Master Thesis

Boolean operations on point set surfaces

Author(s):
Rota, Pascal

Publication Date:
2008

Permanent Link:
https://doi.org/10.3929/ethz-a-005565543

Rights / License:
In Copyright - Non-Commercial Use Permitted
Boolean Operations on Point Set Surfaces

Pascal Rota

Master Thesis
2007/10/01 - 2008/03/31

Dr. Gaël Guennebaud
Prof. Dr. Markus Gross
Abstract

The goal of this project is to perform accurate boolean operation on Point Set Surface (PSS). Since PSS can be defined by an implicit scalar field, Constructive Solid Geometry (CSG) can easily be enabled. However, in order to optimize both the memory consumption and the performance, and to allow further editing on the surface, it is often desirable to create a new point cloud approximating the result of the boolean operation. This is a much more challenging task involving sampling optimization and surface re-fitting.

In this thesis I propose a possible approach to the above problem. The algorithm presented in this thesis can be subdivided in two steps:

1. Step one consists on creating a coarse approximation of the boolean operation itself. In this step I evaluate the crease line and eliminate all non-visible points on each involved PSS. This is trivial for dense point clouds, however, in case of a low sampling density it is not trivial. Therefore I present a robust classification algorithm to approximate the boolean operation.

2. The second step consists of optimizing and re-fitting the borders and surfaces of both PSS objects such that the final surface fits the involved models as good as possible.

I will use the Point Sampled Cell Complex (PSCC) approach in a similar way as described in [AA06] and combine it with the Algebraic point set surfaces (APSS) [GG07] method which results in significant improved stability where planar MLS fails. Beside the PSCC approach gives me a solid framework to handle boolean operation on PSS and to build a complete CSG tree.
Zusammenfassung

Das Ziel dieser Arbeit ist die Ausführung einer genauen booleschen Operation auf einer Punktwolke genannt Point Set Surface (PSS). Da PSS als eine implizite Oberfläche dargestellt werden kann, kann man CSG (Constructive Solid Geometry) auf eine einfache Art und Weise definieren. Nichtsdestotrotz, um in Bezug auf Speicherverbrauch und Leistung zu optimieren und eine Weiterverarbeitung der Oberfläche zu gewährleisten, ist es oft vorteilhaft eine neue Punktwolke zu erzeugen, die das Ergebnis einer booleschen Operation approximiert. Dies ist ein viel schwierigeres Problem, das eine Optimierung der einzelnen Punkte und eine Oberflächenanpassung mit sich bringt.

In dieser Arbeit schlage ich eine Methode vor, die das obige Problem löst. Der Algorithmus kann in zwei Schritten beschrieben werden:

1. Als erstes wird eine grobe Approximation der booleschen Operation errechnet. Dabei wird die Schnittlinie errechnet und die nicht benötigten Punkten der zwei beteiligten PSS entfernt. Dies ist einfach, wenn die Punktwolke eine hohe Dichte aufweist, aber bei niedriger Dichte ist es komplizierter. Deshalb presentiere ich eine robuste Methode, die dies erreicht.

2. Der zweite Schritt besteht darin die nun ungenau gewordenen Ränder und Oberfläche zu optimieren, sodass die Ursprüngliche Form der beteiligten Objekte so gut wie möglich angenähert werden.

Ich werde die Point Sampled Cell Complex (PSCC) Annäherung in einer ähnlicher Art und Weise wie in [AA06] benutzen und diese mit Algebraic point set surfaces (APSS) [GG07] kombinieren, was eine deutlich bessere Stabilität, wo planare MLS scheitern würde. Ausserdem gibt mir die PSCC-Annäherung ein solides Framework um die boolesche Operation auszuführen und ein komplettes CSG-Baum daraus zu bauen.
Introduction

Point Set Surfaces (PSS) define a smooth surface from a set of unstructured points using local Moving Least Squares (MLS) approximations. Cutting edge PSS now support low sampling density, sharp features and real-time visualization.

The goal of this project is to perform accurate boolean operation on PSS (Point Set Surface). Since PSS can be defined by an implicit scalar field, CSG (Constructive Solid Geometry) can easily be enabled. However, in order to optimize both the memory consumption and the performance, and to allow further editing on the surface, it is often desirable to create a new point cloud approximating the result of the boolean operation. This is a much more challenging task involving sampling optimization and surface re-fitting.

Task/Initial Roadmap

1. Implementation of basic boolean operations on PSS (CSG tree)
2. Creating of unique point clouds approximating the result of the boolean operation using the Point Sampled Cell Complex (PSCC) approach:
   a) Reconstruct the crease line using the PSCC: each sample of the simplexes of dimension 1 (crease line) and 0 (corners) are interpolating and have one normal per connected 2D simplex. The normals comes from initial surfaces.
   b) Extend the approach to be able to perform boolean operation between two PSCCs
   c) Implement smooth transitions.
3. Improve the accuracy of the approximation using a sampling optimization/re-fitting algorithm.
4. If time permits: extend the approach such that all samples have the same behavior. In other words remove the Point Sampled Cell Complex concept (or at least remove some of its constraints)

Student: Pascal Rota
Supervisor: Gaël Guennebaud
Start: 2007/10/01 - End 2008/03/31
Acknowledgments

First of all I would like to thanks Gaël Guennebaud, for supervising this thesis, his patience and his many advises. Without his help, this thesis would not have been possible.

Last but not least, my friends and family, who supported me during the whole time and never stopped to believe in me.
Contents

List of Figures xi

1 Introduction 1
  1.1 Motivation .......................................................... 1
  1.2 Related Work ....................................................... 2
  1.3 Outline ............................................................. 3

2 Background 5
  2.1 Local Weighting ..................................................... 6
  2.2 APSS Definition .................................................... 6
  2.3 Sphere Fitting ...................................................... 9
  2.4 Point-Sampled Cell Complexes .................................. 12
    2.4.1 The Cell Complex ........................................... 12
    2.4.2 Tangent Frame Point Set Surface ......................... 13
    2.4.3 Tangent Frame Based Cell Complex ....................... 14
  2.5 Conclusion ......................................................... 15

3 APSS Based Cell Complexes 17
  3.1 Crease Line interpolation - Hermite Method ................. 17
  3.2 Crease Line interpolation - APSS Method .................... 18
  3.3 APSS on Cell Complexes ....................................... 20
  3.4 APSS on Cell Complexes - Results ........................... 20

4 Accurate Boolean Operations on PSS 23
  4.1 Background ........................................................ 23
  4.2 Boolean operations on PSS ...................................... 25
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Weighting scheme</td>
<td>7</td>
</tr>
<tr>
<td>2.2</td>
<td>Scalar field illustration</td>
<td>8</td>
</tr>
<tr>
<td>2.3</td>
<td>Instability of geometric fit</td>
<td>9</td>
</tr>
<tr>
<td>2.4</td>
<td>Algebraic vs geometric distance</td>
<td>10</td>
</tr>
<tr>
<td>2.5</td>
<td>Generic weighting scheme</td>
<td>13</td>
</tr>
<tr>
<td>2.6</td>
<td>Projection on cell complex</td>
<td>14</td>
</tr>
<tr>
<td>3.1</td>
<td>Hermite curve approximation</td>
<td>18</td>
</tr>
<tr>
<td>3.2</td>
<td>APSS curve approximation method</td>
<td>20</td>
</tr>
<tr>
<td>3.3</td>
<td>APSS cell complex example</td>
<td>22</td>
</tr>
<tr>
<td>4.1</td>
<td>Boolean operations</td>
<td>24</td>
</tr>
<tr>
<td>4.2</td>
<td>CSG tree</td>
<td>25</td>
</tr>
<tr>
<td>4.3</td>
<td>Implicit field - error case</td>
<td>26</td>
</tr>
<tr>
<td>4.4</td>
<td>Basic optimization</td>
<td>29</td>
</tr>
<tr>
<td>4.5</td>
<td>Basic optimization</td>
<td>30</td>
</tr>
<tr>
<td>4.6</td>
<td>CSG Examples</td>
<td>30</td>
</tr>
<tr>
<td>5.1</td>
<td>Vertex clustering method</td>
<td>32</td>
</tr>
<tr>
<td>5.2</td>
<td>Decimation results - clustering</td>
<td>34</td>
</tr>
<tr>
<td>5.3</td>
<td>Decimation results - error quadrics</td>
<td>37</td>
</tr>
<tr>
<td>A.1</td>
<td>Main UI</td>
<td>42</td>
</tr>
<tr>
<td>B.1</td>
<td>CSG Example I</td>
<td>46</td>
</tr>
<tr>
<td>B.2</td>
<td>CSG Example II</td>
<td>47</td>
</tr>
<tr>
<td>B.3</td>
<td>CSG Example III</td>
<td>48</td>
</tr>
</tbody>
</table>
List of Figures

B.4 CSG Example IV ......................................................... 49
C.1 Surface splatting pipeline ........................................... 52
C.2 Main UI ...................................................................... 53
D.1 2D APSS implementation ............................................ 56
Introduction

1.1 Motivation

Nowadays most modeling tools are based on polygonal meshes which can be simple triangles or higher order surfaces, such as subdivision surfaces [Cat74]. Those representations have been well studied, and many optimized algorithms are available for various applications. During the last few years we observed an increasing demand on efficient tools allowing to manage more and more complex models. However, owing to their lack of flexibility, mesh-based representations become rapidly inefficient to handle such models, hence making such approaches impracticable.

