal, which is to improve the early explorative design phase by computer aided methods. We discuss the related work of specific system components in the corresponding sections.

Methods to develop new shapes have existed for centuries. In recent years, several virtual methods have tried to imitate and make improvement over real methods or to create a new way of design.

2.1 Sketching

A well established and still widely used method is (seemingly) pedestrian: Sketches using pen and paper. To think about a shape and sketch it quickly, then think again, refine, and improve is a very good method to spawn creativity as described in [31]. There Lipson and Shpitalni discuss the benefits of sketching in conceptual design and analysis which is summarized as follows:

1. It is fast, suitable for the capacity of short term memory
2. It is implicit, i.e. describes form without a particular sequential structure
3. It serves for analysis, completeness check and simulation
4. It is inexact and abstract, avoiding the need to provide unnecessary details
5. It requires minimal commitment, is easy to discard and start anew

The attempt to design a digital tool which is as comfortable to use as pen and paper and incorporates the advantage of the digital world is a very challenging one and interestingly has already been explored by Sutherland in the early 1960s [38]. Even though there are tools for 2D sketching, we will discuss mainly three dimensional (3D) sketching tools. Such a tool should not only be a mere clone of a real sketch board but should have logic to automatically transform the user's strokes into three dimensional shapes. One of the most widely known works is the one described in [27]: This software called *Teddy* inflates user-drawn two dimensional (2D) strokes into 3D shapes. Abstract and simple 3D shapes of organic form can be quickly created even by a novice. Our work partly aims to achieve what *Teddy* does in terms of simple usage, but also aims to go further in the sense that our underlying model will be a volumetric one providing physical simulation of the material. Another more recent tool is *iLoveSketch* [12], which is a 3D curve sketching system. One of the differences from our approach is again that we use a volumetric representation and that we use material simulation to improve the surface quality. The main difficulty is to interpret a 2D stroke and transform it into 3D spatial information. An interesting approach implemented by [28] transforms wire frame sketches into actual 3D shapes by using a template shape used to interpret the 2D strokes. Such tools dive deeply into the realm of computer vision, where the task is to interpret 2D images and find corresponding 3D semantics. The aforementioned *template shape*, a 3D skeleton shape, has to fit the drawn 2D sketch to a certain degree. This mesh is then transformed into a shape that matches the drawing best. Our tool attempts to provide more of a unified approach. This means that the sketches are transformed to 3D from the very beginning. This gives the user the possibility to follow a more spatially-oriented work-flow as she is exposed to the third dimension right away.

2.2 Sculpting

Another method frequently used, besides pen and paper, is sculpting: Modeling a piece of clay or plasticine is most common. Often the shape then has to be further processed on a computer to get to a final product, be it digital or not. As one does not want to lose the time already spent on clay modeling, one could use a shape acquisition method to digitize the object, such as the one described in [18]. This is often done even for non-explorative work as artists might be more comfortable with real clay. The first reason is that they are used to model with their hands and the second is that, in the past, the amount of detail which could be modeled on a piece of clay exceeded what was commonly possible with digital volume modeling tools by far. The associated field is called *virtual sculpting* and every state of the art modeling package has support to sculpt surfaces in one way or another. Our system implements sculpting related tools but their effects are different in some cases, as we work on a volumetric representation. Related work regarding actual algorithms is described in 5.1.1.

2.3 Design by Tool Design

Several factors, which are not immediately obvious, have an influence on today's zoo of shapes. Aspects like production cost or availability of manufacturing chains have their respective influence on what kind of shapes are finally presented to a consumer. If the shapes reside in the digital domain the mentioned factors cease to have an immediate influence. However, as creativity is often a recombination of impressions, they still play a role to some degree. This is where an approach recently named *Design by Tool Design* [21] could step in to boost creativity.

The influence of the used tool on the appearance of the final shape should not be underestimated. Depending on the available tools, certain shapes are easier to achieve than others, leading to a monotony in design after a few years. This also holds for non digital media: It is easy to draw two dimensional shapes with a pen on paper but hard to draw even simple three dimensional ones in an unambiguous way. One finds that it is particularly hard to model certain shapes by using an ordinary surface-based mesh-modeling tool. A class of such hard-to-model shapes is high-genus objects. The solution presents itself in the creation of a new tool which simplifies the design process by narrowing the overwhelming size of the original configuration space down to a smaller set that contains a larger density of actual appealing shapes. One such tool is *Topmod*, which was used to create the shapes presented in [11]. Our goal is to improve the early design phase of shapes, and we want our tool to be able to create as many types of shapes as possible in an intuitive way. This includes, in particular, topologically more complex shapes.

In order to create shapes which are hard to imagine, one can fall back to mathematics and use its generative power to create shapes. One recent example of such an approach is presented in [24], where a tessellations of the Poincaré disc is mapped into different spaces, yielding interesting sculptures with stunning complexity and beauty. We explored and implemented the use of simple mathematics to generate an initial shape.

2.4 Tool Intelligence

As mentioned, the underlying representation of a modeling tool, be it digital or not, has an impact on the shapes commonly produced. In the real world it is nature which defines the properties of the system and the forces shaping it. These properties, as well as macroscopic forces like gravity and other environmental influences like wind, water or living creatures are all entities which participate in the shaping of an object. In the digital world the programmer defines and models what influences the shape of an object. One particularly instructive approach is described in [13], which demonstrates what is meant by *tool intelligence*: Intelligence is built into a small cell. The cells interact with each other according to evolving rules. Many cells emerge and aggregate into a macroscopic structure during an artificial evolutionary process without user interaction. We apply the core idea of this work which is to use some logic to create self-organizing geometric primitives. While we do not reuse the cell approach directly, our shapes also emerge by a union of many very simple primitives, namely points.

As an example of the exploitation of physically-inspired computer simulations for shape finding see [29]. There, a mass-spring system represents the *tool intelligence*. The final form corresponds to an equilibrium state of the mass-spring system under gravity. Afterwards, the model is taken, flipped around, and may then serve as a building. Such a construction ensures an improved stability because the force distribution inside the structure is advantageous once the shape has been flipped around, and gravity acts the other way round. Therefore in this work the tool intelligence improves the likelihood of structurally stable shapes. The basic modeling primitive of our tool is a point set, and there are many possibilities to let *tool intelligence* shape the point cloud in order to produce shapes with certain properties.

Chapter 3

Material Simulation

3.1 Overview

The basis of our shape modeling tool is the matter simulation subsystem. All features and tools are built on top of it. We want to have real-time user interaction (see 1.3), hence we ensured that all our algorithms exploit multi-core architectures. The matter representation in our system is a point cloud. This representation helps us to reach our goals: for example, the absence of explicit connections between primitives simplifies topology changes. The downside of points as primitives is that neighborhood queries get more expensive compared to other data structures which store the relation explicitly. However, for almost any task there are efficient algorithms that work with point clouds.

As a point is the most simple geometric entity it can be easily understood by users. Other modeling primitives, such as NURBS, are mathematically more sophisticated and the designer needs to know their properties in order to be able to work with them.

Usually one calls points enhanced with additional properties, such as mass or thermal energy, *particles*. Based on the *particle's* information we compute how the system evolves over time. Additionally, the point based representation has several advantages when it comes to flexibility. It is rather easy to add algorithms which perform operations on points. For example, one can transform points by mathematical functions to generate interesting effects.

We implemented different methods defining material behavior. The most intuitive ones were derived from physical models. For more information regarding this subject refer to [20]. In section 5.1.2 we explain how the three types of matter interact with each other.

3.2 KNN optimization

In order to compute the next state of our system we need to solve a set of differential equations. In particular, we need to know spatial derivatives of a certain property P of the material at a given point $\vec{x}$. Our algorithm is built on top of Smoothed Particle Hydrodynamics (SPH), where a property is spread in space using superposition of kernel functions W. The used kernels have special properties.

In particular W is:

- radially symmetric
- finite in space

- smooth

The central equation used to interpolate properties looks as follows:

$$P(\vec{x}) = \sum_i P_i \frac{m_i}{\rho_i} W(\vec{x} - \vec{x}_i, h_i). \tag{3.1}$$

Note that ρ_i has to be computed for each step by $\rho_i = \sum_i m_i * W(\vec{x} - \vec{x}_i, h_i)$ before any property can be interpolated.

Since, the kernel's influence is finite, we only need to consider all particles inside the kernel's cutoff radius $2 \cdot h_i$ to compute a property (or its derivative) at a certain point. To speed this up, we need some fast algorithm to compute the set of the k-nearest neighbors (KNN). And this is where our optimization kicks in.

3.2.1 kD-Tree

To speed up KNN queries, we rely on a kD-Tree [14]. The kD-Tree is able to perform the task of a single query in $O(log\ n)$. The construction needs to find the median first to be able to use the median-split strategy. Therefore, the construction time accounts with $O(n\ log\ n)$, if a linear median-finding algorithm is used.

We decided to use a kd-Tree over a grid based approach because the kd-Tree is not restricted to a certain domain size. However, our code could easily be adapted to use a hash-grid for example.

The most time consuming task for solving an SPH system is the KNN search. Therefore, we decided to parallelize the kd-Tree. This is pretty straight forward, compared to a parallel construction, as querying the kd-Tree is a read-only operation and therefore no data races can occur. Our system spends about 30% of its time performing KNN queries. The kd-Tree construction is not parallelized, it is not worth the effort as only 3% of the time is used to reconstruct the kd-Tree at the beginning of each computation step.

3.2.2 Adaptive Integration

We used the simple *leap frog method* [26] enhanced with an adaptive step-refinement as described in [25] (page 427). This step-refinement algorithm is applied whenever a change in position would exceed half the kernel size and ensures that no two particles can *jump* over each other without interacting. In *Eulerian* approaches, the same concept occurs and is called the Courant-Friedrichs-Lewy (CFL) condition [17]. It is important to note that this condition can be circumvented by using an implicit integration schema.

Chapter 4

Visualization

4.1 Overview

The visualization system is right next to the material simulation layer at the core of our system. It has to provide a fast, appealing and accurate presentation of the modeled objects on screen.

4.2 Rendering

Our rendering system is based on a hardware (GPU) accelerated rasterization using OpenGL 2.1 [6] as basic Application Programming Interface (API).

4.2.1 Deferred Shading

In order to spend the precious fragment processing time on pixels that are actually visible, we implemented a renderer based on *deferred shading*. This enables not only a cost efficient implementation of the sphere splatting algorithm, but also reduces the shadow-map lookup costs as it is only evaluated for visible fragments.

4.2.2 Shadow Mapping

We visualize a 3D world on a 2D screen. We try to give the user many visual hints in order to make a shape's structure as clear as possible. Productivity depends on how easy one can understand the actual shape from its projected appearance. Shadows in combination with proper shading can help a great deal in understanding otherwise ambiguous configurations. Figure 4.2 and figure 4.3 are a compelling argument why modeling tools should support more advanced rendering than for example simple Gouraud shading.

4.3 Surface Extraction

First of all one has to define what the surface is. This is not straight forward for our SPH [19] based method. A common approach is to generate an iso-surface of a certain threshold value from the objects smoothed density field. One uses radially symmetric kernel functions which are positive and have a finite radius of influence. Hence, using zero itself as a threshold value might seem a natural starting point.

However, surface reconstruction done with zero-threshold yields a surface of a volume which is a union of spheres. Usually, this leads to a pretty bumpy surface, depending on the average inter-particle distance. The usual practice is to use a larger threshold value yielding more smooth interpolated surfaces. Such a surface fits the actual particle positions better up to the point where individual particle spheres start to appear. It is important to understand that the appearance of the object varies with the selected iso-value and there is no one-size-fits all solution, therefore this is a user controllable value in our system.

4.3.1 Screen Space Methods

The actual iso-surface extraction can be done in various ways. One being the family of algorithms based on the well known marching cubes algorithm [32], which is a view independent extraction of the whole surface. Another family are the view dependent methods which extract only the visible surface for a given viewing position. This family can be further subdivided into ray casting and splatting methods.

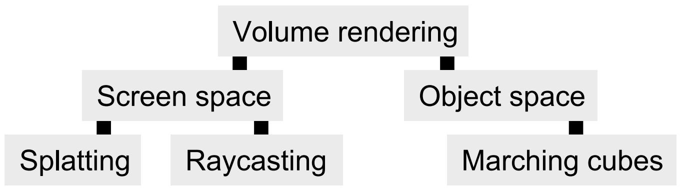


Figure 4.1: Categorization of volume rendering algorithms.

We use the view independent marching cubes algorithm to extract an iso-surface and export it as a triangle mesh. To visualize our scene on the screen we decided to use a screen space based method. Since our goal is to have real-time interaction, we do not rely on the well known marching cubes (MC) algorithm. It would have been too slow for this task. Since our system can be very dynamic, we need a fast algorithm, which can handle large changes in shape and topology easily. It is also important that the algorithm does not rely on inter-frame coherences, because we support large scale deletion and insertion of matter.

To render the objects to the screen we implemented a sphere splatting algorithm 4.3.2, where an additional smoothing step makes it possible to interpolate surface properties, like *color* or the *surface normal*. In contrast to other screen space methods our selected algorithm has a straight forward implementation and runs almost entirely on the Graphics Processing Unit (GPU), complementing very well with the fact, that our physics simulation runs solely on the Central Processing Unit (CPU).

4.3.2 Sphere splatting

For a simple first implementation we relied on sphere splatting which simply renders a sphere at each particle's position.

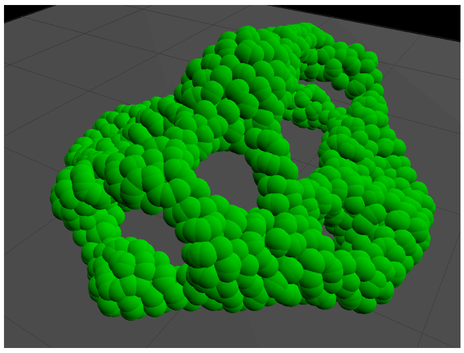


Figure 4.2: Simple sphere splatting already gives a quite good impression of shape.

