Computational Science and Engineering
(Int. Master's Program)

Technische Universität München

Master's Thesis

The Big Work

Author:	Yoshiyuki Sakai
1st examiner:	Prof. Dr. However Wasit
2nd examiner:	Prof. Dr. However Wasit
Assistant advisor(s):	Prof. Dr. However Wasit
Thesis handed in on:	August 15, 2006

Acknowledgments

If someone contributed to the thesis... might be good to thank them here.

Contents

Outline and Overview

Purpose of the document

Document overview

Part I: Introduction and Theory

CHAPTER 1: INTRODUCTION
This chapter presents an overview of the thesis and it purpose. Furthermore, it will discuss the sense of life in a very general approach.

CHAPTER 2: THEORY
No thesis without theory.

Part II: The Real Work

CHAPTER 3: OVERVIEW
This chapter presents the requirements for the process.

Part I.

Introduction

1. Introduction

Motivation etc.?

1.1. BGCE - who are we?

1.2. Important concepts

1.2.1. Computer Aided Design - CAD

1.2.2. Topology Optimisation

1.3. Project structure

1.3.1. Aims and Goals

1.3.2. Timeline and Structure

1.4. Acknowledgement of supervisors

Part II.

Background Theory

2. Background Theory

2.1. CAD in computers

2.2. Topology Optimisation

2.3. From CAD to Voxels

2.4. From Voxels to a surface representation

2.4.1. Isosurface Contouring

Now when the optimized voxel data was obtained, the next step is to generate a *mesh based geometry*. It will be useful in the further NURBS implementation. In order to achieve it, the data will be represented by a contour of a smooth function, rendering an isosurface. The isosurface allows to visualize Scalar Volumetric Data in 3D. It furthermore permits a mesh representation of the volume data. The mesh can be composed of triangles or quads, according to the algorithm used. There are two main approaches to solve this problem, the most famous one is Marching Cubes.

Marching Cubes

This algorithm takes as an input a regular volumetric data set and extracts a polygonal mesh. It divides the space into cubes, which are defined by the volume information. Each cube has scalar information on its vertices, the value is equal or above a marked isovalue. Therefore each of the eight vertices of a cube can be marked or unmarked. According to these values vertices are drawn on the edges of the cube at calculated points with the use

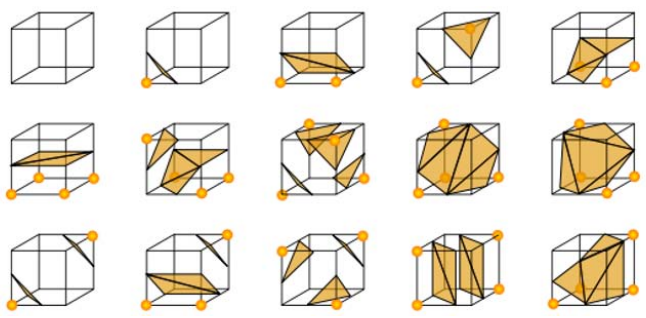


Figure 2.1.: Basis cases of Marching Cubes

of interpolation. A cube that contains an edge is called active. Non active cubes are not further considered in the algorithm.

By connecting the vertices we obtain a polygon on each cube. There are 256 possible scenarios, but most of them are just reflections or rotationally symmetric cases of each other. Therefore there are 15 base cases which represent all the possibilities of the marching cubes (Figure 2.1). The original algorithm presents two main problems. Firstly it does not guarantee neither correctness nor topological consistency, which means that holes may appear on the surface due to inaccurate base case selection. The second problem is ambiguity, which appears when two base cases are possible and the algorithm chooses the incorrect one. These cases can be grouped into face ambiguities and internal ambiguities. There are many extended Marching Cubes algorithms that tackle the problems of the original one, getting rid of the ambiguities and providing correctness.

