...iteit Groningen Faculteit der Wiskunde en Natuurwetenschappen

Informatica

Interpolating points with given normals by means of a non-uniform B-Spline surface scheme

Kero van Gelder

begeleider: Dr. G. Vegter

december 1997

RuG

Interpolating points with given normals by means of a non-uniform B-Spline surface scheme

Kero van Gelder

December 19, 1997

Abstract

This Master's thesis is about constructing curves or surfaces by interpolating points with a given normal. For this, a scheme using piecewise polynomials built up from B-Splines is used. This scheme was "invented" by Hans-Peter Seidel. Via B-Patches, Seidel's scheme is derived from the Bézier scheme. As for Bézier patches, normally some points are given by the user, which the splines approximate as well as possible. In this thesis the scheme will form a surface that interpolates these user-defined points, where the normal of the surface in these user-defined points is given as well. For Bézier patches the solution is known, for Seidel's scheme a new algorithm, using piecewise polynomials of degree two, is given.

Contents

Introduction

Since ages, people design. In times we don't remember anymore, people designed spears to hunt. They designed and made those items all by themselves. In ages we *do* remember, this process was split up. Designers designed, but other people made the items afterwards. The problem of course is How To Let Them Make What You Designed.

This is where Computer Aided Geometric Design (CAGD) comes in. The designer designs, but now in a mathematically unique way, aided by the computer to visualize the just-made definitions. The greatest advantage is the unique way in which the design is to be interpreted. Thus the factory that will finally produce the designed items, can use this definition to produce without error.

There are also disadvantages. The most irritating one is lack of flexibility. The limitations dictated by the design scheme (or the computer program) can make the designer go real mad.

Description of the problem

In CAGD, often the designer gives a set of **control points**, which is then approached by a curve or surface. See figure 1a.

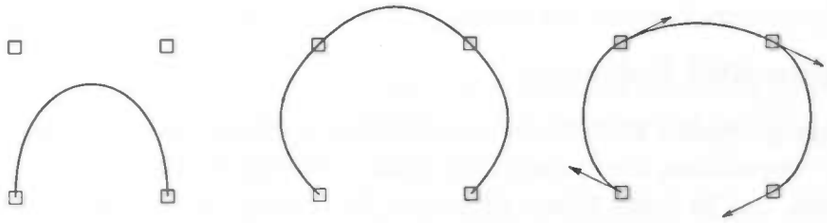

Figure 1: Three ways of designing: (a) approximation, (b) interpolation and (c) interpolation with given tangent of the control points.

Sometimes, the designer wants more: the curve or surface should run *through* the control points with a certain smoothness[1]. See figure 1b. Sometimes the designer wants to specify in what *direction* the curve or surface should run through the control points. See figure 1c.

In 1959 De Casteljeau found a simple polynomial-based scheme, which is very good for approximating a set of control points. This scheme is not so good for interpolating control points, because smoothness of the design is not guaranteed. However, Bézier was the first to publish the scheme, therefore its name.

Purpose of this thesis

In 1990 Hans-Peter Seidel published another scheme, also based on polynomials, but with much better smoothness properties.

[1] Interpolating with straight lines is simple to implement, but too limiting for the designer.

The purpose of this thesis is to find a way to use Seidel's scheme, in such a way that the curve or surface runs through the control points with given normal. From now on, that is called **the Quest**.

Structure of this thesis

The first chapter is about CAGD, to give a general feeling why we are doing this in the first place. Then some theoretical chapters follow, to give sufficient background to understand the last chapter, in which the Quest is solved.

The scheme found by Bézier is explained in chapter 2. Additionally, an algorithm to evaluate polynomials of Bézier's scheme, found by De Casteljeau is given. A generalization of Bézier patches, the so-called B-Patches are treated in chapter 3. The generalization of the De Casteljeau algorithm is also given.