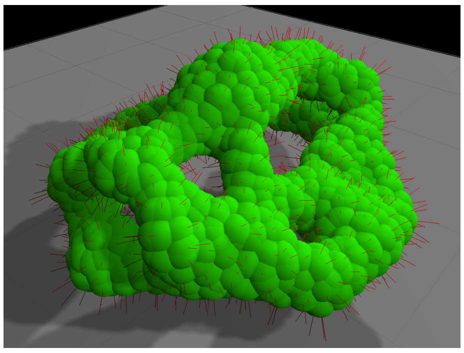


Figure 4.3: Additional shadow mapping.

The motivation of using spheres can be illustrated optimally by a correspondence to the iso-surface of the smoothed density field: the volume enclosed by the zero-level iso-surface corresponds directly to the union of spheres, centered at the particle positions. This holds if the sphere radius is chosen to be the kernel size.

4.3.3 Simple Normal and Depth Smoothing

A smoothing pass of the depth and normal map of an extracted surface promises to be a good solution to generate a visually smooth and appealing representation on screen. [16] and [34] use a smoothing step to enhance visual quality of the surface. Those algorithms generate a depth and a normal map and filter them in an additional step with a smoothing kernel. We found that having a fixed kernel size for the smoothing filter, which is applied in a non perspective aware manner, does not live up to the expectations we had:

- Due to a fixed kernel size, viewing the surface from distance is not the same as a close up shot.
- Since the kernel is in screen space, tilted surfaces are smoothed over a larger distance in world space.

As long as the view position stays at the same distance to the visualized surface, the mentioned effects are not as obvious. However, in a highly dynamic system as ours such effects are quite troublesome. Though it would be possible to overcome the limitations by adapting the kernel size and shape to cope with zooming and perspective the prize would be a huge performance penalty for large kernels. Hence, we decided to use the splatting algorithm described in the next section.

4.3.4 Attribute Blending of Spheres

To improve the visual quality we implemented the algorithm proposed in [10]. This algorithm is able to smoothly blend between any kind of sphere property. We used it to blend normals and colors which yields a surface that appears to be smoother and has nice color gradients between spheres of different color. The algorithm could also be used to additionally blend other properties like *texture coordinates*

The algorithms basic steps are:

1. Render spheres to obtain an initial depth buffer.

2. Shift the depth buffer back by ϵ. In our implementation we set ϵ equal to the particles sphere radius. This has the effect that only particles contributing to the front surface will be rendered in the next pass.
3. Render spheres again and accumulate properties and weights.
4. Normalize properties for all pixels by dividing the accumulated values by the accumulated weights.

If a fragment lies in front of the shifted depth buffer its weighted property (normal, color, etc.) is accumulated using *alpha blending*. All properties along a *view ray* (the dashed line in figure 4.6) sum up and are finally divided by the summed up weights. In order obtain a smooth interpolation of properties the used weighting function has to satisfy two constraints:

- Vanish towards border of the splatted spheres (i.e. the ellipse in screen space).
- Vanish towards the shifted surface depth.

In our implementation we use a product of two weighting functions W^1 and W^2, computed as follows:

$$W^1 = 1 - \frac{r_f}{r_p} \tag{4.1}$$

$$W^2 = \frac{z_f - z}{\epsilon} \tag{4.2}$$

$$W_i = W_i^1 * W_i^2 \tag{4.3}$$

Figure 4.8 visualizes the used quantities in equation 4.2 and 4.1. The final weight is the product of W^1 and W^2. Other functions could be used as well, as long as the properties mentioned above are satisfied. The algorithm works quite well. However, the silhouette of the shape corresponds exactly to the one produced by simple sphere splatting 4.3.2. This is an issue which other approaches, like ray casting of an iso-surface, do not have.

4.3.5 Shadow mapping combined with sphere blending

The art in computer graphics is to find a set of algorithms which work well together. In our case this is the particle splatting method which has to work well with shadow mapping. A straight forward combination of the two can lead to the artifacts shown figure 4.10.

This is because the surface might *appear* very smooth after the spheres have been blended, but the positions used to compute the shadows are still the ones from the original bumpy surface. To solve this problem we implemented percentage closer filtering (PCF) which samples the shadow map over a larger area, reducing the impact of the artifacts. Figure 4.12 shows the final result.

4.4 Managing Complexity

As mentioned in our goals (see 1.3) we want to support as much complexity as possible even though we focus on the early design phase where especially surface detail is not yet of importance. The reason is that there are some shapes for which already first prototypes can become rather complex. Despite the need for complexity we also have to keep a convenient work flow and ensure a good user experience. These two goals are of competing interests as more complex objects require more computational resources and therefore reduce the interaction-rate. Current hardware allows the simulation of up to 100k particles using our Smoothed Particle Hydrodynamics (SPH) based algorithm. But our rendering system, lacking an effective culling algorithm, quickly becomes the bottleneck, therefore we allow to take material out of the simulation and let the user reduce the number of representative particles visible on screen. This enables one to work on models with more than the maximal number points that can be handled simultaneously by our system.

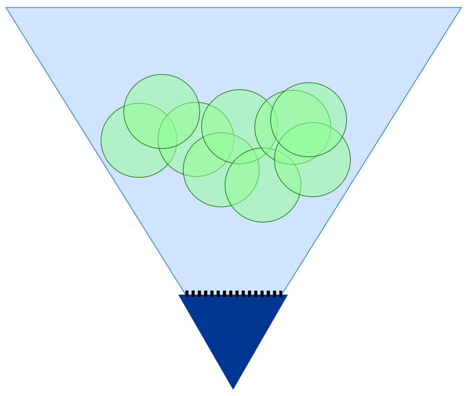


Figure 4.4: A 2D projection of an exemplary setup. The material is in light green, the camera frustum in light blue. The camera and the image plane (dotted line) are visualized at the bottom of the image.

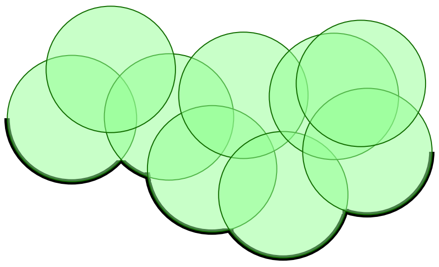


Figure 4.5: The first render pass initializes the depth buffer (thick black line).

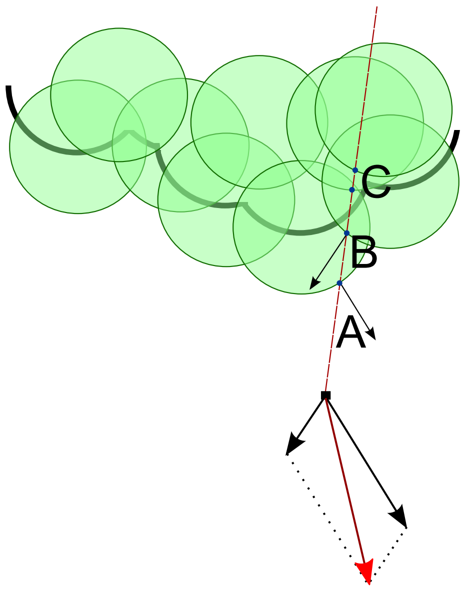


Figure 4.6: The fragments which lie in front of the shifted depth buffer (thick black line) have an influence on the blended properties. The fragments generated at point C are culled by the hardware depth test and do not contribute. Surface normals serve to illustrate property blending: At point A and B normals are computed, weighted and accumulated into the frame-buffer.

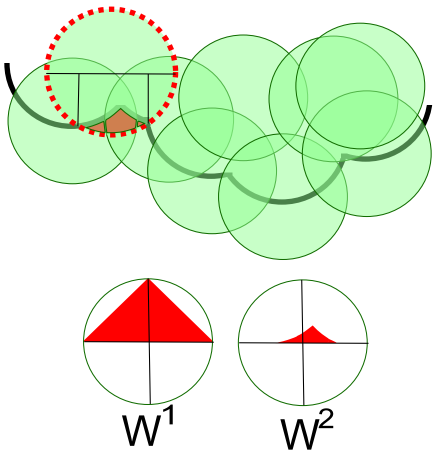


Figure 4.7: Weight Computation: The function chosen is a product of two terms. The first W^1 being a linear fall of proportional to the projected radius. The second weighting term W^2 smooths out discontinuities otherwise introduced by spheres which intersect the depth buffer.

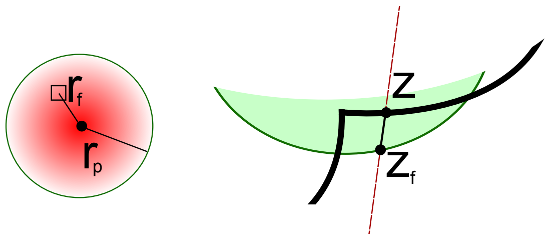


Figure 4.8: Visualization of the quantities used to compute the weighting function. Compare with equation 4.1 and 4.2

.

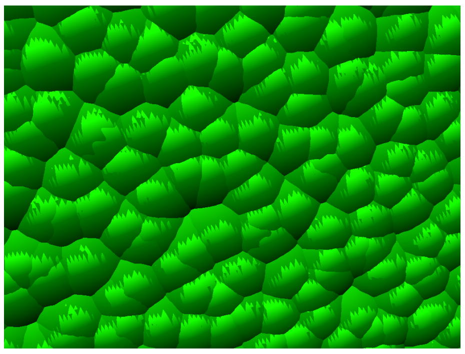


Figure 4.9: No PCF, not smoothed: the lowest visual quality

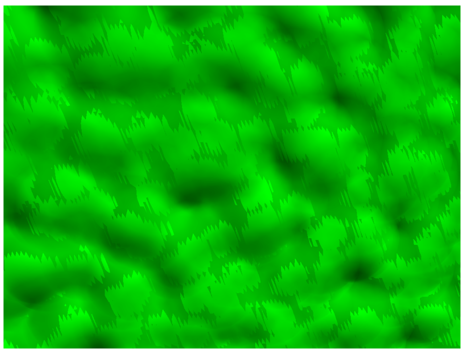


Figure 4.10: No PCF, smoothed: unwanted shadows are still visible

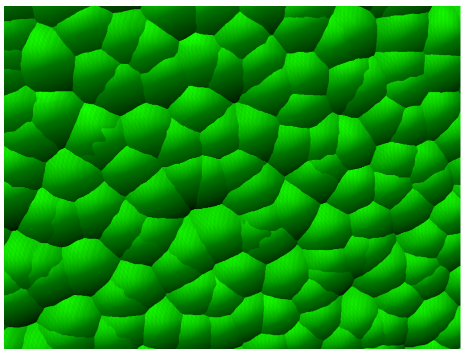


Figure 4.11: PCF, not smoothed: shading not continuous

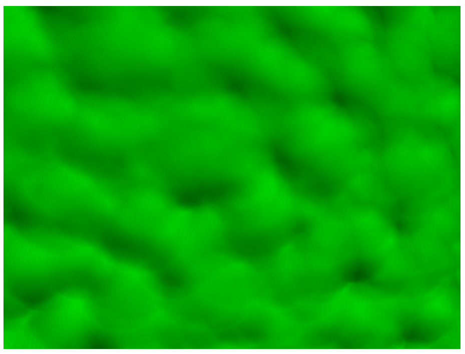


Figure 4.12: PCF, smoothed: the best visual quality

Chapter 5

Classic Modeling: Sculpting

5.1 Overview

In this section, we present what we call the *classic modeling* approach. It is best described as a *volumetric sketching and sculpting* process, which is in its nature similar to actual real-world sketching and sculpting. However, porting this technique to a virtual world offers many opportunities to optimize the handling and to increase the creative scope. One of the benefits of virtual worlds is that physical laws are not predefined: they need to be explicitly modeled and can therefore be twisted, tuned and adapted to one's needs. The tricky part is to choose a suitable approach for the actual implementation: What algorithms and data representations can provide an authentic behavior and still allow for the efficient simulation of a sufficient amount of matter?

The concrete goals derived from the abstract ones (see 1.3) describe a system which allows the following:

- Addition and subtraction of material.
- Topological changes of the shape.
- Plastic deformations for sculpting.
- Fast user interaction.

Besides the functional goals, the system should also be intuitive to use.

5.1.1 Related Work

During the past twenty years, there has been a lot of research on *digital sculpting* and *volumetric modeling* systems. The difference between *digital sculpting* and *volumetric modeling* is that the latter supports topological changes of the shape without additional effort. While designers working mostly on genus-zero shapes will not benefit directly from this advantage, it readily allows for the sketching of more complex shapes in a simple way. And for this, the method does not have to provide the detail that a mesh-based approach can provide for topologically trivial shapes. To our knowledge [22] was the first attempt to implement many of the goals defined for this project. This system uses a volumetric pixel (voxel) grid to represent volume. A simple boolean value at each position in the three dimensional (3D) grid states whether matter is present or not. They also introduced a series of manipulation tools to add or subtract matter, as well as using the sandpaper tool to smoothen the surface. In this work, we try to go one step further by incorporating physically-based effects to the material. For the voxel-based methods there are several ways to allow for elastic deformations of the system. One simple approach proposed by [23] is to assume that the voxels are connected by springs. In our case, a major drawback of the standard voxel-based methods is their domain limitation. Even though this could be solved by using a spatially adaptive data structure like an *octree*, we decided not to use such an approach, as the solution of the differential equations on such structures becomes a lot more tricky and is not yet fully researched.