Dual Contouring

The idea of this algorithm is similar to Marching Cubes, but instead of generating vertices on the edges of the cubes, it locates them inside the cube. Figure 2.5 shows the basic differences in both approaches. The vertices associated with the four contigous cubes are joined and form a quad. The question now is which place inside the cube is the ideal one to insert each vertex. Different dual algorithms are classified according to the answer for this question. Dual contouring generates a vertex positioned at the minimizer of a quadratic function which depends on the intersection points and normals. Therefore the method needs Hermite data to work with.

$$E(x) = x^T A^T A x - 2x^T A^T b + b^T b$$

Where A is a matrix whose rows are the normals and b is a vector whose entries are the product of normals and intersection points. To solve this system, a nummerical treatement is needed. As proposed in [1] the best approach is to compute the SVD decomposition of A and form the pseudo-inverse by truncating its small singular values.

The main advantage of this method over MC is the adquisition of better aspect ratios. On the other hand the need of Hermite Data represents a disadvantage. Furthermore there is no open source algorithm that implements the Dual Contouring scheme.

2.4.2. The VTK Toolbox

Installing VTK

VTK was installed using the Linux platform, for it to be successfully implemented a gcc compiler must be already on the machine. VTK offers the possibility to use Python, TLC or C++ for development. VTK toolbox is actually a C++ library, which is implemented in other languages. We decided to continue the project with C++ since it gives the possibility to explore the original code. A few dependency problems were encountered, nevertheless they were easy to track back. If any problems were to be found at installation time, please refer to the VTK Wiki where the procedure is explained step by step.

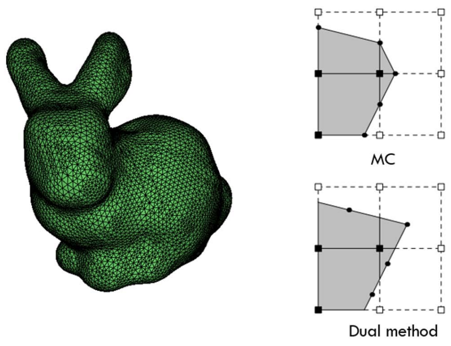


Figure 2.2.: *Right:* The famous Standford Bunny. *Left:* Main difference between MC and Dual methods

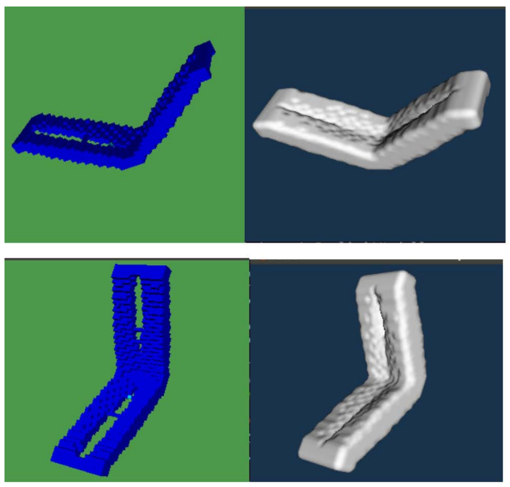


Figure 2.3.: Contour Filtering tool after Implicit Modelling

Implementing the VTK Classes

The VTK toolbox was used in order to implement the algorithms on our optimized data. It is a heavily object oriented toolbox. Our first approach was to use the built in Marching Cubes algorithm, nevertheless it did not work with our unstructured grid data. It just works for ImageData and PolyData . For structured and unstructured grids the tool to render the isosurface is the contour Filter tool. Unfortunately the documentation does not present which algorithm the tool uses. It can be inferred that it is an extended Marching cube algorithm.

The Contour Filtering seemed to work fine but the visualization of our data was still not possible and an intermediate step was needed. We used the Implicit Modelling tool which is a filter that computes the distance from the input geometry to the points of an output structured point set. This distance function can then be ”contoured” to generate new, offset surfaces from the original geometry. It finally allowed visualization but it created one problem. Holes are lost in the process.