Among the wide variety of alternative representations that have been proposed during the last decade, point-based geometries are by far the most successful one [Gro06, GP07]. Here, the surface of an object is represented as a set of points where each point contains several surface attributes, such as a normal, a radius or colors information. A central feature of such point-based representations is the lack of any explicit connectivity between the samples. The connectivity information is retrieved by so called neighbor queries which are usually performed using space-partitioning data structures like grids or kd-trees.

Compared to polygonal meshes, meshless representations exhibit several advantages. First of all, since no connectivity information have to be managed, the manipulation of huge datasets and dynamic models become significantly simpler. In particular, topology changes, which often occur during boolean operations, are automatically handled. Last but not least, point-based techniques allows to directly handle data coming from real-world acquisition devices, such as 3D scanners or Time Of Flight cameras. Those devices become more and more popular and produce huge datasets.

In the context of geometric modeling, a major challenge with mesh-less representations is to
1 Introduction

define a smooth and continuous surface over the input point cloud. To do so, the most appealing approach are so called Point Set Surfaces (PSS) \[^{[ABCO^{+}01a]}\] which are based on Moving Least Squares (MLS) surface approximation \[^{[Lev01]}\]. Such representations normally do not interpolate the input points but instead approximate them in a controllable manner. One the numerous advantages of PSS is the hybrid explicit/implicit representation that makes it particularly attractive for modeling purpose.

While the current research mainly focuses on freeform modeling, boolean operations remains a fundamental widely used modeling tools. Indeed, boolean operations are an easy and intuitive way to create new objects out of the original ones. Though a few algorithms to perform boolean operations on point clouds have already been proposed \[^{[PKKG03, AD03]}\], these algorithms assumes a densely sampled point cloud and do not take into account a continuous representation, making them non robust and inaccurate.

The main goal of this thesis is to design a robust algorithm to perform accurate boolean operations on approximating PSS. To achieve this goal, the first step is the choice of the surface representation. In this thesis, I introduce a new representation combining the robustness to low sampling density and efficiency of the Algebraic Point Set Surfaces (APSS) definition \[^{[GG07]}\] with the Point Sampled Cell Complexes \[^{[AA06]}\] concept which offers a robust framework to handle sharp features and non manifold situations that may occurs during CSG operations. The second step is the computation of the boolean operation itself. Since a PSS can be defined as an implicit scalar field, most of the techniques developed for implicit modeling could be applied. However, a PSS can only be evaluated in the close neighborhood of the input points. In this thesis I present a simple and robust way to evaluate the PSS everywhere.

A further step is to apply the boolean operation, i.e., to compute a new point cloud which yields a PSS approximating as good as possible the result of the exact boolean operation. This is a non trivial step which requires the detection and generation of crease lines as well as surface optimization. Indeed, central to my boolean operator algorithm is a surface re-refitting algorithm which optimizes a given point cloud such that its corresponding PSS matches a target surface. While such an algorithm is fundamental to perform accurate boolean operations, it turns out to be useful in many other applications. Therefore, in this thesis, I also investigated the use of my surface optimization routines to the problem of surface simplification.

1.2 Related Work

This work is highly based on two recent techniques concerning surface modeling using point primitives:

The first technique is about the continuous representation of a point set surface (PSS). The representation is based on a moving least-squares (MLS) fitting of algebraic spheres and was introduced by Gaël Guennebaud and Markus Gross in \[^{[GG07]}\]. It provides a stable and continuous approximation of the surface under low sampling rate and high curvature while still being very efficient to compute. However, an APSS can only represent a smooth surface while a boolean operation usually generate a lot of sharp features. While the APSS framework comes with an extension to represent such sharp features by mean of tagged point cloud, this approach
turns out to be too limited for our purpose since the shape of the creases lines are difficult to control, it is limited to corners with three edges and non-manifold situations cannot be handled.

To overcome these limitations, we opted for the cell complex concept. Here the idea is that an object can be represented by multiple surfaces connected to each other in a way that the surface is piecewise smooth. Those surfaces are called cells and each cell has a border to delimit itself from its neighbors. This structure is called a cell complex and was introduced by Anders Adamson and Marc Alexa in [AA06]. The main disadvantage of this method is the surface representation. They used a tangent frame based approach to represent the PSS, which is a planar MLS fit, and has a poor stability behavior in presence of high curvature and/or low sampling rate.

Both techniques complete each other mutually. The APSS framework provides a robust approximation of a point cloud while the cell complex representation provides a way to combine multiple surfaces into one object without the need to create a new point cloud representing the surface. These two advantages are perfect to represent the result of a boolean operation on two PSS.

1.3 Outline

Chapter 2 introduces the two techniques on which this thesis is based upon. One of them is the “Algebraic Point Set Surfaces” (APSS) framework [GG07] and the other one is the “Point-Sampled Cell Complexes” (PSCC) framework.

Chapter 3 presents the novel surface representation combining the cell complex method with APSS. This will give us the foundation for boolean operations on point set surfaces (PSS). In particular two novel methods to reconstruction a curve in 3D space, on of them based on APSS, are presented.

Chapter 4 presents our novel accurate boolean operation algorithm on PSS. This includes the technique used to compute the boolean operation itself, as well as the optimization/refitting algorithms used to optimize the resulting PSS.

Chapter 5 presents a common application on PSS, namely point cloud decimation. It uses the optimization algorithm introduced in chapter 4.

Chapter 6 gives general discussions on our approach, and concludes with an outlook for future work.

Appendix 1 introduces the software platform bopss_viewer, which serves as an implementation for the algorithm and methods presented in this thesis.

Appendix 2 shows some results of CSG operations on point set surfaces.

Appendix 3 makes a short overview of the implementation of surface splatting using a screen space formulation of the Elliptical Weighted Average (EWA) filter.
1 Introduction
Background

The goal of this chapter is to present to the reader the main mathematical foundations and techniques on which this thesis is built. These include the Algebraic Point Set Surfaces (APSS), which are an extension of Point Set Surfaces (PSS) based on moving least squares (MLS) fitting of algebraic spheres and the Point-Sampled Cell Complexes (PSCC) method, which is introduced in section 2.4. Both approaches are based on MLS surfaces [Lev98], which is a method to reconstruct a continuous surface from a point cloud. The surface is locally approximated by fitting a basis surface in weighted least squares sense.

While several MLS variants have been proposed [ABCO+01b, AA03, SOS04, AA07, AK04], in this thesis we will focus on the recent APSS definition which has the advantages to present a higher stability to handle low sampling rate, while achieving tight approximations and high performance.

In contrast, PSCC allows to explicitly and robustly represent sharp features and non manifoldness. More generally, PSCC gives us the possibility to combine several PSS into one objects without the need to merge all PSS into one big point cloud and to form out of them one continuous surface. For this reason the combination of both approaches (which is presented in the next chapter) is perfect for processing and managing of boolean operations.

Before describing the APSS definition and its associated sphere fitting operation, I will first introduce the concept of local weighting. Finally, the PSCC technique will be presented.
2 Background

2.1 Local Weighting

The principle of moving least squares surface fitting is to reconstruct a continuous surface using local weighted least squares to fit a primitive. Therefore we need a weighting scheme to define the local weight of samples. We define the weight of a point $p_i$ in the neighborhood of a given query point $x$ with the following generic weight function:

$$w_i(x) = \phi \left( \frac{\|p_i - x\|}{h_i(x)} \right)$$

(2.1)

where $\phi$ is a smooth, monotonically decreasing weight function. There are two possible approaches to define the weight function. One of them is an interpolatory approach which interpolates a point cloud exactly. This is done with a function which has a singularity at zero and therefore an infinite weight value at this location. The advantage is clear that the point cloud can be interpolated exactly. But on the other hand it leads to oscillations and visually disturbing artifacts if the point cloud contains noise. The other approach is an approximation of the point cloud. In this case every point has a weight between 0 and 1 and they are not interpolated. This results in a much smoother surface.

The challenge is now to find a good middle way between interpolation and approximation. The weight function should result in a smooth surface, even on a noisy data set, but it should also retain surface details. This is done with a function that resembles to a Wendland radial basis function $(1 - r/h)^4 \ast (4r/h + 1)$ [WHG02]. This class of function only depends on the distance from the origin, i.e. $\phi(x) = \phi(\|x\|)$. Furthermore it should also resemble to a Gaussian function, which is the ideal low-pass filter and filters noise out of the point cloud. Such a function is presented here:

$$\phi(x) = \begin{cases} 
(1 - x^2)^y & \text{if } x < 1 \\
0 & \text{otherwise}
\end{cases}$$

(2.2)

As it is depicted in figure 2.1 an exponent of $y = 4$ yields a nice shape and it is close to the Wendland function. Furthermore it is faster to compute, because the square root evaluation can be simply bypassed.

$h_i$ is a parameter describing the support size of the weight function. There are several methods on how to set this parameter. It can be constant, depending on $x$, on $p_i$ or even be a function over its neighborhood.

2.2 APSS Definition

The APSS approach which was introduced by Gaël Guennebaud and Markus Gross at SIGGRAPH 2007 [GG07] is a robust and stable extension of PSS and has significant advantages in terms of stability of the projection under low sampling rate and in the presence of high curvature.
2.2 APSS Definition

An algebraic sphere is defined as the 0-isosurface of the scalar field \( s_u(x) = [1, x^T, x^T x] u \) where \( u = [u_0, \ldots, u_{d+1}] \in \mathbb{R}^d \) and \( d \) the dimensionality. Further details are discussed in section 2.3.