Recently, a more popular approach chosen by several digital sculpting systems, such as [36] and [39], has been the use of an adaptively sampled distance field (ADF) to store the object's shape. The distance field is a scalar function that relates to each point in space its distance to the surface. If a signed distance field is used, one can also perform inside/outside tests and therefore some implicit notion of volume is present. However, the main incentive for using an ADF is its ability to store a lot of surface detail while still supporting efficient operations, such as carving out material from the surface with arbitrarily shaped tools. Unfortunately, so far there are no straight forward implementation of algorithms to perform physically-based deformations for ADFs. Therefore the ADF approach is not suitable for our project either.

A similar approach to ours is found in [37]. This system is also a sculpting solution based on particles with real-time deformations. The implementation relies on the simulation of the *Lennard-Jones potential* [30] to compute and integrate inter-particle forces. In order to maximize performance, particles are actually only simulated in regions spatially close to the current position of the deformation tools. Because all their tools have a short interaction range, they are able to model objects with millions of particles and report that their marching cubes (MC) [32] based rendering system is the bottle-neck. In contrast, some of our tools incorporate forces which act on a global scale – potentially on all particles of the scene. Consequently, we need to be able to run the material simulation on all particles in parallel in our project. [30] also noted the advantages of using spheres as the basic building blocks because it simplifies several of the involved data structures and algorithms compared to systems allowing for more complex primitives. This is one of the main reasons why we use a *point-cloud* for modeling. We also use a similar approach to implement custom tools which are made out of particles themselves. However, compared to [30], our tool goes further with regard to its features and the complexity of the underlying simulation, as well as concerning the usability of its tools.

An overview of existing pysically-inspired algorithms for shape manipulation and a detailed description of our implementation can be found in the complementing work of this thesis [20].

5.1.2 Material System

By using a *point-cloud* as our basic modeling primitive we benefit from the fact that it is easy to implement algorithms operating on them. Many different methods are possible to animate the material. Our first attempt was to create an exact copy of a potters workbench, including physical friction between objects, heat transfer to simulate change of matter state, real-time collisions, and of course gravity. The basic idea behind this was to have an intuitive and powerful environment which would enable new ways of shape modeling. For example, there might be shapes which are particularly easy to achieve by adding gravity to the scene. We found that most of the time our material would simply flow away and loose its shape - in general it was hard to control. Thus we increased the stiffness of our material. This lead to a numerically instable material simulation. We solved the stability problem and noticed that material which is able to withstand gravity is not adequate for shape modeling: it is too inflexible as the material yields only after high stresses are applied. As we want to apply simple forces in order to deform the material, these two goals are contradictory. It is also important to note here, that a perfectly plastic material, that instantly yields under stress, is fluid. It is not able to keep its shape against any force no matter how viscous the material is: if one exposes the material long enough to gravity, it will eventually deform into a flat shape on the floor. Therefore, at least some elasticity or even rigid material is needed if we want to incorporate gravity. Another important insight we had is that elastic behavior in general is not desired: after the user has deformed a shape she wants it to stay in the deformed state. Elastic material would go back to its original rest state. Therefore, we want plastic behavior at least for all user performed sculpting operations. We found that it is hard to find a material simulation method which provides all the desired properties out of the box:

- Ability to withstand gravity, which is of importance for our other design methods.
- Plastic material behavior, which is suitable for sculpting.
- Real-time simulation, enabling a convenient work flow.

These difficulties are not so surprising if one thinks about how many materials actually exist in nature which provide the desired properties: for centuries there was basically only natural clay which was up to the task. Natural clay is heavy, which leads to a large contact friction and therefore the object does not move if the modeler pushes his thumb into the object, but yields and deforms plastically at the point where the thumb touched the surface.

After we explained to an architecture student what the goal of our system is, he was really excited and told us that he enjoys designing on the computer as he does not have to care about the buildings statics: there is no gravity in his design tool. This triggered the idea of letting the user decide which parts of his model are affected by gravity and which are not. We found that it solves most of the issues related to numerical stability, matter plasticity and the amount of freedom we want to give to the user. Finally, we implemented a slightly modified version of this concept where we try to put everything into context that makes sense to the user.

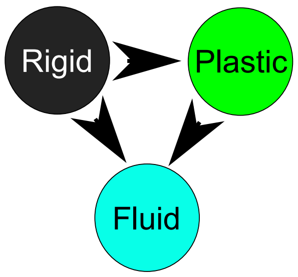


Figure 5.1: The three matter types are illustrated. The arrows state how they influence each other if a collision occurs.

We introduced three types of matter state: *Rigid*, *plastic* and *fluid*. *Rigid material* can only be moved by the user directly. It is not sensitive to any type of force. It does collide with the two other matter types and deforms them. *Plastic material* deforms *fluid material* when a collision happens. It is as if, from the perspective of the *fluid material*, the *plastic material* would have infinite mass. If *fluid matter* collides with an other matter type it does not exert any forces to them but it deforms itself. So for *fluid matter*, the two other types look like rock. For *plastic matter*, only *rigid matter* is solid.

The force based tools can be customized: one can select whether they have an effect on *fluid material*, on *plastic material*, or on both. *Rigid material* is not sensitive to forces at all and can only be moved by the user.

5.2 Tools

In order to implement tools, we rely on the following information:

- two dimensional (2D) mouse coordinates.
- 3D spatial coordinate of the surface point under the mouse cursor (if possible).
- Surface normal of the surface point (if possible).

If it is not possible to compute the spatial coordinates or the surface normal, we do the following:

- The spatial coordinate lies on the plane which includes the last point touched on the surface and is parallel to the screen plane.
- The surface normal is simply not updated anymore until the mouse is again hovering over the surface.

Besides a few special selection tools, all tools use a sphere to implement their behavior. Once a user understood this concept, she can immediately use all tools as they behave very similar: the user activates the tool by pressing the left mouse button. The mouse scroll wheel controls the sphere size which is mapped to a suitable configurable quantity of the current tool. The standard behavior for all tools using this sphere is, that the sphere automatically centers at the point of the material which lies under the mouse cursor. The sphere attaches itself to the surface where the mouse pointer is. This makes sense most of the time. If the user wants to move away from the surface, she can do this in the following way:

1. Move the mouse cursor away from the material silhouette.
2. The sphere is now moved under the cursor in the a parallel plane to the screen plane.
3. If the user enters the silhouette of an object the sphere will again attach to the surface. This can be disabled by pressing the *ALT*-key on the keyboard.

5.3 Selection

Several operations in our system are performed on whole blocks of matter. This includes *translation*, *rotation*, and *deletion*. Therefore we implemented a series of selection tools, which allow one to select a subset of the material and, in a second step, perform a desired operation such as moving or deleting the selected objects. Similar to selection regions in 2D graphic packages, we allow to constrain the effect of tools to the selected material. This behavior is enabled by pressing the *CTRL*-key on the keyboard while using the tool. This is beneficial if work has to be done on material that is spatially close, but one wants to work only on a subset. The user just selects the material she wants to work on and performs the action. If the user holds down the *Shift*-key during the selection process, she can add previously unselected material. If she holds down the *CTRL*-key, she can remove material from the active selection.

5.3.1 Circle and Rectangle Based Selection

The first type consist of the *circle* and *rectangle* selection tools, which are not aware of depth and simply select all matter that lies inside the world frustum of the selection shape. Most users are very familiar with those types of tools and they are useful for certain tasks. However, one has to keep in mind that back-projection of the circle into the world space yields a cone and not a cylinder.

5.3.2 Sphere Based Selection

The sphere selection tool is *surface aware*, meaning the center of the sphere is always on the surface where the mouse pointer is hovering over. This is, as already mentioned, common behavior of most tools. By scrolling with the mouse wheel one can increase or decrease the sphere's size and therefore make selections that go deeper into the material or ones that are more shallow. With the current implementation there is no way to decouple the area of the selection and the depth it enters into the material. We found that such a tool is not necessary, but it might very well be implemented.

5.4 Force

The force tools are user activated and exert force in a specific way on the material.

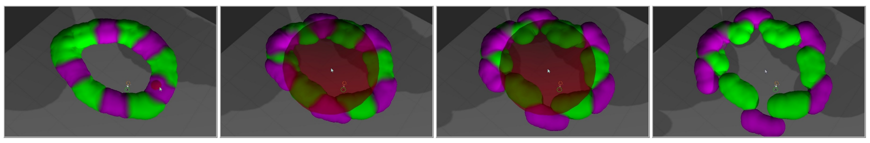


Figure 5.2: The *push tool* is used while holding down the *CTRL*-key to only exert force on the selected material.

5.4.1 Push tool

The *push tool* consists of a *surface aware sphere* and applies a force to surrounding matter pointing radially away from the sphere's center. This tool can be used to dig holes into the material or, with increasing sphere size, to push parts of the object around. It also has a smoothing effect if applied to bumps. Small forces applied to the material can be used to carve small channels or to smooth out ridges. Overall a very versatile tool.

5.4.2 Pull Tool

The *pull tool* is the counter part of the *push tool*: it is *surface aware*, applying force normal to the surface pointing away from the object. Due to the na-

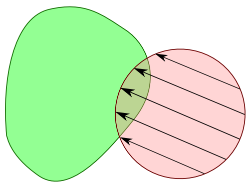


Figure 5.3: Push Tool: force normal to surface tangent plane.

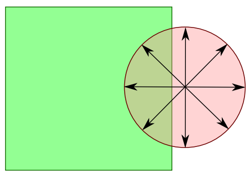


Figure 5.4: Push Tool: Force pointing radially outwards from sphere center.

ture of our underlying physics simulation, which is based on Smoothed Particle Hydrodynamics (SPH), some undesired effects occur: holes inside the object. The material separates easily since SPH has only a spatially limited elasticity. We tried circumvent this issue by simultaneously adding material inside the object. The outcome was too similar to the ordinary *material-addition tool* and thus we did not incorporate this feature.

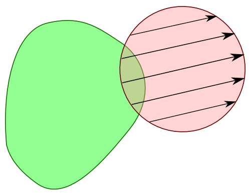


Figure 5.5: Pull Tool: Force normal to surface tangent plane.

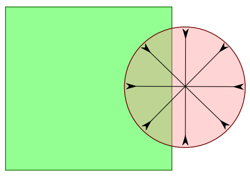


Figure 5.6: Pull Tool: Force pointing radially inwards from sphere center.

5.5 Add and Remove Material

The matter addition and subtraction tools are also implemented using *surface aware spheres*. The user can again control the sphere size by scrolling the mouse wheel.

5.5.1 Matter-subtraction tool

The *subtraction tool* simply removes the particle closest to the sphere center with a given rate. The removal rate corresponds to the sphere size: the larger the sphere, the faster it deletes material. We used it to vacuum unneeded material.

5.5.2 Matter-addition tool

This tool adds matter to the simulation randomly inside the sphere. The sphere size corresponds to the rate of material addition. This is a very useful tool to draw outlines of new shapes, to fill them with matter, or to connect parts of existing shapes. Figure 5.7 shows the tool in action. It behaves a bit like a virtual pencil and we found it to be one of the most useful tools to create shapes.

5.6 Rotation, Scaling and Translation Tools

Rotation, scaling and translation are standard tools in every 3D modeling package. Our system makes no exception and is modeled after the ones known from commercial tools and is very similar to Blender [1].

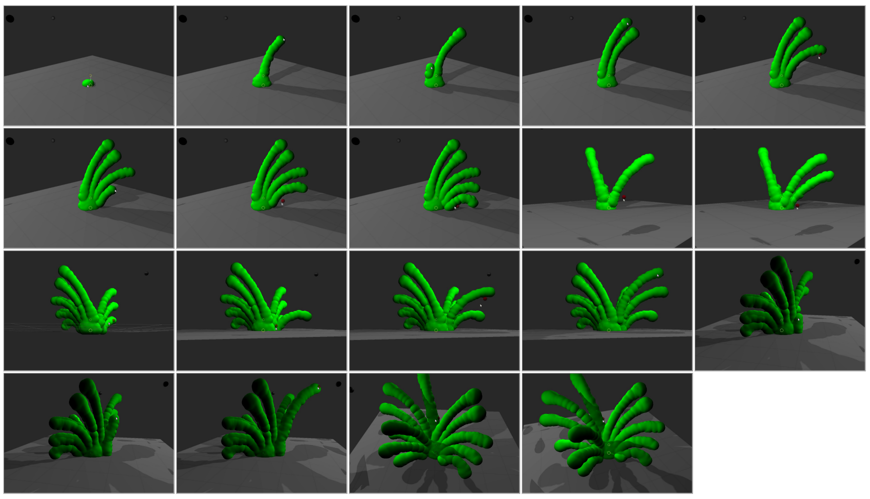


Figure 5.7: The matter addition tool: notice how it adds material in a plane parallel to the screen plane.

5.7 Auto Rotation

A well-known technique from clay work, where one places the object on a rotating table to shape an axially symmetric object, was also implemented. The user can enable a virtual rotation of the system. Now one can use all tools stationary while the material moves. The analogy works quite well and it is a simple and easy to understand tool, which can be handy at times.

5.8 Rigid Matter as Tool

Once matter has been transformed to the rigid state, it is fixed in space and not sensitive to any type of forces anymore. The only way *rigid matter* can be moved around is by using the positioning tools. This behavior encourages the creation of customized tools, which are then used to further shape the object. Time can be saved if the user creates a custom tool designed for a certain purpose. However, the user interface to control a customized tool is not as intuitive as the one for the built-in tools.

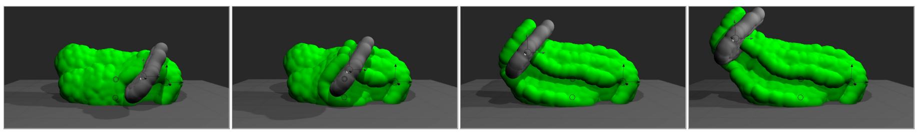


Figure 5.8: Customized tools: simply move rigid matter through plastic matter to leave a footprint.