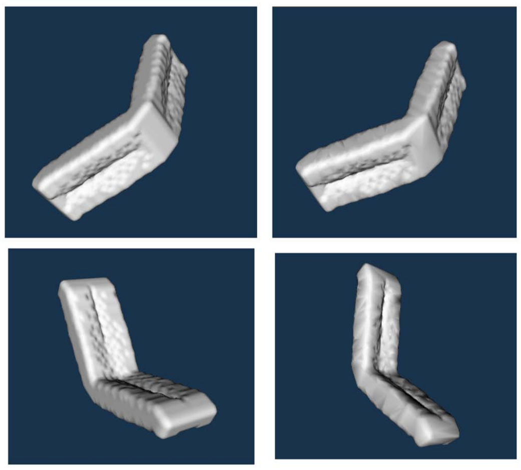


Figure 2.4.: Decimation of triangles. *Top:* 50% *Lower:*90%

A further idea to solve this problem is to convert at the first step the volume data into point data and only then present it to the Contour Filtering Tool. This will be implemented in the next milestone.

The next step was to create a coarser mesh from the fine one. The triangles that represent the isosurface can be reduced with the Decimation tool. A smoothing step is necessary in between to get the new connections right. The top part of figure [2] shows a 50 % reduction of the triangles, a noticeable difference can not be perceived. On the lower part a 90 % reduction is obtained, it is nevertheless still difficult to see a difference. Triangle meshes can be easily coarsened since there are many open source algorithms that simplify the triangles. VTK has the decimation tool which works for 3D triangle data.

2.4.3. The Long Road to NURBS

There are two possible roads to go from the voxel data to the NURBS representation.

Quad Contouring

This approach uses the dual contouring algorithm as first step in order to obtain a quad mesh representation from the voxel data. The first challenge is to implement correctly the algorithm with the ideas presented in [1]. The original marching cubes algorithm is implemented in VTK but the source code is not public, therefore not only an extension of it is needed, but a full implementation. Once this first step is done, the quads will be chosen for the NURBS parametrization. A second step considers multiple smaller quads which have to be combined into one larger patch. This is another challenge, since the remeshing of quad meshes is not as straight forward as with the triangles. Different approaches have been taken in order to achieve this coarsening. In [2] an incremental and greedy approach, which is based on local operations only, is presented. It depicts an iterative process which performs local optimizing, coarsening and smoothing operations. Another approaches, like the one presented in [3] uses smooth harmonic scalar fields to simplify the mesh.

Multiresolution Analysis of Arbitrary Meshes

With this approach there is no need to apply a Dual Contouring algorithm, since it takes as beginning data the triangles from the Marching Cubes. The main concepts are shown in the paper of the same name [4]. It mainly takes a series of intermediate steps which permits a parametrization of data. It includes a partitioning scheme based on the ideas of the Voronoi Diagrams and Delaunay triangulations. Large patches or quads are obtained with this method. Further discussion on this approach is explained in the following section.

Both approaches have not been implemented in open source documentation, therefore it will be a long road to achieve what is required. For the first part the second road was chosen and in case it leads to a dead end, the first way will be taken into consideration.

2.5. From a surface representation to NURBS

2.5.1. Bezier Curves

Bezier curve is a *parametric* curve, which often used for producing a smooth approximation of a given set of data points.

Analytical expression

An analytical expression for the Bezier curve is given by:

$$\vec{B}(t) = \sum_{i=0}^{n} b_i^n(t)\vec{P}_i,$$

where $\vec{P}_i$ is the i–th control point (we have $n+1$ control points). And

$$b_i^n(t) = \left(\begin{array}{c} n \\ i \end{array} \right) (1-t)^{(n-i)} t^i$$

is the i–th Bernstein polynomial of degree n.

Additionally to the expression with the Bernstein polynomials, one can use a recursion formula (**de Casteljau Algorithm**) for the construction of the Bezier–curve, which we will not cover here.

Surfaces

Analogically to Beziercurves, but with $n \cdot m$ Points $\vec{P}_{i,j}$ and the analytical expression

$$\vec{S}(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} b_i^n(u) b_j^m(v) \vec{P}_{i,j}$$

one can define a *Bezier surface*
Bezier–curves and surfaces may be unstable! Minor changes in control points might lead to major global changes!