As already stated, an algebraic sphere is defined as the 0-isosurface of the scalar field \( s_u(x) = [1, x^T, x^T x] u(x) \) where \( u = [u_0, \ldots, u_{d+1}] \in \mathbb{R}^d \). \( u \) represents the scalar coefficients and \( d \) the dimensionality of the sphere. If \( u_{d+1} \neq 0 \) the sphere is defined with its center \( c \) and radius \( r \) and can be computed as follow:

\[
c = -\frac{1}{2u_{d+1}} [u_1, \ldots, u_d]^T \tag{2.3}
\]

\[
r = \sqrt{c^T c - \frac{u_0}{u_{d+1}}} \tag{2.4}
\]

In case of \( u_{d+1} = 0 \) it is a degenerated case and \( u \) represents the plane equation with \( u_0 \) as the distance from the origin and \([u_1, \ldots, u_d]\) as the plane’s normal.

Implicit Definition

Like most PSS variant, APSS can be defined as the zero set of a scalar field \( f(x) \). In the case of APSS, the scalar field is simply defined as:

\[
f(x) = s_{u(x)}(x) = [1, x^T, x^T x] u(x) \tag{2.5}
\]

where \( u(x) \) intuitively corresponds to the coefficients of the sphere which approximate the best the neighborhood of \( x \). This neighborhood is defined by the aforementioned weighting scheme,
2 Background

Figure 2.2: Illustration of the scalar field with a standard temperature based color encoding scheme. The black line represents the 0-isosurface.

and the details on this fitting operation are given in section 2.3. An illustration of an implicit surface on a cutting plane of the Standford Bunny can be seen in figure 2.2.

Gradient

The gradient can be used to determine the direction of greatest rate of increase of the scalar field and the norm denotes the rate of change. It can also be used as an approximation of the normal if the evaluation point is on the surface. The gradient of a surface defined by an implicit function $f(x)$, which represents a scalar field, is defined as its derivative $\nabla f(x)$ and is computed as follow:

$$
\nabla f(x) = \begin{bmatrix} 1, x^T, x^T x \end{bmatrix} \nabla u(x) + \begin{bmatrix} 1 & e_0^T & 2e_0^T x \\ \vdots & \vdots & \vdots \\ 1 & e_{d-1}^T & 2e_{d-1}^T x \end{bmatrix} u(x)
$$

(2.6)

Projection Operator

The above implicit definition is particularly convenient for inside/outside classification, normal and curvature computations. But computing a point on the surface requires the definition of a projection operator. The paper proposes following procedure as a practical receipt to implement a projection operator in APSS. This procedure follows the almost orthogonal projection described in [AA04] and can be implemented as follow:

We begin with the basic idea of the projection operator. Given an evaluation point $x$ relatively close to the point cloud, we then project this point iteratively on the best locally fitted sphere of the previous projection result. This is done until it converges. More precisely we define a series of points $q_i$ such that $q_{i+1}$ is an orthogonal projection of $x$ on an algebraic sphere defined
2.3 Sphere Fitting

The main challenge of APSS is how to fit a sphere on a point cloud. There are several possible approaches to do this. The most obvious solution would be to perform a so called “geometric fit” that minimizes the sum of squared Euclidian distances to a set of point samples. This problem is clearly non-linear and therefore a solution can be found only through an iterative method. However these iterative methods are computationally expensive and numerically unstable. For instance if the algorithm uses center and radius or a parametric form to represent the current sphere, i.e. center and radius, then the radii tends to a infinite value in case of planar or near planar areas. This instability is shown in figure 2.3.

An alternative method would be to perform a so called “algebraic fit” that minimizes the sum of squared algebraic distance to a set of points. The algebraic distance is not linear but it leads to a linear minimization. An example of this solution, which uses algebraic sphere fitting, is introduced in the next section.

Given a point set $P$ with $n$ points and let $W$ be a $n \times n$ diagonal weight matrix defined as:

$$W(x) = \begin{bmatrix}
w_0(x) \\
\cdot \\
w_{n-1}(x)
\end{bmatrix} \quad (2.7)$$

by $u(q_i)$. The starting point is $q_0 = x$. The series end with $q_\infty$. A practical implementation will carry on the recursion until the displacement falls below a given threshold.

Figure 2.3: An illustration of the instability of geometric fit: The curve cannot be reconstructed near the inflexion point because the radii of the locally fitted spheres tend to infinite value.
2 Background

Figure 2.4: The Euclidean distance (green cone) compared to the algebraic distance (orange paraboloid) to the 2D yellow circle. Note that the two distances are tangent at the circle.

and let $D$ be a $n \times (d + 2)$ design matrix defined as:

$$D = \begin{bmatrix}
1 & p_i^T & p_i^T p_i \\
\vdots & \vdots & \vdots \\
1 & p_{n-1}^T & p_{n-1}^T p_{n-1}
\end{bmatrix} \quad (2.8)$$

Then the solution $u(x)$ of the algebraic sphere fit at a given point $x$ is given by:

$$u(x) = \arg \min_{u, u \neq 0} \left\| W^{\frac{1}{2}}(x) Du \right\|^2 \quad (2.9)$$

In order to avoid the trivial solution $u(x) = 0$, $u$ has to be constrained. Due to the high influence of constraints on the solution of the minimization problem the constraint should have some desired characteristics, like reliably handle planar cases and to behave as closely as possible to a geometric fit. Both characteristics are nontrivial and influence the stability of the fitting.

Ideally normals should be used for local sphere approximation to preserve high frequency details. This can be achieved by adding derivative constraints $\nabla s_u(p_i) = n_i$ with $\|n\| = 1$ in the above minimization problem, which matches the input normals with the gradient of the fitted spheres. Thus it is possible to fulfill the above characteristics. Furthermore this constraint forces the algebraic distance to behave very similar to the geometric distance for points which are near the surface. A 2D example of the similarity of the algebraic and geometric distance is depicted in Figure 2.4.

This leads to the following standard linear system of equations that can be solved efficiently:

$$W^{\frac{1}{2}}(x) Du = W^{\frac{1}{2}}(x) b \quad (2.10)$$

with $W$ a $(d + 1)n \times (d + 1)n$ weight matrix, $D$ a $3n \times (d + 2)$ design matrix and $b$ a $3n \times 1$ vector. In case of 3D we have now 4 equations per point, one for the position and 3 for the gradient. The matrices and vectors are defined as:
2.3 Sphere Fitting

\[ W(x) = \begin{bmatrix} \vdots & \vdots & \vdots \\ \beta w_i(x) & w_i(x) & \vdots \\ \vdots & w_i(x) & \vdots \end{bmatrix} \]  \hspace{1cm} (2.11)

\[ D(x) = \begin{bmatrix} \vdots & \vdots & \vdots \\ 1 & p_i^T & p_i^T p_i \\ 0 & e_0^T & p_0^T p_i \\ \vdots & \vdots & \vdots \\ 0 & e_{d-1}^T & p_{d-1}^T p_i \\ \vdots & \vdots & \vdots \end{bmatrix} \]  \hspace{1cm} (2.12)

\[ b = \begin{bmatrix} \vdots \\ 0 \\ e_0^T n_i \\ \vdots \\ e_{d-1}^T n_i \\ \vdots \end{bmatrix} \]  \hspace{1cm} (2.13)

The system can be now split into two parts to minimize the two key constraints (positional and derivative) separately. First we start with the derivative constraints by giving an infinite importance to the derivative constraint by setting \( \beta = 0 \). The resulting sub system of equation allow us to solve the \( d + 1 \) last coefficient of \( u \), which completely defines the gradient of the sphere. This yields the normal equation defined as:

\[
\begin{bmatrix} \sum w_i(x) I_3 & 2 \sum w_i(x) p_i \\ (2 \sum w_i(x) p_i)^T & 4 \sum w_i(x) p_i^T p_i \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} \sum w_i(x) n_i \\ 2 \sum w_i(x) p_i^T n_i \end{bmatrix}
\]  \hspace{1cm} (2.14)

Then the system of equation 2.9 are used to compute \( u_0 \) whereas \( u_1 \) to \( u_{(d+1)} \) are already fixed by the first step. This strategy leads to the following explicit solution for \( u \) (with \( d = 3 \)):
2 Background

\[
\begin{bmatrix}
  u_1 \\
  u_2 \\
  u_3 \\
  u_0
\end{bmatrix}
= \sum \tilde{w}_i \mathbf{n}_i - 2u_4 \sum \tilde{w}_i \mathbf{p}_i - u_4 \sum \tilde{w}_i \mathbf{p}_i^T \mathbf{p}_i - \sum \tilde{w}_i \mathbf{p}_i,
\]

where \( w_i = w_i(x) \) and \( \tilde{w}_i \) is the normalized weight of the sample \( \mathbf{p}_i \): \( \tilde{w}_i = w_i / \sum_j w_j \).

Note that setting \( u_4 \) in the above equations allows to derive a special case of APSS where the surface is locally approximated by a plane passing through the weighted average of the sample positions and having for normal the weighted average of the sample normals. In this thesis we will refer to this variant as Simple Point Set Surfaces (SPSS). The quality of the surface is clearly inferior to the one of APSS due to the planar fit.

For information about implementations of the APSS framework as of a 2D implementation of myself, see appendix D.

2.4 Point-Sampled Cell Complexes

This section introduces Point-Sampled Cell Complexes (PSCC), an approach for high quality piecewise smooth surfaces based on point samples and cell complexes. PSCC was introduced by Anders Adamson and Marc Alexa at SIGGRAPH 2006 [AA06]. An advantage of this idea is that distinct parts of an object are separated into cells. Each cell is represented by a point cloud. This allows the creation of an adaptively sampled cell complex, separate semantic parts from each other and much more. It also simplifies operations when several objects are involved and adds features like sharp edges for free.