Chapter 6

Governed Design

6.1 Overview

Besides the classic modeling approach, we implemented a new design paradigm, which we call *Governed Design*. With the classical approach the designer needs to have a clear-cut idea of the object she wants to model. At this point *Governed Design* assists the artist by allowing her to model and explore shapes by preparing a scene and running a simulation on it.

The design process would look as follows: the artist starts with an empty scene and lets material flow into it by using material generators or procedural shape creation. For example, one could start with a structure extracted from a 3D *Perlin-noise* [35] cube. She then places some constraints into the scene. Symmetry planes, attracting force points or material sources are just a few examples on what she could add to the scene. After setting up the scene, she would start the simulation and watch how the system evolves. We believe this method can substantially stimulate creativity, since it provides the designer with a great deal of explorative freedom.

Our software combines the *classical approach* with the method just described. The system helps the user to create interesting shapes by providing tools that influence the material in the scene in a straightforward way. Now there are numerous smart systems that could be used to shape material. The feature distinguishing a usable system from a mere random generator is the predictability of the outcome. The user needs to have an idea of how a certain change in the configuration affects the outcome of the simulation process. Hence, the governing rules we implemented model well known real world concepts, thus making the effect of a configuration change predictable. The user of our software will still be surprised by the shapes generated through her setup, however the behavior is predictable as the material behaves according to physical laws. The results achieved are similar to the real world where forces of nature shape interesting forms.

The above paragraph can only give a limited perspective on *Governed Design*. For further explanation, we thus refer to [20] where the paradigms potential is explored and discussed in more detail.

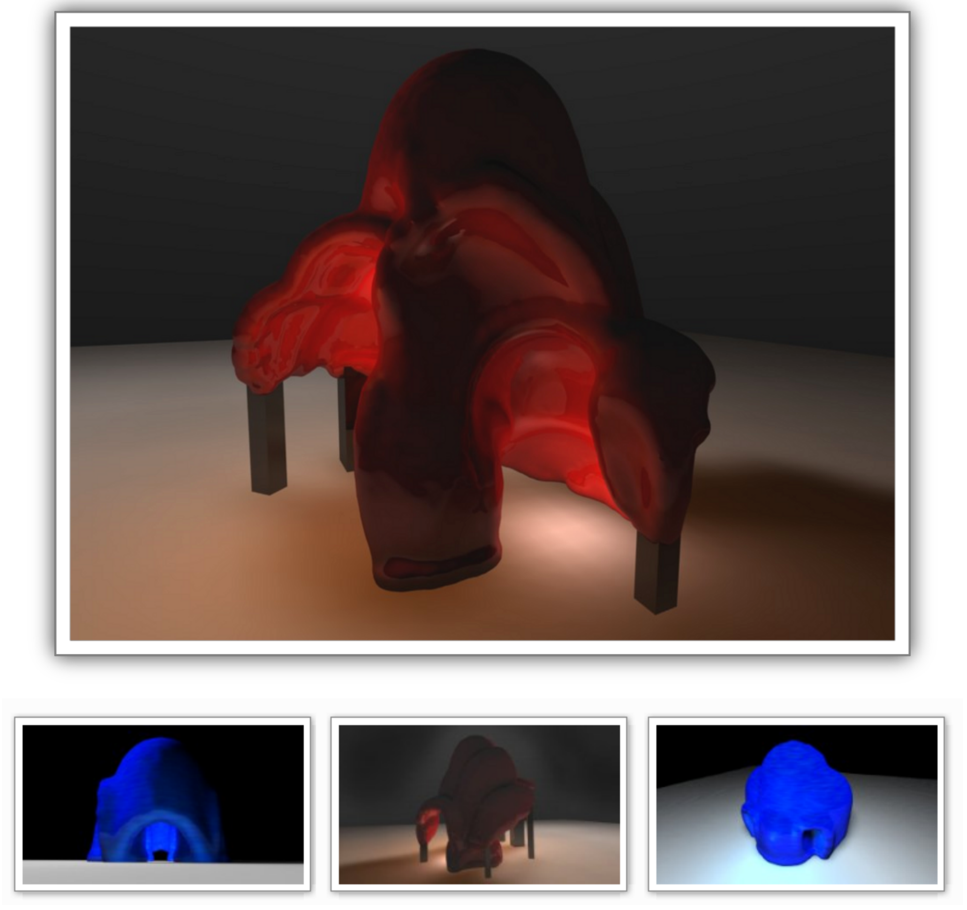


Figure 6.1: A small subset of shapes we created by using our *Governed Design* method.

Chapter 7

Results

To check the practicality of our tool, we modeled a wide variety of shapes using it. In the following section we present the designs we built, using only our *classic tools* and explain how we achieved the results shown and how long it took to model the shape. After finishing a shape, its surface is usually exported from our tool to create a nice rendering. In the following we describe how the tool is supposed to be used:

1. Rough sketches and prototypes are modeled.
2. The surface is exported to a surface modeler, where surface quality is increased and additional detail might be added.
3. An animation package of choice could be used in case the model has to be animated.

If nothing else is stated we used *Blender* [1] to create the images. It is always important to state what skill a modeler has because mostly mastery comes with experience. Our modeling skills sums up to at most a few hundred hours of usage of such tools. This experience comes partly from research we did during the investigation of related tools and the rest is from earlier experience acquired during the years.

7.1 Humanoid characters

Characters are rather easy to create as a single stroke with the *matter addition* tool is transformed to a long volumetric shape in space. Hence, creating animals, aliens and other characters is rather easy and quick. For example, legs can be made thicker by adding material to them using the *material addition tool*. If the user needs a hole somewhere she can use the *matter deletion* tool and therefore this task is a matter of seconds with our tool. In figure 7.1 we show the time series of a character creation. Figure 7.2 shows this caracter from another perspective in our tool.

After creating the basic shape of the character, we exported the surface (figure 7.3) and used the sculpting interface of Blender [1] to add further detail to it. The two arm spikes and the tail spikes are modeled by simply displacing the triangles. Not being able to model high frequencies directly in our tool might be regarded as a drawback, but once an appealing basic shape is found, they can easily be added during a later step. The final result is shown in figure 7.4.

7.2 Genus-8 Sculpture

This sculpture (figure 7.5) was also created in less than a minute. A symmetry plane was added to create order in the otherwise arbitrary drawn strokes. The symmetry plane works wonders and the shape looks (at least to the authors) appealing.

The force-based push tool was used to punch a hole. A connection was added with an additional stroke, creating another hole. A smoothing step was applied after exporting the shape to reduce artifacts introduced by our iso-surface generation. Subsequent re-meshing worked flawless, typical loss of detail for this method was not an issue because there was no detail to be lost on this model.

7.3 High Genus Shapes

In this work (figure 7.6), the tool's capability to model high genus shapes is taken to the limits by using over 50k fully simulated particles. This model was created by selecting a suitable starting point on the object's surface and connecting it with a smooth stroke to another part of the object. The *matter addition* tool feels a bit like a three dimensional pencil. This procedure yielded a three dimensional (3D) spider web structure. After the object was exported from our tool, a plane was added and a movie was rendered where the camera flies between the branches and zooms out. A few cutouts of the movie are presented in figure 7.6.

7.4 Canyon Landscape

This scene (figure 7.7) is different in the sense that canyon was not modeled directly, but by modeling the air.

After about 5 minutes, during which we were experimenting a bist, the shape presented here, was on screen. After the iso-surface extraction, we simply flipped the normals and rendered an image with a little fog to give a better impression of distance. That is all there is to say about the creation process of this shape but there are a few things to note here:

1. The *natural* bridges are really easy to model by punching a *hole* into the air.
2. Depending on the size of the spheres, high frequency can occur at the intersection segment of spheres.

While the approach of modeling the *negative* shape is in general useful, the way high frequencies are created may only be beneficial for certain types of shapes. If these high frequencies are not desired, larger spheres or post processing has to be used to smooth them out.

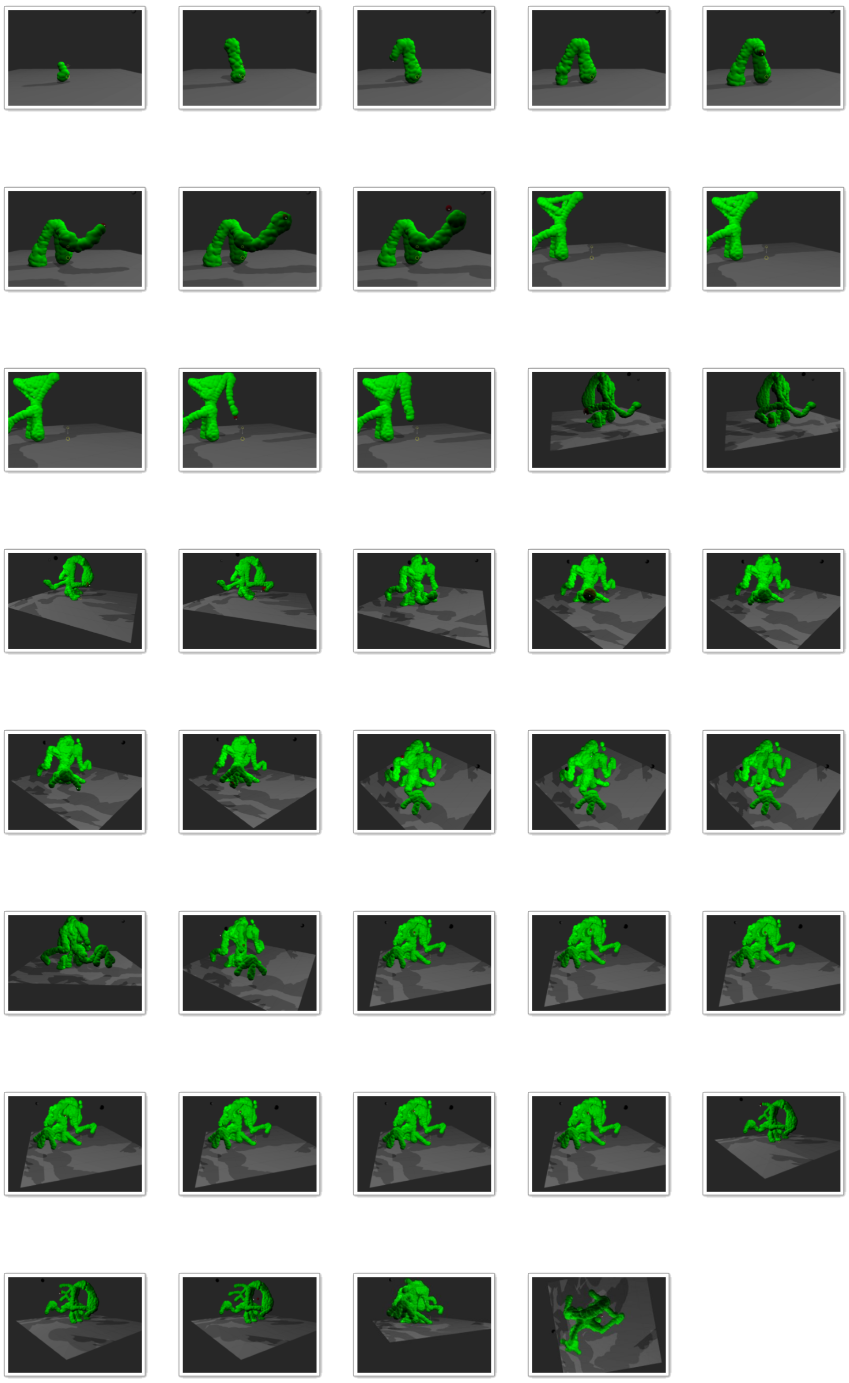


Figure 7.1: Humanoid alien character prototype: modeled in 3D in less than 5 minutes

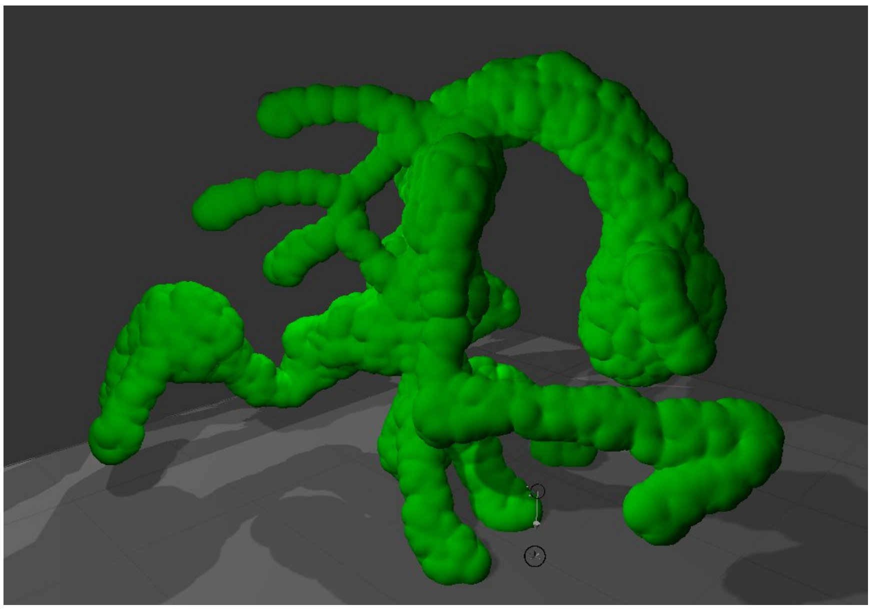


Figure 7.2: Humanoid alien character intermediate shape: the shape in our tool

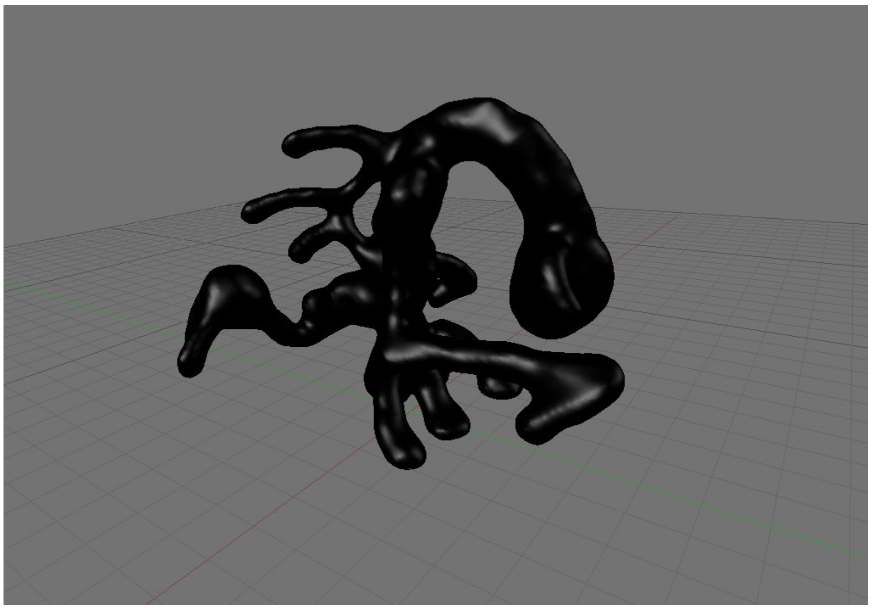


Figure 7.3: Humanoid alien character intermediate shape: output from our tool in Blender [1]