2.5.2. NURBS basis functions

Extending the idea, described in previous section, one could use NURBS basis functions instead of simple Bezier curves.

Unlike Bezier curves, for the B-spline basis a parameter domain is subdivided with, so-called, *knots*. In particular, given the parameter domain $[u_0, u_m]$ (in 1D), *knot vector* is given by $u_0 \leq u_1 \leq ... \leq u_m$. In most cases $u_0 = 0, u_m = 1$, so we get unit interval for our parameter values. Recall, that, N in NURBS stands for *non-uniform*. This means, that our knots $u_0, ..., u_m$ are not equidistant.

Given *knot vector* $[u_0, u_m]$ and a degree of B-spline p one can find i-th B-spline basis function recursively as follows:

$$N_{i,0}(u) = \begin{cases} 1, & \text{if } u_i \leq u < u_{i+1} \\ 0, & \text{otherwise} \end{cases} \tag{2.1}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \tag{2.2}$$

For $p = 0$ we get just step functions (see fig. 2.5.2), for $p = 1$ we get familiar hat functions (see fig. 2.5.2). Quadratic basis looks a bit more complicated, but also quite intuitive (fig. 2.5.2).

Given these basis functions, Non-Uniform Rational B-Spline curve is given by:

$$C(u) = \frac{\sum_{i=1}^{k} N_{i,n} \omega_i P_i}{\sum_{i=1}^{k} N_{i,n} \omega_i}, \tag{2.3}$$

where k is number of points, $\{P_i\}$ are given control points.

B–splines are have the following properties, which are useful for our problem:

- degree n and number of control points $\vec{P}_{i \cdots m}$ are independent!
- B–Splines only change locally (depends on the degree n) when a control point is changed.

Analogically, one can define B–spline surfaces.

2.5.3. Minimization problem

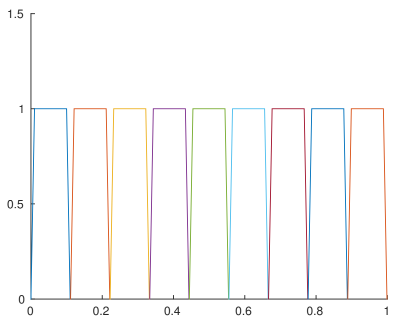
Now, once we defined all necessary tools, we proceed to the fitting problem. **The goal:** Fit in a parametric curve to the set of given data points. In our case our given set of points is a mesh, obtained on a previous step.

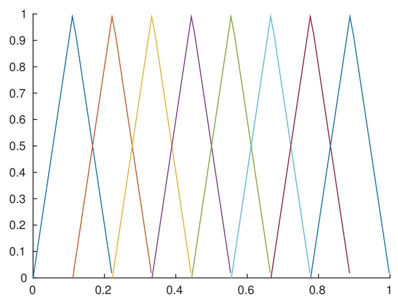
2.5.4. Minimization Problem: Bezier curve

First we want to find a Bezier–curve $\vec{B}_n\,(t)$ of degree n which is approximating a given spline $\vec{s}_m\,(t)$ defined by m points in a optimal way. For this purpose we want to minimize the L2–error. This leads to the minimization problem:

$$\tag{2.4}$$

The function space $\mathbb{B}_n$ is composed of functions of the following form:


(a) B-spline basis for $p = 0$


(b) B-spline basis for $p = 1$

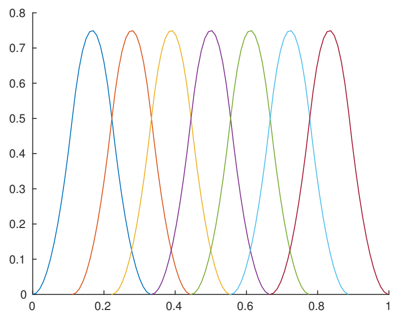

(c) B-spline basis for $p = 2$

Figure 2.5.: B-spline basis functions

Appendix

A. Detailed Descriptions

Here come the details that are not supposed to be in the regular text.