In the following sections I will give a short introduction to the Point-Sampled Cell Complexes and an implementation using the local tangent frames method.

2.4.1 The Cell Complex

The general principle of cell complexes is to decompose an object into several surface patches. Each patch is then connected to one or more curve segments which form its boundary; and each curve segment has endpoints as boundary. It is also possible to have closed curves and therefore no boundaries for curves. This allows modeling of some common shapes much easier. For clarity I will only explain up to three dimensional spaces but the whole concept is also valid for higher dimensional space.

The cell complex is represented through cells, each of them denoted as \( M_i^d \), where \( d = \{0, 1, 2\} \) is the dimension and \( i \) an index. Each cell has also a point set representing its geometry. De-
2.4 Point-Sampled Cell Complexes

Figure 2.5: (a) A cell complex representation using a point set for its geometry. (b) An advanced cell complex using tangent plane projection.

Depending on the surface approximation, each point might have additional properties such as a local normal and/or a tangent frame. Figure 2.5 depicts two objects represented by a cell complex.

A smooth surface $S_P$ over a cell complex is defined as the set of stationary points of a projection operator $P(x)$ with $x \in \mathbb{R}^d$, i.e.:

$$S_P = \{ x \in \mathbb{R}^d | P(x) = x \} \quad (2.16)$$

2.4.2 Tangent Frame Point Set Surface

In the remaining sections I will introduce a point set surface which uses local tangent frames to project a point $x$ onto the surface. This method is described in [AA06] in all details and only a brief summary is given here.

The tangent frame is approximated by least-squares fitting of a tangent plane on a point cloud, i.e. to fit the plane through the neighbors of a point $x$. The tangent space $T = [t_0, \ldots, t_{d-1}]$ is computed as the eigenvectors of a weighted covariance matrix $C$, where $t_i$ are the eigenvectors with the largest eigenvalues. The eigenvector with a small eigenvalue, e.g. for a 2D surface embedded in 3-space it would be the eigenvector with the smallest eigenvalue, can be used as a hint for the normal direction. $C$ is therefore defined as:

$$C(x) = \sum_i \theta(\|x - p_i\|) (x - p_i) (x - p_i)^T, \quad i \quad (2.17)$$

To project a point $x$ onto a surface representing by point samples, a similar projection procedure as described in section 2.2 is used:

1. First evaluate the local centroid $c(x)$. The local centroid is defined as a weighted average of the neighbor positions:
Figure 2.6: A schematic view of the projection on cell complex. There are 9 cells color-coded as follow: [red] 0D cells are; [green] 1D curve cells; [blue] 2D surface cell.

\[ c(x) = \frac{\sum_i \theta(\|x - p_i\|) p_i}{\sum_i \theta(\|x - p_i\|)} \]  

2. Evaluate the current tangent frame \( T \) from \( C(\ldots) \)
3. Project the local centroid \( c(x) \) on the local tangent frame attached to \( c(x) \).
4. Iterate until convergence

2.4.3 Tangent Frame Based Cell Complex

To project a point \( x \) onto a cell complex, the basic projection operator has to be extended to take the boundaries into account and also to enforce patch continuity. This means that if a point is projected on the surface, it must be checked if the projection result is inside its boundaries. This is done by projection the evaluation point on the curves representing the boundaries and to check if the result is inside. This is done recursively till the point primitive is reached.

Furthermore a cell must define a continuous surface over its neighbors. For a 2D surface, a curve is responsible that two surface coincide. This is done by forcing the surface to go through the curve and to create a continuous surface over both cells.

With these additional requirements the method to project a point onto a cell complex can be redefined as follow:

1. Project \( x \) onto all 0 dimension cells. This will yield the point locations itself, denoted as \( q_0^i(x) \), where \( i \) is an index and 0 denotes the dimension.
2. Project \( x \) onto the 1 dimension curve surrounding a cell, using the projections \( q_1^i(x) \). The projection will yield \( q_1^i(x) \), which are stationary points on a curve between two points defining the curve segment, if the curve is not closed.
3. Finally project \( x \) onto the surface defined by a point cloud and use the projection \( q_2^i(x) \) to delimit the surface with its neighbors.

With this bottom-up procedure it is possible to decide if a projected point on a surface is inside or outside of its boundaries or not. If the projected point is outside of its boundary, it will be simply clipped to the nearest projection \( q_d^i(x) \) one dimension below. Figure 2.6 shows this procedure.
2.5 Conclusion

To take the lower-dimensional boundary into account for the projection operator, it is necessary to redefine the centroid computation to take the boundaries into account. Here we denote \( q_j(x) \) as the result of the projection on the boundary of the current cell. The redefined centroid computation looks as follow:

\[
c(x) = \frac{\sum_i \theta (\| x - p_i \|) p_i + \sum_j \omega (\| x - q_j(x) \|) q_j(x)}{\sum_i \theta (\| x - p_i \|) + \sum_j \omega (\| x - q_j(x) \|)}
\]

(2.19)

where \( \omega(r) \) is a function which has a singularity at zero, e.g. the function goes to infinity when the argument goes toward zero (see section 2.1 for further information). This results to an interpolation of the boundaries and allows a continuous transfer over cell boundaries.

The last problem to solve is how to restrict the projection to the cell’s boundaries. So we have to take the lower-dimension boundary into account, more specifically \( M^d_i \) is bound by \( M^{d-1}_j \). Now it is possible to define an half-space containing the local centroid of \( M^d_i \) which goes through \( q^{d-1}_j \). Now it is a simple step to test if a projection result \( q^d_j \) is in this half-space or not. Let be \( T^{d-1}_{q^{d-1}_j} \) the local tangent frame at point \( q^{d-1}_j \) with point set \( M^{d-1}_j \) and \( T^d_{q^d_j} \) the local tangent frame at point \( q^d_j \) with point set \( M^d_i \). The complement of both tangent frame is tangent to \( M^d_i \) and orthogonal to \( M^{d-1}_j \) and can be evaluated as follow:

\[
b_j = T^d_{q^d_j} - T^{d-1}_{q^{d-1}_j}
\]

(2.20)

This defines the half-space completely. Let be \( q^d_i \) a projection on \( M^d_i \) then we can say that the projection is valid if \( b^T_j (q^d_i - q^{d-1}_j) \geq 0 \) otherwise it has to be reset to \( q^{d-1}_j \). This procedure has to be done for all lower level cell which are linked to \( M^d_i \). Therefore the projection operator defined above is expanded with the half-space check, as defined above. A formal description is as follow:

\[
q^\infty(x) = \begin{cases} 
 q^\infty(x) & \forall_j b^T_j (q^\infty(x) - q_j) < 0 \\
 q_j & b^T_j (q^\infty(x) - q_j) < 0
\end{cases}
\]

(2.21)

2.5 Conclusion

We have presented two different but complementary techniques to define a continuous surface from a single, or multiple, point clouds. Now we can combine the advantages of both approaches into one simple and robust representation of a PSS. This representation is introduced in the next chapter and the used to apply a boolean operation over two PSS, as described in chapter 4.
2 Background
APSS Based Cell Complexes

This chapter introduces a novel representation based on APSS and a cell complex decomposition. Here, the main purpose is to combine the advantages of both approaches. As shown in chapter 2 APSS is much more robust than the tangent frame method based on covariance analysis and performs better approximation while being faster to compute. APSS can also take advantages of first order surface attributes, e.g. gradient, tangent, etc. On the other hand point sampled cell complexes are useful to perform and represent CSG operations, due to its cell based structure.

3.1 Crease Line interpolation - Hermite Method

Handling 2D and 0D cells with the APSS approach is straightforward and already given. The major problem lies over the 1D cells, more precisely on the reconstruction of a 1D embedded in 3D space. Therefore we present here a method to reconstruct such a curve. The method is inspired from a paper of Marc Alexa and Anders Adamson called “Interpolatory Point Set Surface - Convexity and Hermite Data” [AA07, Chapter 5]. The reason why we chose the Hermite scheme is because it is somewhere between SPSS and APSS. It gives us also a relatively good reconstruction of the curve.

The idea is to locally approximate the curve by a line passing through the local centroid.

see my previous comments, just say that the idea is to locally approximate the curve by a line passing trough..... and having for normal....

The idea of this method is not to interpolate the points directly, but first to evaluate a special centroid and project the point $x$ on an average tangent through the centroid and repeating this
3 APSS Based Cell Complexes

Figure 3.1: (a) A set of points defining the curve and a query point \( x \). (b) the projection of \( x \) on the tangents yields \( \hat{p}_i \). (c) Finally project \( x \) on the average tangent going through \( c \).

step till convergent.

Let \( P \) be a point set and \( p_i \) be the position, \( t_i \) be the tangent of a point sample with index \( i \) of point set \( P \). The tangent \( t_i \) of a point is either already available or evaluated with a covariance analysis of the neighborhood. The tangent would be in this case the eigenvector with the biggest eigenvalue. The centroid evaluation is a bit special and cannot be done directly. First we need to evaluate \( \hat{p}_i \) which is the projection of \( x \) onto the tangent \( t_i \) through \( p_i \). This is simply done with

\[
\hat{p}_i = p_i + (t_i^T(x - p_i)) t_i
\] (3.1)

Using approximation weighting scheme (eq. 2.2), we can now define the centroid as follow:

\[
\hat{c} = \frac{\sum_i w_i (\|x - p_i\|) \hat{p}_i}{\sum_i w_i (\|x - p_i\|)}
\] (3.2)

Similar to the SPSS approach, the direction of the approximating line is simply obtained by a weighed average of the sample tangents:

\[
t = \frac{\sum_i w_i (\|x - p_i\|) t_i}{\sum_i w_i (\|x - p_i\|)}
\] (3.3)

All ingredients are now set and the last step is to project \( x \) onto the tangent \( t \) through \( \hat{c} \). The projection is defined as a series with following recursive function:

\[
q_{i+1} = c + t^T(q_i - c) t
\] (3.4)

A schematic view of the whole procedure can be viewed in recursive figure 3.1

3.2 Crease Line interpolation - APSS Method

In this subsection I present an alternative approach to the curve reconstruction problem. This method purely based on the APSS idea and the idea is to intersect two APSS and to extract the curve out of it.
The goal of this method is to approximate a curve with the help of algebraic spheres. The basic idea is to evaluate two surfaces using APSS which uses the same point set except that the normals are separated by an angle not equal zero. Then the intersection of those two surfaces will represent a 1D curve. This curve does not interpolate the point set and should be as stable as standard APSS. Nevertheless this techniques highly depends on the choise of the normals and not only on the tangent directions. This makes this method useless for a general curve approximation, but in case of boolean operation it could probably work quite well.