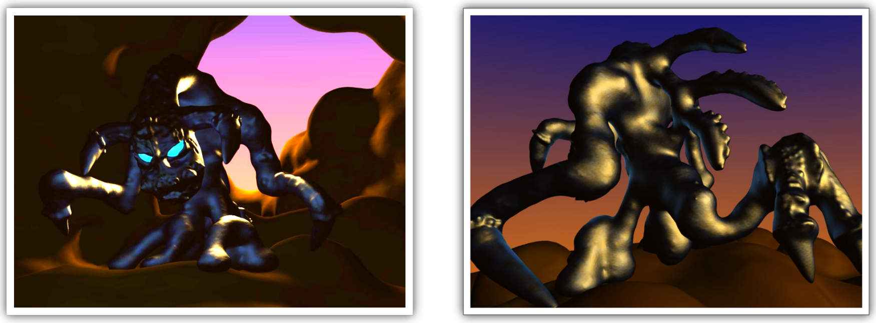


Figure 7.4: Humanoid alien character final shape: surface detail added and rendered

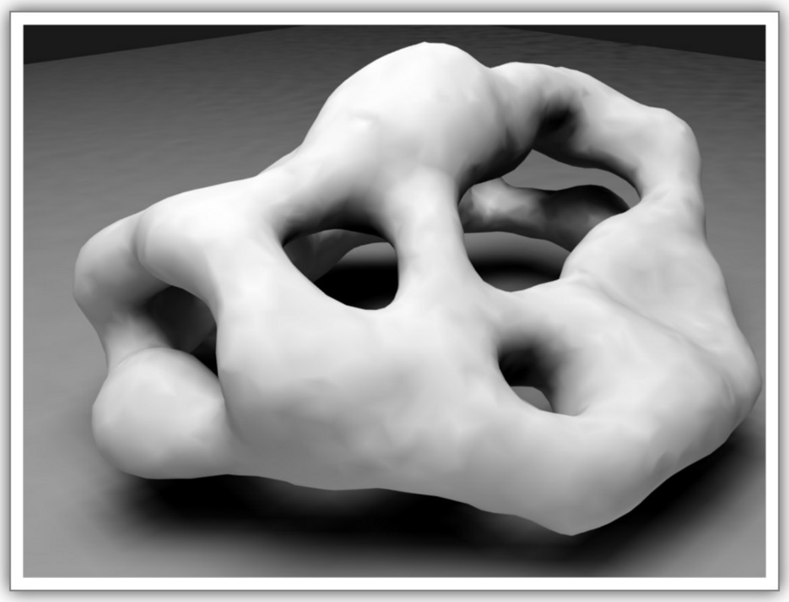


Figure 7.5: Complex topology can easily be modeled with our tool.

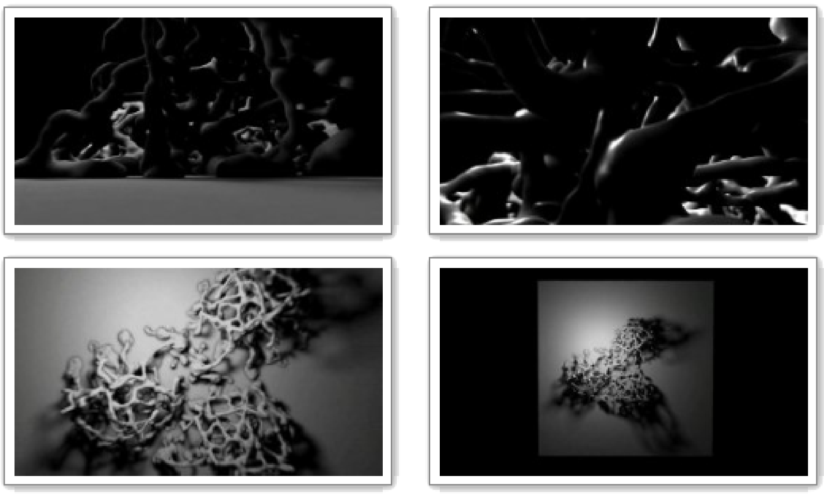


Figure 7.6: Cutouts of the movie featuring our high genus object.

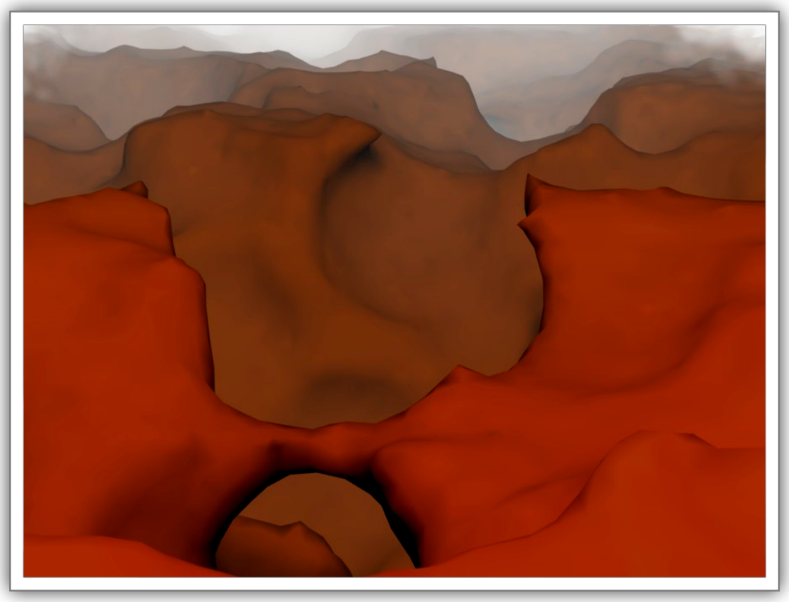


Figure 7.7: The canyon is created by modeling the *air*, not the canyon. High frequency exists due to intersection of spheres.

Chapter 8

Implementation

8.1 System Overview

This chapter gives an overview of the high-level structure, design decisions and technologies used for *Quiqshape*, the software developed in this master thesis.

The application is written from scratch. When we chose which technologies to use, we put special emphasis on portability: All libraries we use are cross platform. Even though we only tested *Quiqshape* on Linux and Windows, it should run without code modifications on MacOS X and other Unix variants like FreeBSD. To implement *Quiqshape* we used:

- C++ as the programming language. We chose this language for performance reasons, and because libraries we wanted to use are written in C++.
- *OpenGL* for visualization. Shaders are written in OpenGL Shading Language (GLSL).
- Qt [7] for the graphical user interface. Qt offers high portability and a simple and productive API. It also allowed us to easily create a modern and flexible user interface with menus, toolbars and a docking architecture. This allows the user to customize the user interface as desired.
- OpenMP [8] is used for loop-level parallelization. OpenMP is not vendor specific and is supported by many compilers.
- Boost [2] for serialization of objects. As a highly optimized and portable C++ library, *Boost* is widely recognized.
- Design by Contract (DbC) [3] to properly specify interfaces and help debug the program.

OpenMP and Boost are optional dependencies, *Quiqshape* can be compiled and run without them. However, saving and loading features are disabled when compiling without Boost. All used libraries are freely available as open-source project.

8.2 *OpenGL* Issues

We wrote our own library to support DbC. The library supports pre- and postconditions as well as assertions. When breaking a contract at runtime, the developer can either step into the code, ignore this time, or ignore forever. DbC was especially useful when working with the *OpenGL* state machine. *OpenGL* functions often have many implicit preconditions. For example, before calling the function that writes a uniform parameter of a shader, the shader has to be activated. While developing, it is often hard and error prone to keep track of what the *OpenGL* state machine looks like at a given time. This can easily lead to situations where a call does not behave as expected, and debugging *OpenGL* is tedious. Using DbC, we can specify in a precondition in what state *OpenGL* has to be before calling this function. This makes it easier to develop and find bugs, and is a good specification about how this function is to be used.

8.3 Custom Preprocessor

Quiqshape depends on several shaders read from GLSL files and simple objects (like spheres and cubes) read from OBJ files. In order to be able to build a standalone binary we wrote a preprocessor that compiles all files into the application.

8.4 Save and Load

To save and load scenes created in *Quiqshape*. we are using the Boost serialization library. We do not serialize whole objects, but only the fields required to correctly load the entire scene again. We support serializing objects of type `Object` and all its subtypes (see the class diagrams in the appendix). This allows us to save all objects, including the symmetry plane, the matter generator and so on.
For particle systems, we only save arrays containing properties like positions, velocity and temperature. Additionally, physics and visualization parameters are saved. We have implemented two different types of formats:

- A binary format which allows efficient saving and loading using less space.
- An Extended Markup Language (XML) based format that is easy to understand.

In addition to our Wavefront OBJ [9] exporter, the XML format ensures a good integration into existing pipelines because one can easily write an custom loader to import our format.

8.5 Parameter Saving

When saving a scene, physics and visualization parameters are also stored. Additionally, the user can store and load sets of parameters independently of the open scene. The parameters are then stored in a hidden directory in the user's home directory.

8.6 Auto-save

The simulation in *Governed Design* is in general not real-time. To allow the user to let the simulation run without the need to constantly keep an eye on it, we implemented auto-save functionality. The function is accessible through the File menu, and the user can select the interval, in minutes, in which the scene will be saved.

8.7 Recording

To capture the dynamic process of a *Governed Design* simulation that did not run in real-time, we have implemented recording functionality where the user can select to save every n-th frame.

8.8 Marching Cubes

Apart from the sphere splatting algorithm for real-time visualization of the Smoothed Particle Hydrodynamics (SPH) simulation, we also have an iso surface extraction using marching cubes (MC) [32]. The algorithm produces a triangle soup, which we can export to Obj format [9].

Using MC we had two choices:

- Extract a level set of the smoothed density field in order to generate a nice and smooth surface
- Approximate the iso surface with the union of spheres, yielding the same surface as with sphere splatting.

We decided to do the later, with the advantage that what the user sees on the screen is very close to what he exports. This results in a simple implementation to calculate the iso value at a point in space: it is the distance to the closest particle. The target value for the surface is then the user-defined sphere size, which is also used for rendering.

Our MC implementation is based on a freely available C implementation by Cory Bloyd [15]. We have adjusted and extended the implementation in several ways:

- We switched from C to C++ and removed the GLUT [4] dependency.
- Instead of a cube, the algorithm works using an axis-aligned box. It also finds the smallest enclosing box, which gives a significant performance boost over the original implementation for most shapes.
- The cell size adaptively changes with the user defined sphere radius. The minimum size of a cell is $2 * \frac{radius}{\sqrt{2}}$. This guarantees that no particle can be contained in a cell without touching any corner, which means that no particles will be overlooked when extracting the surface.
- Before extracting the surface, the user can choose between quality levels *low*, *medium* and *high*. Using *low* quality, the minimum cell size is used, giving simply a octahedron for a single particle. When using *high*, 10*10*10 times as many cells are used, giving a good approximation of a sphere.
- Parallelization of the code using *OpenMP*.

The time to extract the iso surface increases with decreasing sphere radius and higher quality. Because this would lead to arbitrarily long computation time for arbitrarily small sphere radius, there is a hard-coded limit for the maximum number of cells (currently 128x128x128).

8.9 Parallelization

This section explains how we parallelized our code and demonstrates the speedup achieved.

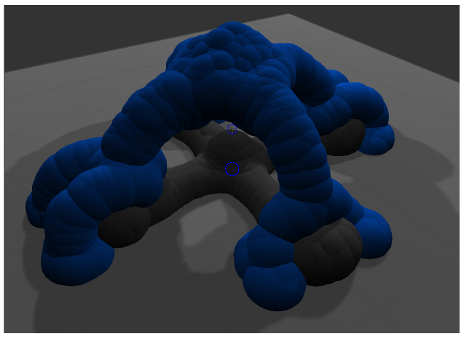


Figure 8.1: Scene rendered using sphere splatting with large sphere radius

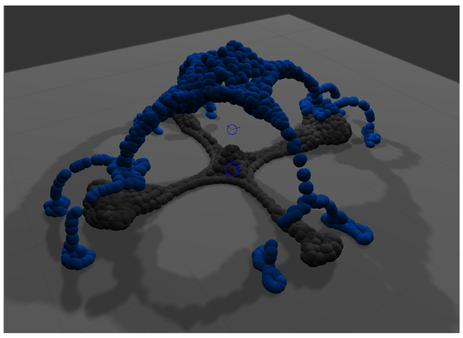


Figure 8.2: Scene rendered using sphere splatting with small sphere radius

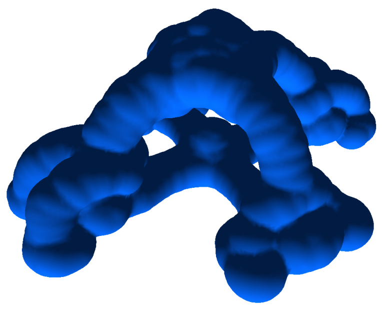


Figure 8.3: High quality surface extraction, 128x120x60 cells

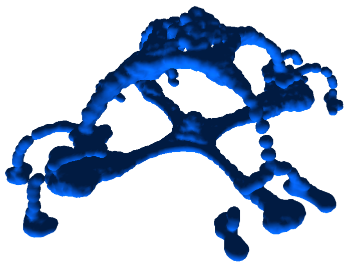


Figure 8.4: High quality surface extraction, reaching the maximum cells limit of 128x128x128

8.9.1 Simulation Code

In order to speed up the actual simulation code, we exploited data parallelism or loop-level parallelism inherent in our code. We used OpenMP [8] to generate parallel code out of our main SPH loop. In order to avoid race conditions while updating the computed derivatives we used thread local storage and an additional reduce step at the end. The alternative would have been to use locking. The benchmarks we made showed that locking was significantly slower than thread local storage and merge (see section 8.10.2).

We added a small abstraction layer so that one could easily switch to other similar implementations like Intel's thread building blocks [5].

8.9.2 Rendering

Initially, rendering and simulation did not run in parallel. This resulted in a rather bad CPU usage, because when rendering, only one core is active while all others are idle. This was especially noticeable when using high quality rendering, which takes a significant amount of time.

For this reason we introduced the usage of Qt's threading infrastructure to run the Graphical User Interface (GUI), the visualization and user input handling in one thread, and the physics subsystem in another thread. The physics subsystem still uses OpenMP. A rough measurement showed that using separate threads for graphics and physics increased CPU usage on an 8-core machine with about 50'000 particles from about 40% to 80%. On future systems with more cores, this

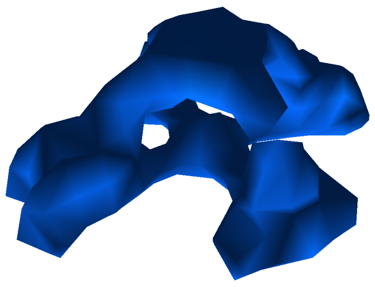


Figure 8.5: Low quality surface extraction, 15x12x6 cells

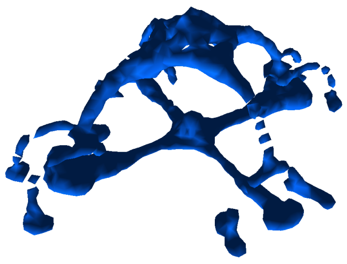


Figure 8.6: Low quality surface extraction, 54x44x20 cells

split will become even more significant.

8.9.3 Marching Cubes

The marching cubes algorithm was also parallelized. The queries to get the iso surface run in parallel. However, when actually building the triangles for the mesh, we use OpenMP's *critical section* to make sure only one thread at a time adds triangles and normals to our mesh data structure. See section 8.10.1 for some benchmarks.

8.10 Benchmarks

8.10.1 Marching Cubes

For the marching cubes benchmark, the algorithm runs on the highest possible grid resolution of 128x128x128 cells and 10'000 particles ordered in a sphere. The benchmark was run five times and the average was taken. The results on two different systems were as follows:

Configuration	Intel Core 2 Quad CPU with 4 cores at 2.40GHz and 3.24GB RAM, running Windows XP
1 core	3:07min, average of 28% CPU usage
4 cores	0:55min, average of about 95% CPU usage (100% for most of the time when 4 cores are running, dropping to 75%, 50% and 25% towards the end when one or more threads are finished)

Table 8.1: Speedup of about 3.4 for MC on a quad core

Configuration	Two Intel Xeon CPUs with 4 cores each at 2.8GHz and 2GB RAM, running Windows Vista
1 core	3:05min, average of of 13% CPU usage
8 cores	0:34min, average of about 95% CPU usage (same pattern as with quad core)

Table 8.2: Speedup of about 5.4 for MC on an 8-core

8.10.2 Updating Derivatives

During the computation of system properties *data race conditions* may occur. There are several ways to solve the issue and different resources suggested different solutions. Therefore we setup a set of synthetic benchmarks to base our decision on empirical data. We chose a synthetic setting in order to reduce the side effects the system is exposed to and to make the benchmark more traceable. The following benchmark compares different methods to update the same data using several threads (in our case: several threads updating the derivatives in our main simulation loop). The methods are:

- *Thread local storage + reduce step*: Each thread writes its own copy of the data. At the end the data is merged. Each thread randomly accesses and writes data.
- *Thread local storage + reduce step*, *cache friendly*: Same as above, with the exception that data is only read from locally close data yielding better cache access patterns.
- *Particle locking*: The data of a particle is locked before writing to it.

Those synthetic benchmarks were used to evaluate the best implementation strategy. We wanted to know whether it is better to use locking or separate storage per thread, and whether optimizing the memory access pattern would be worth the effort.

The cache-friendly code in the benchmark accesses block sizes of 1024 particles, each of which has three doubles per vector, yielding an array that fits into the L1 data cache of each core of our system (see table 8.3): $1024 \cdot \texttt{sizeof(double)} \cdot 3 = 24KB$.

If the *cache friendly* strategy gave a very high performance boost, this would suggest that the algorithm should work on the *kd-Tree* directly and not on *arrays* of the data, because the *kd-Tree* groups spatially close particles to the same memory region. In order to check whether memory bandwidth or Central Processing Unit (CPU) cycles are the limiting factor we implemented two versions of our algorithms:

- Memory bandwidth intensive, almost no computation per data element
- Computationally intensive, many floating point operations are performed per data element

Then we compare how the performance scales with more cores between the two algorithms. The gained insights help to better understand the behavior our real system.

The benchmarks were done on this system shown in table 8.3.

Entity	*Specification*
Name	Intel Xeon E5462
Codename	Harpertown
Lithography	45 nm
Socket	771 LGA
L1 Data Cache (per core)	32 KB, 8-way set associative, 64-byte line size
L1 Instruction Cache (per core)	32 KB, 8-way set associative, 64-byte line size
L2 Cache (two per processor)	6144 KB, 24-way set associative, 64-byte line size
Random Access Memory (RAM)	4 * 1024 MBytes, PC2-6400

Table 8.3: Used CPU for the following benchmarks

The setup for the benchmark is shown in table 8.4.

Number of particles	100'000
Number of threads	8
Number of iterations (to reduce error)	100

Table 8.4: Benchmark attributes

8.10.2.1 Results

Algorithm	*Time in seconds*
Thread local storage + reduce step	1.35
Thread local storage + reduce step, cache friendly memory access	0.72
Particle locking	12.53

Table 8.5: 100 iterations, no cache flush

Algorithm	*Time in seconds*
Thread local storage + reduce step	1.37
Thread local storage + reduce step	1.01
Particle locking	12.45

Table 8.6: 100 iterations, cache flushed between individual measurements

The benchmarks in tables 8.5 and 8.6 clearly show that *thread local storage* with an additional *reduce step* afterward is better than *particle locking*. Therefore we used *thread local storage* in our system. Even though a cache friendly access pattern yields better results, we decided that it was not worth the effort. Figure 8.7 shows how these algorithms perform with increasing number of cores.

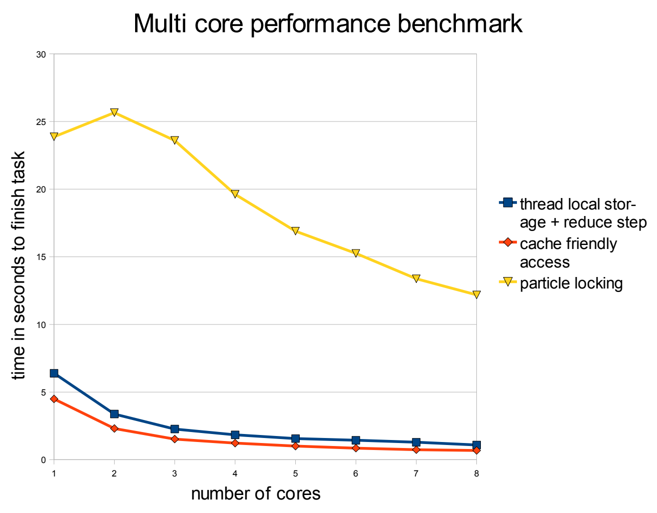


Figure 8.7: Clearly thread local storage with an additional merge phase outperforms a locking based approach.

With thread local storage we achieve a speedup of about 5.4 using 8 cores, which is a bit disappointing. The probable cause of this behavior is illustrated in the two benchmarks shown in figures 8.8 and 8.9. Most likely we use too much memory per loop, which leads to a congestion of the memory bus and a pollution of the data caches.

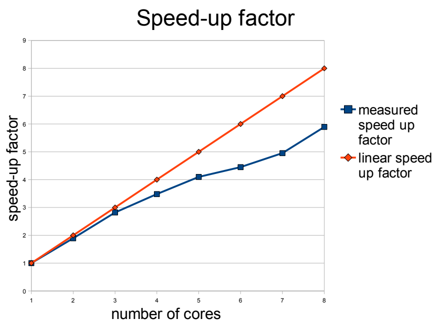


Figure 8.8: Only simple computations per particle: a sub-linear scaling is typical for bandwidth limited algorithms.

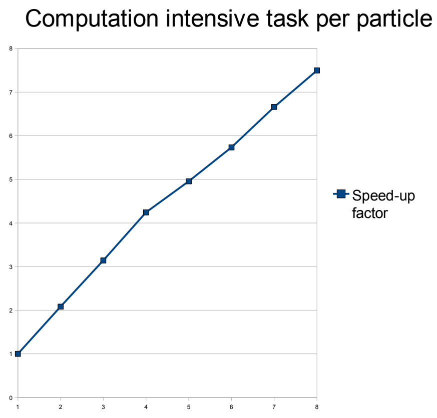


Figure 8.9: Complex numerical computation per particle: This computation intensive benchmark shows that linear speedups are achievable. For less than 5 cores there is a super-linear scaling which is probably due to implicit data prefetching into the four L2 caches.

Chapter 9

Summary and Conclusions

We set out to improve the early design phase of geometric shapes. The benefits of a modeling tool that uses a material simulation became obvious. It allows more powerful modeling techniques and methods to be used. Our tool tries to channel the augmented creativity found in the symbiosis of tool intelligence and human guidance into uniquely new shapes.

9.1 Classic Modeling

The *classic modeling* approach based on volumetric models is not new. However, few experiments are made where the underlying material is driven by physically based algorithms to mimic real-world material. We believe that the approach is helpful in modeling, as it gives a very intuitive way of how to create a model. Our experience while working with *Quiqshape* has shown that many kinds of shapes are indeed easy to model. The tool is also easy to use for newcomers, who can immediately understand the concept of creating, deleting and pushing matter. *Classic modeling* is very useful by itself, but also complements *Governed Design*, by allowing the user to modify and tweak a generated shape. *Classic modeling* can be compared to a pencil tool in a 2D drawing program, while *Governed Design* could be compared to a filter, for example one that generates a Perlin Noise texture, from which the user wants to create a height map.

We believe that many types of objects can be created faster using *Quiqshape* than in a surface-based tool, and we think that the main reason preventing a volume-based modeling tool from widespread usage is mainly the lack of enough computation power. However, this will change in the near future.

9.2 Governed Design

The paradigm of *Governed Design* works well, and we have created a set of interesting shapes. There are many more combinations of the tools we provided, which would result in other kinds of shapes. It is up to artists to explore these combinations. The shapes we have created have a rather organic look due to the underlying representation.

Our experience has shown, however, that creating nice shapes is not trivial in *Governed Design*. There is a chance that the resulting shapes brake apart or do not look interesting at all. A user will therefore need some time to get used to this method of designing shapes before she will be able to create interesting shapes. Also, *Governed Design* is in general not real-time, which causes a rather slow process of creating a shape. We believe that *Governed Design* could greatly benefit from being real-time.

9.3 Future Work

Many of the goals we defined have been reached. Some of the issues we tackled are now a lot easier to solve, like the ability to prototype topologically complex shapes. The overall amount of surface detail that can be represented in our system currently is lower than what is possible with mesh based surface oriented tools. However, as our tool is geared towards the support of early prototypes, which usually do not need a lot of detail, this fact does not become a major show stopper. We see our software as an improvement for the early design phase. It is not thought to be used for the final polishing of an otherwise mostly defined shape.

In the following sections we describe what we believe would be the most promising ideas for further pursuit.

9.3.1 General Purpose Modeling

With about 100k particles we are already able to model some quite detailed shapes. If our approach to geometric modeling is to be used for arbitrary complex shapes, we would like to model with tens of millions of particles.

This could be achieved by running the material simulation only where significant changes happen, for example where user interaction happens, and by using an adaptive particle system to simulate large volumes. Also, the rendering would have to be optimized to use culling algorithms, and to incorporate Level of Detail (LOD).