Let \( P \) be a point cloud defining the crease line, each of them with two normals. These normals are the result of the boolean operation. For a stable approximation of the crease line the angle between both normal should be greater than zero. Otherwise the case is degenerated. The ideal case would be with an angle greater than a few degrees. If only one normal is available at each point it is possible to evaluate the second one. For this we can proceed as follow:

1. First we compute the local tangent frame at every point using the covariance method discussed in 2.4.2. With this method we get the tangent, which is the eigenvector with the highest eigenvalue.

2. Secondly a simple cross product gives us the needed second normal, also called binormal with respect to the Frenet frame.

Like APSS we will use equation 2.9 for the geometric sphere computation. Due to the two normals we also have to solve two linear systems:

\[
\begin{align*}
\mathbf{u}_0(x) &= A_0^{-1}(x)\hat{b}_0(x) \\
\mathbf{u}_1(x) &= A_1^{-1}(x)\hat{b}_1(x)
\end{align*}
\]

where \( A_0 \) and \( A_1 \) the weighted covariance matrix as defined in equation 2.17. With \( \mathbf{u}_0 \) and \( \mathbf{u}_1 \), it is now possible the evaluate both centers \( c_0, c_1 \) and radius \( r_0, r_1 \). The center \( c \) and radius \( r \) of the 1D intersection circle can be evaluated as follow:

\[
\begin{align*}
a &= \frac{r_0^2 - r_1^2 + \|c_1 - c_2\|^2}{2\|c_1 - c_2\|} \\
c &= \frac{c_0 + a(c_1 - c_2)}{(r_0 + r_1)} \\
r &= \sqrt{r_0^2 - a^2}
\end{align*}
\]

The next step is to project the point on the plane through \( c \) and with normal \( \|c_0 - c_1\| \) and then project the result on the nearest point of the circle. This can be done very efficiently by first projecting \( x \) onto the plane and then onto the circle. Figure 3.2 shows this method schematically.
3 APSS Based Cell Complexes

3.3 APSS on Cell Complexes

The last part is to combine multiple APSS surfaces together and create a unified implicit field. The method is very similar to the one used by the tangent frame based cell complexes and can therefore be adapted quite easily.

The centroid can be computed in the same way as described before. Therefore the centroid can be defined as follow:

$$c(x) = \frac{\sum_i \phi(\|x - p_i\|) p_i + \sum_j \omega(\|x - q_j(x)\|) q_j(x)}{\sum_i \phi(\|x - p_i\|) + \sum_j \omega(\|x - q_j(x)\|)}$$  (3.10)

Another problem is how to define the halfspace to partition the cell inside from its outside. This can be also done as described above with a local tangent frame computation and following the same procedure as for the tangent frame based method. This requires the evaluation of the local tangent frame of the affected surfaces as well as on the curves linked to this surface, which nearly doubles the work of a projection operator. Therefore I present here a method which does not require any tangent frame evaluation and only assumes that the projection is nearly orthogonal on curves and surface. This assumption is given for APSS.

Let be $$q_i^{d-1}$$ be the result of the projection on curve $$j$$ which is linked to surface $$i$$ and $$q_i^d$$ be the projection result on surface $$i$$. Now we have to check if $$q_i^d$$ is inside or outside the surface boundaries. First we have to evaluate the centroid $$c(x)$$ which is defined through equation 3.10 and the check on which side $$q_i^d$$ lies. A formal definition is given here:

$$q_i^d = \begin{cases} q_i^d & \forall j \left(c(x) - q_j^{d-1}\right) \left(q_i^d - q_j^{d-1}\right) \geq 0 \\ q_j^{d-1} & \text{otherwise} \end{cases}$$  (3.11)

3.4 APSS on Cell Complexes - Results

An example of a cell complex representing a cube is shown in figure 3.3. The cell complex is represented as follow: Blue represents the surface, green represents the curves and red represents the one dimensional cell, normally called point. The visualization of the curves is done by extracting a surface at an iso value greater than zero. In this case 0.5 was used for the isosurface extraction. The implicit surface representation was done with a standard temperature encoding scheme. Furthermore the surface was rendered using marching cubes. All this was done with
the BOPSS software I developed for this thesis. Information about the software can be found in Appendix B.
Figure 3.3: A cell complex representing a cube and using APSS to represent its surface. (a) The raw cell complex; (b) A visualization of the complete cell complex. The curve is drawn as the 0.5-isosurface and the surface as the 0-isosurface; (c) The surface; (d) The curves; (e) The implicit surface representation; (f) The implicit curve representation.
Accurate Boolean Operations on PSS

In this chapter we present boolean operations on point set surfaces. This is a common technique used for modeling and it is available in most of today’s modeling software, such as Blender, Maya, etc. The presented method is called constructive solid geometry (CSG) [Hub90]. Furthermore some difficulties and challenges that I encountered during implementation of the CSG operation on point set surfaces are also discussed here.

Boolean operations are based on boolean algebra, which was developed by George Boole ¹. With his theories he laid the basis of all modern computer arithmetic and he is seen as one of the founders of the field of computer science.

One advantage of boolean operations is that modeling of objects is intuitive for the user, which is the reason of its popularity, and is therefore implemented in most computer aided design (CAD) softwares. Another significant advantage of CSG lies when using devices for manufacturing, which uses processes like milling or grinding. Those processes can be simulated through a series of CSG operations on an initial surface.

4.1 Background

Constructive Solid Geometry (CSG) is a method to model complex objects by combining simpler objects together, called primitives. Primitives are often very basic ones, e.g. spheres, cubes, cylinders, pyramids, etc., but can also be quite complex objects acquired from a 3D scanning device for instance. With this tool it is easily possible to create very complex $n$-dimensional objects by cleverly combining primitives.

¹George Boole: 1815-1864
In most cases only two objects are combined together with a boolean operation at a time. For this, three basic boolean operations are used which are union (\(\cup\)), subtract (\(-\)) and intersection (\(\cap\)). Let be \(A\) an orange and \(B\) a blue sphere, then the result of these three basic boolean operations are illustrated in the following figure:

![Figure 4.1: An example of boolean operation using two sphere.](image)

As can been easily seen all other standard boolean operations can be substituted by the three basic ones. For example the xor operation is in reality multiple boolean operations cleverly combined together as can be seen: \(A \otimes B = (A - B) \cup (B - A)\). For this the concept of a CSG tree is introduced.

When dealing with implicit surfaces, CSG implicit tree, or Blob Tree[SW05], is a convenient way to combine multiple CSG operations into one big tree. It is built as a binary tree. Each leaf holds a primitive and each inner node represent the result of a boolean operation over its two children. The root node is the results of all boolean operations together.

The CSG tree construct can be implemented in several ways:

A way to implement the tree is to do it in an implicit way. This means that an object is always represented as the result of the tree and no new object is created. A big advantage of this implementation is that the user can change every time the position or orientation of any primitives, and even replace a boolean operation somewhere in the tree without losing its previous work. A drawback is that the computational effort to visualize the final objects is depending on the tree size. In case of objects represented by a scalar field, another drawback is present: After a couple of boolean operations the scalar field becomes very poor and it is very tedious to control the result. In this case it is desirable to represent the tree explicitly, which is the other way to implement it.

The explicit approach means that after each boolean operation a new object is created and both primitives are dropped. This approach leads to a stable scalar field and allows better processing and rendering performance. A sample illustration of a CSG tree can be viewed in figure 4.2.
4.2 Boolean operations on PSS

As we saw in chapter 2, a PSS can be defined as an implicit scalar field. Implicit surfaces are particularly well suited to perform boolean operations, and several techniques have been developed to achieve this, described in [BGC98, LTH86, ?]. However, the application of these techniques to implicit PSS is not as straightforward as it looks like. There are several restriction to take into account in case of PSS.

A general problem of PSS is that the implicit scalar field is not defined everywhere in space, instead it is only defined close to the surface. This region depends on the sampling density, respectively to the current local radius. Furthermore the scalar field is undefined outside this region. Because of this limitation of a general PSS, we have to find a way to define the scalar field everywhere.
Figure 4.3: This figure demonstrate a case where the basic method to extend the implicit field fails.

To define the scalar field everywhere one could follow the approach proposed by Pauly et al [PKKG03]. First we checked if the evaluation point lies inside the domain where the surface is defined. If this is the case we can directly evaluate the sign of the implicit surface. On the other hand if the point lies outside this domain, we use following strategy: First we search the nearest point on the point cloud. Then we check with the normal of the nearest point if it is inside or outside of the surface. The last step consists of computing the value at the location of the evaluation point, which is done by taking the Euclidean distance to the nearest point and sign it with respect to its location, i.e. inside or outside the object.