9.3.2 Input Devices

If the goal is to get closer to actual clay modeling, the usage of a three dimensional (3D) input device, possibly with haptic feedback, could be explored. The tools which are most usable in the current version are rotation-invariant: all built-in tools are based on a sphere reducing the six degree of freedoms of a rigid body's orientation to only three. While two dimensions can be controlled by a mouse-like input device, there remains only one which is determined by the system automatically. However, as soon as one starts to use self-made tools, based on objects of *rigid material*, control becomes inconvenient and cumbersome. Therefore, one could either implement a better user interface for such tools or use a 3D input device with six degrees of freedom.

9.3.3 Round Trip With Mesh-Based Tools

Even though our main goal was to improve the early design phase of an object, it might be desirable to load existing triangle meshes and sample them with particles. This would enable to have a full round trip integration with surface based modelers, which usually are able to import and export triangle meshes.

9.3.4 Surface Detail

The ability to model detailed surface directly within the tool could also be desirable. If one only slightly changes the shape, those details would be preserved as good as possible. If large deformations occur, including potential topology changes, it is understandable that some or all surface details might be lost.

9.3.5 Automatic Inter-Object Collisions

We chose a Lagrangian (particle based) approach to matter simulation. We are simulating material which can split itself up, yielding multiple objects, or which can be combined to one new object. Given a particle p_i, which object does it belong to? Our formulas are currently not ware of this notion: we do not explicitly track objects. To address this issue, we introduced *rigid*, *plastic* and *fluid* systems. However, with a more general approach that tracks objects, more techniques for modeling could be discovered. For example, it could become possible to take two plastic objects, and throw them at each other. In our simulation, the objects would merge, which is not real-world behavior.

9.3.6 Governing Global Properties

There could be more different kinds of *governing properties* besides symmetry planes and grids. For example, one could place boxes in space and link them together, inducing symmetrizing or anti-symmetrizing forces in certain regions.

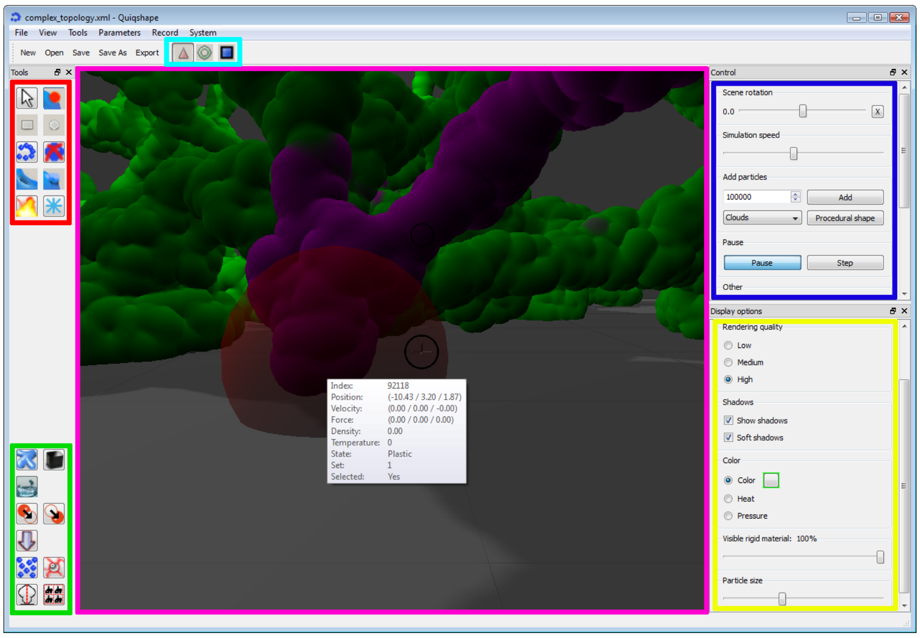
Appendix

User interface

Figure A-1 shows a screenshot of our tool. The basic usage of the program can be described as follows:

The tools in the classic modeling section can be used to select, add, remove, push, pull, melt and freeze matter. Items can be selected by either clicking on them or by using one of the selection tools. All items except the matter have a small circle around them. This is to make them easily selectable, even if the item is hidden behind something else. Items can either be simply dragged with the mouse, or when they are selected one of the translate, rotate and scale tools can be used on them.

The first three buttons in the *Governed Design* section convert matter between plastic, rigid and fluid. The two buttons below these move matter to set 1 or set 2. Those two sets can be independently shown or hidden by changing the visualization settings.



```
Index:        92118
Position:     (-10.43 / 3.20 / 1.87)
Velocity:     (0.00 / 0.00 / -0.00)
Force:        (0.00 / 0.00 / 0.00)
Density:      0.00
Temperature:  0
State:        Plastic
Set:          1
Selected:     Yes
```

Scene rotation: 0.0

Add particles: 100000; Clouds

Rendering quality: Low Medium [✓ High]

Shadows: [✓ Show shadows] [✓ Soft shadows]

Color: [✓ Color] Heat Pressure

Visible rigid material: 100%

Figure A-1: Screenshot of the user interface of *Quiqshape*, the software developed in this master thesis. The marked elements are: Classic modeling tools (red), governed modeling tools (green), 3d view (pink), Translate/Rotate/Scale (cyan), scene control box (blue) and visualization settings (yellow)

The remaining tools add new items to the scene: gravity, matter generator, force point, symmetry plane and symmetry grid. Several of those items will present the user with a dialog first, to specify some default values. In this dialog, the user can also select to generate several matter generators and force points at once. The scene contains a 3D cursor (white arrow) where newly created items are put.

To navigate in 3D space, the user can use the middle mouse button to rotate around the center of rotation (green circle, can be placed arbitrarily) and the scroll wheel to zoom. The camera can be translated using Shift+Middle Mouse and rotated around its center using Ctrl+Middle Mouse.

Code Sample

For all physical processes in our simulation, whether they are simulating Smoothed Particle Hydrodynamics (SPH), calculating collisions between fluid, elastic and solid, to transferring heat, the same basic code is used:

- Loop through all particles
- For each particle, find its nearest neighbors
- For each neighbor, calculate something, usually depending on the distance to the neighbor

The code for this looks as follows (including parallelization using OpenMP):

Code Listing 9.1: Usual loop through all particles

```
// Number of neighbors queried
int num_of_neighbors = 20;

// Loop over all particles;
// Parallelize loop using OpenMP
#pragma omp parallel for
for(int i = 0; i < size; i++) {

  // Check if the current particle is used
  if (used[i])
  {
    // Vector that will contain indices of neighbors
    std::vector<int> neighbor_indices(num_of_neighbors);

    // Query neighbors (maximum 'num_of_neighbors')
    // of particle i at position position[i],
    // store neighbor indices in 'neighbor_indices'
    nearestInRadiusQuery( position[i],
                          radius,
                          num_of_neighbors,
                          neighbor_indices);

    // Loop through neighbor indexes
    std::vector<int>::iterator iter;
    for (iter = neighbor_indices.begin();
         iter != neighbor_indices.end();
         iter++)
    {
            // Get position of neighbor
            Vector3 neighbor_position = position[*iter];

            // Find distance to neighbor
            Vector3 offset = neighbor_position - position[i];

            // Do something with the neighboring particle.
            // Examples:
            // - Calculate Lennard-Jones forces
            // - Calculate SPH forces
            // - Diffuse heat
            YOUR CODE HERE
        }
    }
}
```

The array `position` is a large array, which usually is not fully used. For this reason there is another array `used` which indicates whether a position is used or not. The reason for this implementation was to be able to easily delete particles at almost no cost, as well as to be able to easily keep neighbor indexes over several timesteps. An alternative would be to have a compact array where particles are rearranged when particles are deleted.

System overview

Objects

The most important GUI classes are `MainWidget`, which manages all menus, buttons and keyboard input, and `GLWidget`, which shows the OpenGL content, handles mouse input and initiates the physics calculation and the rendering.

All visible objects are of type `Object`. The inheritance structure can be seen in figure A-2. The most important facts about the types of objects are:

- `SystemObject`: Objects of this type (for example, the lights, the grid, the camera) are not saved with the scene, but created independently. `ParticleSystem` is also a `SystemObject`, because we do not serialize it as an object, but handle it in a custom way.
- `WorldObject`: All these objects can be serialized and saved. They include symmetry planes, force points and so on.
- `EasySelectObject`: These objects have a small, always-visible circle around them, which makes it easy to select them even if they are hidden behind other objects or material.

Each object can have one or several identifiers (managed by the `NameObjectMapper`), which are used for picking. Most objects only have one identifier. Others, like the `ParticleSystem`, manage a set of objects, and are in charge of a range of identifiers. Therefore, for all actions in our system (like deleting and object and selecting and object), the command includes the object the action should apply to as well as an identifier. The object can then determine what to do with the action (for example: the `ParticleSystem` only deletes a specific particle).

Objects can implement one or several of the following interfaces: `TranslationSupport`, `RotationSupport`, `ScaleSupport`, `SelectSupport`, `TooltipSupport` and `ObjectDeleteSupport`.

Tools

Every tool in our system inherits from `AbstractTool`. A tool can be updated, receives mouse and keyboard input, and has a draw function. Tools are managed by the `ToolManager`. The inheritance structure of the tools can be seen in figure A-3.

Updates

An object in our system can register itself for one or several actions, which can be seen in figure A-4. The order in which these actions are executed is:

- `IPreUpdateClient`: Clients are notified before the actual update step. This step is also called when the scene is paused.
- `IUpdateClient`: Theses objects have an update function, which gets called at the beginning of every step of the simulation.
  The `AbstractMatterSystem` (which is the actual implementation of the particle system) also gets updated this way. The update does not get called when the scene is paused.
- `IPostUpdateClient`: After the simulation and integration step, this step can be used to directly reposition particles, but cannot be used to apply forces on the system. The function gets called even if the scene is paused.
- `IResetClient`: Called by each particle system as the first thing in the update function.
- `IForceSource`: After the simulation step of the particle system, but before the integration, force sources are queried for external forces acting on the systems. `IForceSources` are called from the particle systems.
- `IParticleSource`: At the end of of each update of each particle system, clients can add particles in this step.

Global objects and functions

We have several globally available Objects and functions:

- `x744Application`: The main application. This object gives access to most other classes.
- `QMainWindow`: The Main window.
- `GLWidget`: This widget shows the OpenGL output and initiates drawing and updating.
- `MainWidget`: The widget that manages all buttons and keyboard input.
- `AbstractCamera`: Currently active camera, can be queried for various things like position and viewing direction.
- `Scene`: The scene manages all `Objects` (`SystemObject` and `WorldObject` are separate), and initiates save and load.
- `PhysicsEngine`: The physics engine is responsible for updating the simulation.

- `Renderer`: The renderer manages drawing of the `Scene` into the `GLWidget`.
- `ObjectContainer`: There are two `ObjectContainers`, one with `WorldObjects` and one with `SystemObjects`. The scene only stores objects of type `WorldObject` when saving.
- `SelectedObjectsManager`: A list of the currently selected objects.
- `Pointer3d`: Object determining where new objects are placed in the world.
- `CenterOfRotation`: Object determining around which point the camera rotates.
- `PhysicsParameters`: Parameters (target pressure, damping and so on) used in the physics simulation.
- `MatterVisualizerParameters`: Visualization parameters (sphere size, particle color and so on).
- `FileDialogParameters`: Paths that were previously used in the save, open and export dialogs.
- `object_from_key(ObjectName key)`: Function to find and object from a unique identifier. `ObjectName` is the type of the unique identifier; in the implementation this is an `unsigned int`.
- `void set_progress(int percent)`: Function to set the progress of the progress bar visible at the bottom of `GLWidget`.