Nevertheless this method is not perfect and has two drawbacks: First the Euclidean distance is not equal the algebraic distance and therefore the scalar field is not continuous. This is not bad for our case, because we only need the sign. Furthermore the inside/outside test can lead to a wrong result. Indeed in case of a low sampled PSS or sharp features it is possible to classify a point as inside, even thought it is clearly outside and vice versa. This situation is described in figure 4.3.

In order to overcome this last limitation, I propose a more robust algorithm. The idea is to project the evaluation point and then to check with the surface gradient at the projected point. The problem is now to define a robust projection operator which is able to project points which are far the input samples. When the point to project is far away the surface, the idea is to first pick the nearest neighbor sample, and start the iterative projection using this point as the first projection approximation.

As input to our boolean operator we have two PSCC. These PSCC are either native or created on the fly from a point cloud. In the later case the cell complex only consists on one cell, the point cloud itself.

To apply a boolean operation on two PSCC we have to take care of several important things, like cell boundaries and connectivity. For clarity the algorithm is split into several steps:

**Step I: Point clouds classification:** First we have classify the point cloud of each cell if they
4.3 Optimization of the Surface

are inside or outside the other primitive. Here we go cell by cell and uses the global scalar fields of the other primitives to check if a point sample lies inside or outside of it. Depending on the boolean operation used, one of both classes is dropped and the other one defines the new content of the cell. If there is no point sample left the cell is dropped. If the boolean operation is a subtract operation, then the normal is swapped on the second operand.

Step II: Point clouds optimization/re-fitting: Then on every cell a optimization and re-fitting step is done to be as close as possible to the original surface, respectively curve. The optimization/re-fitting algorithm is discussed in the next section.

Step III: Compute new crease curves: The step is to compute the new crease lines. This is done by searching the intersection of every pair of PSS which is not on the same primitive. First we search the samples which are the closest to the other surface. Then we search after the intersection by following the gradient of the other surface and vice versa, until we reached a point near the 0-isosurface of both PSS. Then it is a simple matter of sampling this intersection. Of course the samples obtained by this method only denotes the exact crease line location. Therefore we have to optimize/re-fit those points to be certain that the curve passes through them (see section 4.3).

Step IV: Compute new crease points: This is similar to the crease curves computation except that in this case we intersect a curve with a surface. This can be done by searching a sample on the crease line which is close to the surface and the searching after the intersection of them. Another way to get new crease points is to intersect two curves.

Step V: Relink everything together This step is the hardest one, because we have to relink everything together. First of all each cell retains its links, if they were not dropped while applying the boolean operation. During each crease line and point computation we can directly link the curves with the surfaces and the points with the curve used to compute them.

These five steps is the whole mystery of a boolean operation over two point-sampled cell complexes. To visualize the cell complex, we can simply use a Marching Cubes algorithm on the scalar field.

In the next section, we discuss the optimization/re-fitting algorithm used by the boolean operation.

4.3 Optimization of the Surface

After each boolean operation, we need to optimize/re-fit the surface to approximate as best as possible the geometry of the original PSS. But this is not only the case with PSS, but also the case of boolean operations on free-form solids. In the later case free-forms solid are bounded by a multi-resolution subdivision surfaces and after a boolean operation the subdivision surfaces must be optimized to fit the geometry of the original surfaces [BKZ01].

In this thesis I investigated and implemented an optimization/re-fitting strategy, which optimizes a PSS in such a way that it fits a given surface sampling representing the desired 0-isosurface.

The main problem is that we have to many degree of freedom and too many parameters. Each
point sample has 6 degree of freedom: 3 for the position, 2 for the normal direction, and an additional one for the support radius. Which is not practicable even on small point clouds. Therefore we have to define some additional constraints.

Assuming an even sampling, a practical and intuitive approach is to restrict the control points to move along their normal directions. In this case we have only one degree of freedom per sample. But this is still not practical if the surface consists of several thousands of samples. Therefore we have to find a way that works on a local part of the point cloud, and also leads in the direction of a global solution. A straight forward solution would be to define a local optimization problem as follow:

Given a point cloud $P$ to optimize/re-fit towards a given target surface. Now we take each point iteratively and optimizes the point itself with its neighborhood resulting in a local minima. This local minima is the applied on $P$ and the next point is taken into account.

A better method is to use an energy to minimize, which I implemented. This solution is also hard to solved globally, but can be subdivided into smaller local problem which leads to a minimization, even it is not a global minima.

We will now define the optimization problem in a general way and then, based on an energy function:

More precisely, our optimization problem can be expressed as the minimization of an energy function defined as follow. Let $P$ be the point set to optimize and $S_P = (P)$ the PSS defined by $P$. Now we want to optimize/re-fit $p \in P$ and its $k$-nearest neighborhood. Each sample has one degree of freedom in direction of its normal and the displaced samples are defined as

$$q_i = p_i + t_i n_i$$  \hspace{1cm} (4.1)

where $n_i$ is the normal of sample $p_i$ and $t_i$ is the displacement. The optimized/displaced surface is now defined as $S_Q(t) = (Q)$ where $q_i \in Q$ and $t$ is the vector of the displacement coefficients $t_i$.

As the last ingredient we need a surface $R$ which describes the desired 0-isosurface, e.g. the goal of the optimization process. The minimization problem can now be defined as follow:

$$\min_{t \in \mathbb{R}} e(S_Q(t), R)$$  \hspace{1cm} (4.2)

where $e(.)$ measure the distance/error between the two surfaces. It is difficult to integrate the error between two PSS and so we have to use a discrete integration using a sampling of the target surface. The error approximation get better and better with increasing sampling density. Here we have to find a good middle way between performance and accuracy. A trivial solution would be to use the current sampling density of the point cloud. A more accurate method is to use the current sampling density and then to use a subdivision scheme to compute a higher density and therefore to get a better error approximation.

Therefore $e(.)$ can be defined as follow:
4.3 Optimization of the Surface

Figure 4.4: (a) The starting condition. The blue curve is the current 0-isosurface and the black curve is the desired one. (b) The result of the optimization without constraints. (c) The result of the optimization with planarity constraints.

\[ e(Q, R) = \sum_i f_R(q_i) \]  \hspace{1cm} (4.3)

where \( f_R(q_i) \) is the scalar field value of \( Q \) as described in equation 2.5. This function measures the energy of the fit of the surface in contrast to the displaced points. We can now derive the gradient of it to help the minimization procedure to minimize the problem. This can be easily done thanks to the simplicity of APSS or using the ADOL-C\(^2\) library, which does compute the gradient in an automatic way.

The global optimization process works as follow. First we take a sample and its \( k \)-neighborhood and evaluates the gradient direction for this subset. The gradient will describe a smooth change on the surface where the sample subset is located. With this it is possible to guarantee that the samples will behave in a nice manner without inserting too much noise.

A drawback of optimizing a point cloud only in normal direction is that it can introduce noise and the resulting point cloud is generally not nice. With a planarity constraint, a square distance error of the \( k \) nearest neighbors over the tangent plane of the point to optimize, it is possible to reduce this effect. This situation is depicted in figure 4.4.

We used the conjugent gradient method to minimize the problem and this lead to a good result. Some samples showing the optimization are depicted in figure 4.5. We used an union operation on two very coarse meshes (25 points) and SPSS to enhance the error to a maximum. Some artifacts in (b), (c) are due to the marching cube algorithm which is aware of sharp features. Furthermore in (b) we can clearly see that both PSS does not intersect exactly at the crease line, whereas the optimized version does it.

I did not optimize the computing of the minimization problem, therefore the processing time lies in the region of one minute to optimize one thousand points, and their neighborhoods.

\(^2\)A Package for Automatic Differentiation of Algorithms http://www.math.tu-dresden.de/ adol-c/
4 Accurate Boolean Operations on PSS

![Images of boolean operations](image_url)

**Figure 4.5:** The result of a boolean operation in three representation: (a) exact, (b) without and (c) with operation and re-fitting.

![Images of CSG examples](image_url)

**Figure 4.6:** Two different CSG examples, Further examples can be admired in appendix B.

### 4.4 Results

Some results using multiple boolean operation can be seen in figure 4.6. More figures of CSG made by the BOPSS software can be admired in appendix B. All examples uses the methods described above including the energy based optimization.

### 4.5 Conclusion

As can be seen the optimization of a point set surface is not a trivial matter. In this process we tried several different methods with several different constraints. Furthermore many different methods can be applied to optimize and re-fit the surface. We have seen that a surface optimization on global scale is not practicable, due to the large amount of degrees of freedom. Therefore the problem has to be split into smaller, local problems. These problems can now be viewed separately (if they are far enough from each other) and can also solved in parallel.
Accurate Point Set Simplification

First I present some background about point set simplification and then I present two methods which I implemented.

5.1 Background

There are several methods how to simplify a point set [PG01, WK04, WZK05]. Point set surfaces often describes a complex objects and uses millions or even billions of point samples to represent a given objects surface. As this PSS are big it is often necessary to reduce its complexity to be able to apply a modeling or visualization algorithm efficiently. Therefore some application areas for simplification algorithms are: efficient geometry processing, level-of-details (LOD) rendering, adaptation to certain hardware capabilities (e.g. visualization on mobile devices) and many more.

The main goal of a simplification algorithm is to provide an approximation of a point set surface that uses fewer sample points that the original point set surface and resemble the original surface as close as possible.

The problem of point set decimation can be formalized as follow: Let $S_P = (P)$ be the surface $S_P$ represented by the point set $P$. The goal can be defined in two ways:

1. A first method to define the target is by defining a target sampling rate $n$ which can be defined as the number of points in the target point set. The target can be defined as follow: Find a point set $P'$ with its corresponding PSS $S_{P'}$ such that $|P'| = n < |P|$ and $|S_P - S_{P'}|$ is minimal, where $|P|$ denotes the cardinality and $|S_P - S_{P'}|$ it is a surface distance metric.
5 Accurate Point Set Simplification

2. Another common criterion is to define a target error \( \epsilon \) where \( |S_P - S_{P'}| < \epsilon \) and \( P' \) is minimal.

Of course it is possible to define further criteria for the point set decimation. Further criteria could be normal deviation, point set regularity, etc.

But those problems are hard and therefore most techniques concentrates itself on local solution and uses local error metric which is sub-optimal. Over the last past years several algorithms where developed and those algorithms can be categorized into two groups.

5.1.1 Vertex Clustering

Vertex clustering [LT97] is a relatively simple method for mesh and point set decimation. They are used in many computer graphics applications to reduce the complexity of 3D objects in a fast and reliable manner.

Vertex clustering methods works as follow: A bounding box is placed around the object and then partitioned into a grid [NS04]. The grid resolution is an indication of the final complexity of the decimated mesh. Then each cell is processed separately and a common representative point is computed, which replaces all samples. The position of the new point can be computed in several ways. One possibility is to average the point position over all samples in a cell, another would be to use some error metric. As a last step all representatives are either linked together (in cases of a mesh) or their radii are computed with respect to the new neighborhood or simpler, deduced from the grid resolution. A graphical representation of this method can be found in figure 5.1.

As can be seen there are several drawbacks. In particular the topology of the object can be severely altered. It is possible that unconnected part of an object gets connected, depending on grid size and object geometry. Furthermore the error is only limited by the grid resolution, which gives an upper bound for the geometric error after decimation. This error can vary depending on the orientation of the object inside the bounding box surrounding it. Therefore the quality of the approximation is quite low. Some of the drawbacks can be minimized with adaptive methods like using a kd-tree or BSP-tree as the grid.

Further details about clustering methods based on meshes can be found in [SZL92] and methods based on point clouds in [Pau03].
5.1.2 Iterative Decimation

The iterative decimation works, as the name already suggested, in an iterative way. This means that an decimation operator, e.g. sample removal, edge collapse, etc., is applied iteratively over a mesh or point cloud till a desired quality or sample count is reached. Normally this method is implemented in two phases.

**Phase I - Initialization:** For each region, a vertex or point sample and its neighborhood, the quality after applying the decimation operator is evaluated. This quality can be seen as the priority in which this region is taken into account. All evaluated regions are put into a priority queue and sorted by their priority. Following pseudo code describes this step:

```plaintext
Code 1 Phase I - Initialization
1 foreach(region) {
2     evaluate quality after decimation
3     enqueue(quality, region)
4 }
```

**Phase II - Decimation:** In this phase we pick a region with the highest quality after applying the decimation operator from the priority queue and apply the decimation operator over the given region. After that the whole neighborhood has changed and must therefore be reevaluated and the priority queue entries must be updated. Following pseudo code describes this step:

```plaintext
Code 2 Phase II - Decimation
1 do {
2     Pick the region with the highest quality
3     Apply the decimation operation
4     Update priority queue
5 } while (!SimplificationTargetMet);
```

The advantage of the iterative method is that unconnected parts of an objects are left unconnected and can therefore be used on non-manifold surfaces.

5.2 Point Set Decimation with Clustering

Following the grid based vertex clustering algorithm described in section 5.1.1, I have implemented a decimation algorithm based on clustering. This method is described in section 5.1.1 in all details and will not be further discussed here.

The major difference of this implementation with standard techniques is the following: I have to take into account that the surface is continuous a based upon APSS. Furthermore I added an optimization step to re-fit the surface to its original shape. This optimization/re-fitting step is described in previous chapter.
5 Accurate Point Set Simplification

![Figure 5.2](image)

**Figure 5.2:** An example of a decimation using a clustering technique. In (a) is the original point set surface with approximative 170k samples. (b) shows the decimation with the optimization. The mesh was decimated using a $50 \times 50 \times 50$ grid and resulting in 7600 sample points.

After this step an optimization/re-fitting algorithm is run over the decimated mesh to correct the smoothing introduced by the decimation. This step is described in section 4.3. As can be seen in figure 5.2 the results are convincing. The decimation process for decimating an 170'000 point sample object down to 7600 sample points took about two minutes, most of the time is used for the optimization/re-fitting of the surface.

The representative of each cell is chosen by computing the mean position of each sample in a cell. Furthermore the normal is done in the same manner and the radius of the representative is chosen according to the grid size.

5.3 Point Set Decimation with Error Quadrics

The algorithm presented in the previous section is simple and efficient. However, the resulting sampling is far to be optimal since no geometric error is taken into account. In this section, I will present an adaptive decimation algorithm based on the popular quadric error metric [GH97]. Even though the method described in “Surface Simplification Using Quadric Error Metric” is originally based on meshes it is possible to adapt it to point set surfaces. This method belongs to the category of iterative methods. An implementation of this method ported to point set surfaces is presented in the next subsection.

In the following few paragraphs a short introduction of this decimation method is given:

This method is an iterative methods based on edge contraction. At every iteration an edge is chosen respectively to its quality and then contracted. The quality measurement is done using error quadric which is a symmetric $4 \times 4$ matrix $Q$ attached at each vertex, respectively point sample. The error quadric is simply the squared distance to a plane and defined as follow: Given
a plane $q = (a, b, c, d)^T$ where the plane normal is $n = (a, b, c)^T$ and $d$ is the distance to origin, the square distance of a point $p = (x, y, z, 1)^T$ is defined as

$$\text{dist}(q, p)^2 = (q^T p)^2.$$ (5.1)

A quadric representing of the square distance can be defined as

$$Q_q = \begin{bmatrix}
a^2 & ab & ac & ad \\
ab & b^2 & bc & db \\
ac & bc & c^2 & cd \\
ad & bd & cd & d^2
\end{bmatrix}.$$ (5.2)

To evaluate the error at a point $p = (x, y, z, 1)^T$ with a given quadric $Q$, it is a simple evaluation of a quadric form.

$$\Delta(p) = p^T Q p$$ (5.3)

As a first step of this algorithm we have to evaluate the initial quadric at a given sample point. We will describe here the initialization in case of a mesh. An implementation using point set surfaces is described in the next subsection.

Given a point $p$ where the error quadric shall be evaluated and $n$ faces $q_i$ directly connected to $p$. In this case the resulting quadric $Q$ at $p$ can be computed as a simple sum over all faces $q_i$ which is

$$Q = \sum_i \text{dist}(q_i, p)^2$$ (5.4)

The next step of the algorithm is to initialize the priority queue with all edges that can be contracted. For this we can simply go over all edges, evaluate the error after the contraction and put the result in a priority queue. The error after contraction can be evaluated as follow. Given an edge $e$ with its endpoints $p_0, p_1$ and the respective quadrics $Q_0, Q_1$, the error is

$$\Delta(v) = p^T (Q_1 + Q_0) p$$ (5.5)

at a given point $v$. The point $v'$ which minimizes the error is the result of following linear system:

$$Q_q = \begin{bmatrix}
q_{11} & q_{12} & q_{13} & q_{14} \\
q_{21} & q_{22} & q_{23} & q_{24} \\
q_{31} & q_{32} & q_{33} & q_{34} \\
0 & 0 & 0 & 0
\end{bmatrix}, \quad v' = \begin{bmatrix}
0 \\
0 \\
0 \\
1
\end{bmatrix}.$$ (5.6)
The decimation algorithm using error quadrics to prioritize edge contraction can be summarized as following:

1. Evaluate the quadric $Q$ for every vertex or point sample.
2. Initialize the priority queue with each edge, by evaluating the error while contracting the edge. In this case an edge is a connection of a point with one of its neighbors.
3. Pick the edge with the smallest error from the priority queue and contract the edge. After contraction it is necessary to update the priority of all outgoing edges in the neighborhood. This step is done iteratively till the required decimation quality or desired sampling rate is met.

5.3.1 Implementation and Results

Originally the “surface simplification using quadric error metrics” was designed for triangle meshes. But in our case we only have a point set which represents an implicit surface approximation. Therefore the main problem of porting this algorithm on point set surfaces was how to handle the non-existing edges, respectively the non-existing connectivity between point samples. A trivial solution would be to triangulate the whole point set, but this would wipe out all advantage of a point set. Another solution would be to triangulate the local area of a point with its $k$-nearest neighbors. In the case $k = 6$ the behavior of a point is nearly the same as with a mesh, because a well triangulated mesh has ideally valence 6 everywhere (except at his borders of course).

To evaluate the quadric error on a point cloud we have basically two options: The first option is a local triangulation (fan cloud) [LP02]. In this case we fake a local valence 6 mesh by locally triangulate the point cloud and then evaluate the quadric with this local triangulation. It is obvious that this is not very efficient at all and takes a long time to complete. This method works as expected and results in good decimation. The only drawback is a long computation time.

Another option is is described in [Pau03]. The only different consists in the way how the initial quadrics are computed. Instead of locally triangulate, he takes the $k$-nearest neighbors and uses tangent planes to evaluate the initial quadrics. More precisely: Given a sample point $p$ and its $k$-nearest neighbors $p_i$ where $i = \{0, \ldots, k - 1\}$. The tangent plane is then spanned by $t = p - p_i$ and $b_i = t \times n$ where $n$ is the normal vector at $x$.