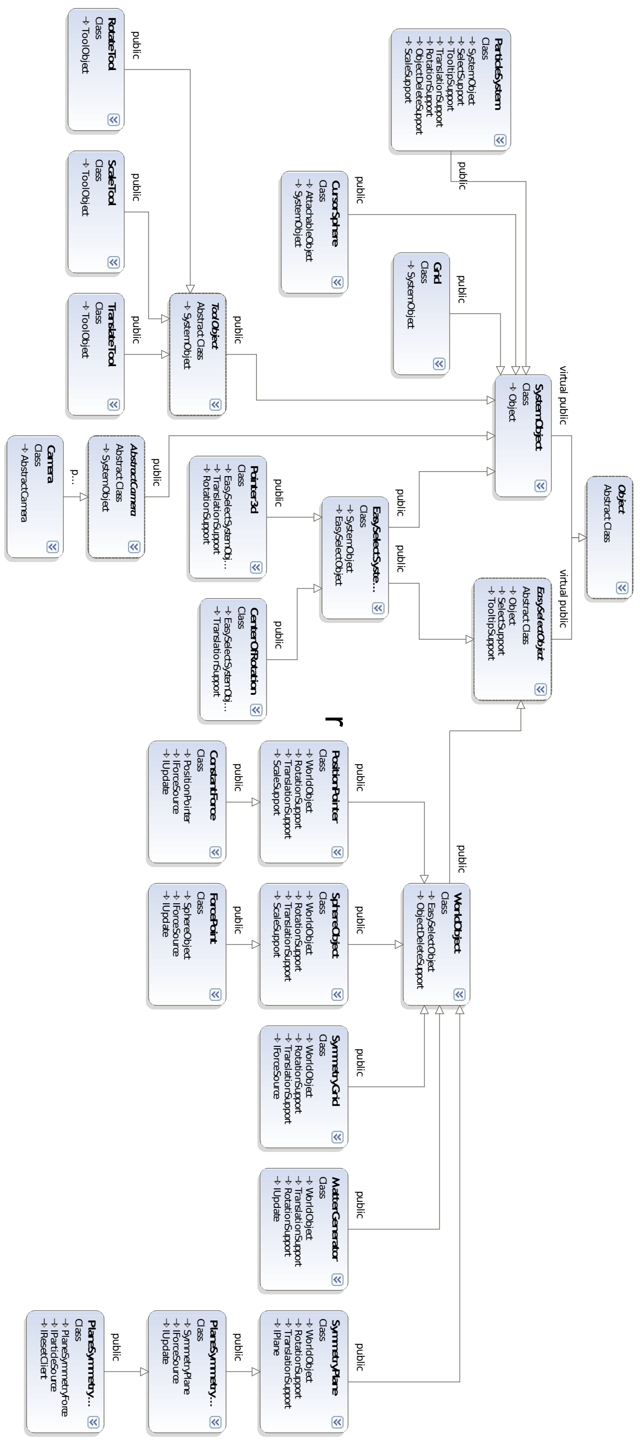


Figure A-2: Object hierarchy

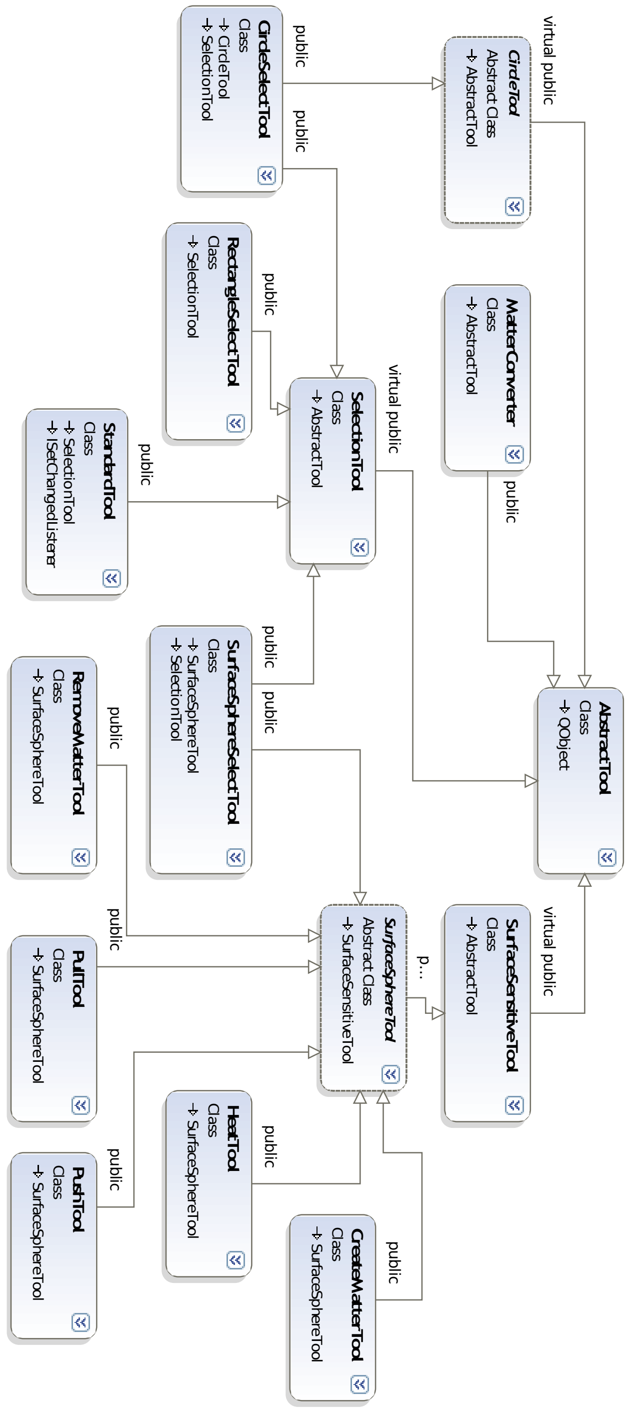


Figure A-3: Tools hierarchy

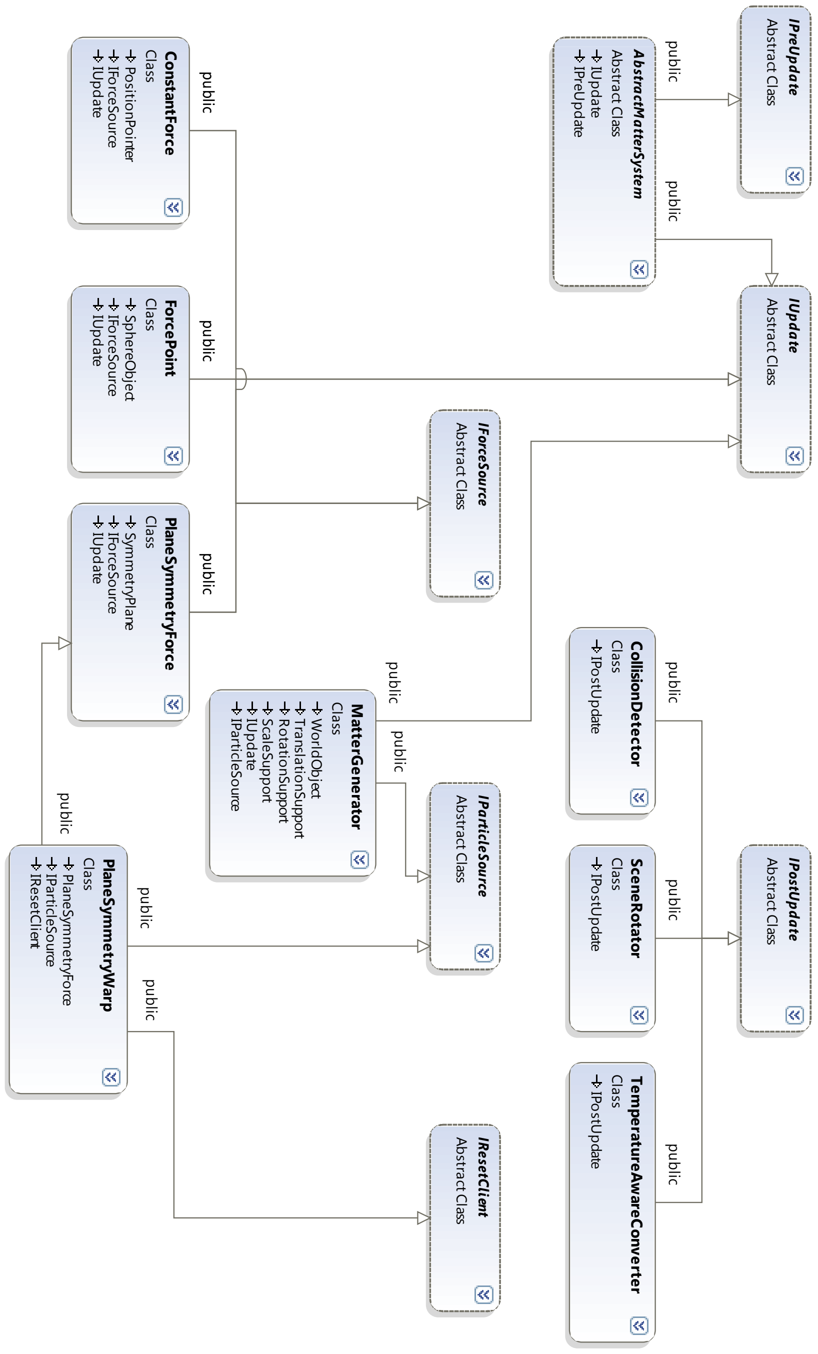


Figure A-4: Update hierarchy

Abbreviations

2D	two dimensional
3D	three dimensional
ADF	adaptively sampled distance field
API	Application Programming Interface
CAD	Computer Aided Design
CFL	Courant-Friedrichs-Lewy
CPU	Central Processing Unit
DbC	Design by Contract
GLSL	OpenGL Shading Language
GPU	Graphics Processing Unit
GUI	Graphical User Interface
LOD	Level of Detail
KNN	k-nearest neighbors
MC	marching cubes
PCF	percentage closer filtering
RAM	Random Access Memory
SPH	Smoothed Particle Hydrodynamics
voxel	volumetric pixel
XML	Extended Markup Language

Bibliography

[1] Blender 3D. http://www.blender.org.

[2] Boost C++ Libraries. http://www.boost.org.

[3] Design by Contract. http://www.eiffel.com.

[4] GLUT - The OpenGL Utility Toolkit. http://www.opengl.org/resources/libraries/glut/.

[5] Intel Threading Building Blocks. http://www.threadingbuildingblocks.org/.

[6] Kronos Group, OpenGL ARB. http://www.opengl.org.

[7] Qt Software. http://www.qtsoftware.com.

[8] The OpenMP API specification for parallel programming. http://openmp.org/.

[9] Wavefront Technologies OBJ 3D Format. http://en.wikipedia.org/wiki/Obj.

[10] ADAMS B., LENAERTS T., D. P. Particle splatting: Interactive rendering of particle-based simulation data. Tech. rep., Katholieke Universiteit Leuven, 2006. Technical Report CW 453.

[11] AKLEMAN, E. Designing symmetric high-genus sculptures. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Sketches* (New York, NY, USA, 2006), ACM, p. 73.

[12] BAE, S. H., BALAKRISHNAN, R., AND SINGH, K. Ilovesketch: as-natural-as-possible sketching system for creating 3d curve models. In *UIST '08: Proceedings of the 21st annual ACM symposium on User interface software and technology* (New York, NY, USA, 2008), ACM, pp. 151–160.

[13] BAI, L., EYIYUREKLI, M., AND BREEN, D. E. Automated shape composition based on cell biology and distributed genetic programming. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation* (New York, NY, USA, 2008), ACM, pp. 1179–1186.

[14] BENTLEY, J. L. Multidimensional binary search trees used for associative searching. *Commun. ACM 18*, 9 (1975), 509–517.

[15] BLOYD, C. Marching Cubes Implementation. http://astronomy.swin.edu.au/pbourke/modelling/polygonise/.

[16] CORDS, H., AND STAADT, O. Instant liquids, July 2008.

[17] COURANT, R., FRIEDRICHS, K., AND LEWY, H. *ÃIJber die partiellen Differenzengleichungen der mathematischen Physik*, vol. 100. Mathematische Annalen, 1928.

[18] CURLESS, B. From range scans to 3d models. *SIGGRAPH Comput. Graph. 33*, 4 (2000), 38–41.

[19] DESBRUN, M., AND PAULE GASCUEL, M. Smoothed particles: A new paradigm for animating highly deformable bodies. In *In Computer Animation and Simulation '96 (Proceedings of EG Workshop on Animation and Simulation* (1996), Springer-Verlag, pp. 61–76.

[20] DÖNNI, U. Governed design: A new paradigm to shape finding.

[21] FÜSSLER, U. Design by tool design. In *Füssler Berlin* (September, 2008).

[22] GALYEAN, T. A., AND HUGHES, J. F. Sculpting: an interactive volumetric modeling technique. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1991), ACM, pp. 267–274.

[23] GIBSON, S. F. F. Beyond volume rendering: Visualization, haptic exploration, and physical modeling of voxel-based objects. In *In Proc. Eurographics workshop on Visualization in Scientific Computing* (1995), Springer-Verlag, pp. 10–24.

[24] HART, G. Sculptural forms from hyperbolic tessellations. *Shape Modeling and Applications, 2008. SMI 2008. IEEE International Conference on* (June 2008), 155–161.

[25] HERNQUIST, L., AND KATZ, N. Treesph: A unification of sph with hierarchical tree method. *Astrophysical Journal Supplement Series 70* (1989), 419–446.

[26] HUT, P., AND MAKINO, J. The art of computational science: Moving stars around. http://www.artcompsci.org/vol_1/v1_web/node34.html, January 2004.

[27] IGARASHI, T., MATSUOKA, S., AND TANAKA, H. Teddy: a sketching interface for 3d freeform design. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1999), ACM Press/Addison-Wesley Publishing Co., pp. 409–416.

[28] KARA, L. B., D'ERAMO, C. M., AND SHIMADA, K. Pen-based styling design of 3d geometry using concept sketches and template models. 149–160.

[29] KILIAN, A., AND OCHSENDORF, J. Particle-spring systems for structural form finding. Journal for the international Association for shell and spatial structures: IASS, 2005.

[30] LENNARD-JONES, J. E. C. Lennard-jones, j. e. cohesion. In *Proceedings of the Physical Society* (1931), no. 43, pp. 461–482.

[31] LIPSON, H., AND SHPITALNI, M. Conceptual design and analysis by sketching. In *In AIDAM-97* (1997), pp. 391–401.

[32] LORENSEN, W. E., AND CLINE, H. E. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph. 21*, 4 (1987), 163–169.

[33] MOUSTAKAS, K., NIKOLAKIS, G., TZOVARAS, D., CARBINI, S., BERNIER, O., AND VIALLET, J. E. 3d content-based search using sketches. *Personal Ubiquitous Comput. 13*, 1 (2009), 59–67.

[34] MÜLLER, M., SCHIRM, S., AND DUTHALER, S. Screen space meshes. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, Switzerland, 2007), Eurographics Association, pp. 9–15.

[35] PERLIN, K. An image synthesizer. *SIGGRAPH Comput. Graph. 19*, 3 (1985), 287–296.

[36] PERRY, R. N., AND FRISKEN, S. F. Kizamu: a system for sculpting digital characters. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2001), ACM, pp. 47–56.

[37] S. CRISTIANO, M. FIORENTINO, G. M. A. E. U. Real-time particle based virtual sculpturing.

[38] SUTHERLAND, I. E., AND EDWARD, I. Sketchpad: A man-machine graphical communication system. *AFIPS Spring Joint Computer Conference* (May 1963).

[39] TIMO BREMER, P., PORUMBESCU, S. D., KUESTER, F., HAMANN, B., JOY, K. I., AND LIU MA, K. Virtual clay modeling using adaptive distance fields. In *in Proceedings of the 2002 International Conference on Imaging Science, Systems, and Technology (CISST* (2002).