After the initial quadric computation the decimation process is done in the exact same way as for meshes described above. Then an optimization/re-fitting process is done over the point cloud to guarantee that the result is as close as possible to the original PSS. This optimization/re-fitting process is described in previous chapter. In figure 5.3 some examples can be seen using a more accurate method than described in [Pau03].
Figure 5.3: (a) A bunny with about 16000 sample points. The bunny was the decimated to (b) 8000 points (50%); (c) 4000 points (25%); (d) 2000 points (12%); (e) 1000 points (6%); (f) 500 points (3%) with optimization/re-fitting.
5 Accurate Point Set Simplification
Conclusions

This chapter concludes this manuscript with a summary of the thesis and its contribution, my impression about the time working on it and also some ideas about future work are given.

6.1 Summary

In this thesis I described a robust algorithm to perform accurate boolean operations on approximating point set surfaces. In particular, the proposed algorithm features a surface re-fitting procedure allowing to optimize a PSS such that it matches a given surface target. I also explored some alternative applications of this optimization algorithm, and showed that it is also well suited for surface simplification. I also presented new MLS techniques to approximate a 1D curve embedded in a 3 dimensional space using either the idea of Hermite interpolation or sphere fitting.

This thesis is implemented in a software platform called “bopss_viewer”. It consist of all methods described in this thesis and has many features built in. A subset of all features is described in appendix A. The software plattform can not only do boolean operations, it also can manipulate objects in an advanced manner. This includes translation, rotation and scaling. But also includes some manipulation on a lower level.

6.2 Impression

Boolean operations on point set surfaces is a field which has not yet be conquered by many scientists. There was nearly no material and paper available about methods to use to achieve
the goal of this thesis. This was a big challenge to do something that not many people already tried and this was also fun for me.

My first challenge was the framework I used. The framework only worked on Linux and had at the beginning of my thesis only little experience with development on a Linux machine and it was hard to find an adequate replacement for an IDE like Visual Studio. I ended up using eclipse\(^1\) in combination with the QT integration\(^2\), subclipse, texclipse, and CDT. After six month of using it I get used to those tools and I must say they are good. The only missing thing yet is a decent debugger.

The Expe framework is a framework designed especially for APSS and optimized to the maximum. There are nice features in it, but for most things one has to dig pretty deep to find something, which is the other side of the medal. Most things in Expe uses template and macros extensively, which makes it very hard to even see which functions a class posses, without looking in some other classes and applying the macros manually. Unfortunately the eclipse intellisense/auto completion feature is not usable with Expe. Besides those technical details, Expe is a solid, fast and reliable framework, which needs some time to mature.

### 6.3 Future Work

As future work, it would be interesting to extend the presented methods and algorithms to other application fields including real-time applications. One interesting example would be to combine the optimization method with level of details (LOD) and develop a progressive point set surface. Another application in this direction would be the morphing of point sets from one object to another one or to apply some surface modifiers.

Another idea would be to further optimize the re-fitting algorithm to achieve a better approximation with shorter computation time. One could think of using CUDA or other methods to parallelize the optimization process.

A last idea for a possible future work is to design a complete modeling tool where the user can model a point set in a similar way as in Maya or 3ds Max.

---

\(^1\)http://eclipse.org

\(^2\)Can be found at http://trolltech.com/developer/downloads/qt/eclipse-integration-download
Program Manual

In this chapter I will present the software platform called bopss_viewer, which serves as an implementation for this thesis. As you may have noticed “bopss” stands for “Boolean Operation on Point Set Surfaces”. This software platform is designed to be used like a 3D modeling tool. It can load, store, and import objects and manipulate them in any way. The manipulation includes translation, rotation and scaling. The main User Interface can be seen in figure A.1 and it should be easy to use.

A.1 System requirements

The software platform was developed on Ubuntu 7.10 using eclipse as the development IDE. It should compile on every GNU/Linux system which has the required libraries (see the README file for further information). It was not tested on Windows (Expe does not build there) nor on Mac OS X. But the later one should compile, if Expe compiles.

A.2 Libraries

For the implementation only a few library were used:

- **QT:**
  Trolltech’s QT was used for the user interface. [Tro].

- **Expe:**
  Expe is the implementation of APSS. It is used as basis for the APSS computation. [Gue]
Figure A.1: The main user interface of the software platform bopss.

- **Expe:**
  This library is used for automatic evaluation of the gradient on a scalar field. [AW]

### A.3 Features

In this section a brief overview of the main features of “bopss_viewer” is presented:

- **Point Set Loading:**
  Two point set formats can be loaded at this time. One is the native Expe format. This format includes the point set with its attributes, e.g. normal, radius, etc., and it is stored in binary or ASCII form. The other format is my format and it has .pscc as file extension. It is also a point set representation, but includes information about the cell complex used. Furthermore it is possible to import meshes in obj format and transform them into point sets.

- **Point Set Storing:**
  At the moment it is only possible to store into the native pscc format.

- **CSG:**
  It is possible to do CSG operations on any two loaded objects. All three basic boolean operation can be found in the “csg” menu or in the toolbar. To execute an boolean operation it is only required that two models are loaded.

- **Surface Inspector:**
  With the surface inspector it is possible to view the implicit surface generated by a point
set in cell complex mode. It can be translated, scaled and rotated. From the toolbox it is also possible to adjust some parameters. Furthermore it is also possible to visualize the implicit surface generated by the curves.

- **Curve and Surface Reconstruction:**
  This feature reconstructs the curves and surfaces using the adapted APSS method for cell complexes.

- **Model Decimation:**
  Each model can be decimated and up-sampled. The method to use and parameters can be set from the toolbox.

- **Further Features:**
  There are several other features in “bopss_viewer”. An incomplete list of additional features follows: EWA splatting, screenshot tool, point set and cell complex inspection (normals, tangent drawing, etc.), . . . .

---

**A.4 Documentation**

The documentation of the source code can be found as a Doxygen documentation in folder “doxygen-doc/html” or by opening “doxygen-doc.html” on the CD-Rom.
CSG - Results

This chapter is a summary of random results of CSG operations.
Figure B.1: Two cubes subtracted from one cube intersected (a) Curves; (b) reconstructed surface;
Figure B.2: Two spheres intersected (a) Cell Complex; (b) curves; (c) reconstructed surface; (d) implicit surface.
Figure B.3: Two low sampled spheres added (a) Cell Complex; (b) curves; (c) reconstructed surface; (d) implicit surface.
Figure B.4: A cube and a cylinder subtracted from each other (a) Cell Complex; (b) curves; (c) reconstructed surface.
B  CSG - Results
EWA Surface Splatting

I have implemented a surface splatting algorithm which directly renders point set surfaces. This method is based on a screen space formulation of the Elliptical Weighted Average (EWA) filter. The method was first published by Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar and Markus Gross in “Surface Splatting” at Siggraph 2001 [ZPBG01].

I implemented this method in a straightforward way as described in [Räis]. The implementation is software only and does not use the GPU at all, except for surface clipping using OpenGL, and it is therefore not very fast. But on a standard modern PC (Intel Core 2 Duo) a point cloud with around 25’000 points can be smoothly visualized.

Another method to do surface splatting is called “EWA Splatting” [ZPBG02]. The difference to the previous surface splatting algorithm is that this method is entirely based in object space and therefore it is possible to make a better use of the rendering pipeline of modern graphic cards. Unfortunately this method seems to be patented (United States Patent 7215340).
C EWA Surface Splatting

**Figure C.1**: A schematic view of the surface splatting pipeline.

### C.1 Implementation

As already said, I implemented this method in a simple way using [Räs] as guidance. I will give a brief description of the implementation here:

The algorithm for surface splatting is based on following pseudo code:

```
for each point P[k]
    project P[k] to screen space;
    determine the resampling kernel rho[k];
    splat rho[k];
}
for each pixel x in the frame buffer {
    shade x;
}
```

The rendering pipeline is designed in such a way that it skips point sample as soon as possible to prevent unnecessary computation. First a backface clipping and a view frustum clipping is done only after these two steps the view-transformation matrices and image plane rasterization are computed. A schematic view of the render pipeline can be seen in figure C.1 and some sample images rendered with this pipeline can be viewed in figure C.2.
Figure C.2: Some sample renderings using screen space splatting.
C  EWA Surface Splatting
APSS Implementations

I made a 2D C# implementation of APSS using ALGLIB Libraries. It is a simple implementation without any optimizations. It is possible to set individual points on a 2D surface, evaluate its normal using the normal propagation approach and finally to project hundreds of points onto the 0-isosurface. It is even possible to view the implicit surface.

If a point cloud has no normal information then it is possible to evaluate the missing normal using a normal propagation algorithm. In this case two steps are needed. The first step is a solution of the minimization problem (2.9) to evaluate all missing normals with a different constraint, namely the Pratt’s constraint [Pra87]. This will yield the normal direction, but not its orientation. The second step consists in propagating the orientation of all normals in a consistent way. This is done using a minimum spanning tree (MST) similar to [HDD+92]. Starting with a single normal orientated by hand, the orientation can then be propagated to the neighborhood. This propagation can be done through locally fitting an algebraic sphere halfway between two points and this over the whole point set.

An screenshot of the application can be seen in Figure D.1. I have implemented it to have a simple 2D framework to play around and to learn how APSS works exactly.

A solid and fast implementation of the complete APSS framework described in the paper is Expe (Experimental Point Set Engine) an open source engine under GPLv2 and can be found at http://graphics.ethz.ch/apss/.
Figure D.1: Implementation of APSS framework in 2D.
Bibliography


Bibliography


Bibliography