To be able to explain Seidel's scheme, an introduction to B-Splines is given in chapter 4. In chapter 5 the B-Splines are normalized and joined to complete the scheme, accompanied by Seidel's discovery that normalized B-Splines are equal to B-Patches on certain regions of parameter space. These patches are then used to form surfaces in chapter 6.

Then a description of the new "algorithm" is given in chapter 7, followed by the conclusion chapter 8.

An overview of the notation used can be found in appendix A.

Examples and Software

The theory presented here holds for curves and surfaces, but it is valid for hyper-surfaces in higher dimensions, also. Therefore, most theorems in this thesis are given in arbitrary dimensions, but to make things clearer to the reader, many examples in 2D and 3D are given.

The pictures accompanying these examples are mostly generated by a package of software, written during this project (C++, free-ware). This package is a straightforward implementation of Seidel's theories[2]. All efforts went into correctness, few into speed: all degeneracies are accepted and processed as should be, but the package is slow... It provides pictures both on-screen (using Linux™ with `svgalib`) and in PostScript. The results of the latter can be found throughout this thesis. Also `Mathematica`™ input is generated, so 3D output can be viewed without bothering to write a visualization module. For the explanation of the pictures, see appendix B.

Thanks

Thanks to Gert Vegter as my advisor and Meinte Boersma, for scrutinizing this thesis.

[2] in $\mathbb{Q}^2$ and $\mathbb{Q}^3$ only, not $\mathbf{R}^t$, because collinearity can not be computed exactly with reals

1 Principles of CAGD

Computer Aided Geometric Design includes any computer aided design where geometric forms appear. This starts simply with "Draw a Point". Then there are "Draw a Straight Line/Circle/Rectangle through Some Points", follow by "Draw a Curve through/near Some Points". The whole is —in principle— unbounded by the amount of dimensions and amounts of points. In this chapter especially curves and surfaces will have attention, since this thesis is about some specific representation for them.

Design can be done in any number of dimensions, but for humans useful spaces count only up to three dimensions. In those useful spaces design is done with curves, surfaces and volumes. For an object in, say, three dimensions it is required that the object be three dimensional itself to have any meaning. A line or plane is too thin to be an object in three dimensions. (A line with a thickness of .5 is not a line, but a three dimensional rope!)

For the purpose of designing and manufacturing, such an object has to be finite. Thus such an object must have a boundary, which it reaches everywhere and exceeds nowhere[3]. So you could also design a three dimensional object by defining its boundary. This boundary is in fact two dimensional.

Take for example a sphere. One definition is all space within a certain radius of the midpoint (mathematically: $\|x - x_0\|^2 \leq r^2$). Another would be to just define its boundary (mathematically: $\|x - x_0\|^2 = r^2$) and then *interpret* the object as being the entire "inside" of it, where inside means the bounded volume of the two volumes separated by the boundary.

But why? Why should you define the boundary of an object instead of the object itself? There are several reasons for it, most of which can be summarized by *"We only* see *the boundary of an object"*:

- When constructing the object from a big block of material, excess material has to be removed until you reach the boundary. This is exactly the same as reaching the object itself, so there are no problems introduced when manufacturing the object from the definition by boundary instead of the complete body.
- When previewing the object on the computer (we are doing CAGD, after all), you are only seeing the boundary (except in cases with materials like glass).
- Defining 2D surfaces as boundary is more flexible than defining 3D bodies. Even though Constructive Solid Geometry (CSG) can do many things for you, often solid objects are used which stem from a definition by boundary.
- The boundary is often simpler to describe than the object (for the schemes in this thesis a reduction of dimension), so computations are likely to be faster.

Again, to have any meaning, the boundary defining the object *must* be closed. Concrete: In two dimensions, a curve used as boundary has to close up on itself. If it does not, the

[3]There are lots of boundaries: A sphere has a simple, smooth boundary, most fractals have complicated boundaries. These fractals are not geometric forms in the sense of CAGD, so we can ignore them.

boundary is not closed, so there is no two dimensional object. The same holds for 3D: a surface used as boundary must close up on itself.

Of course, there are occasions where the designer really wants to design a piece of paper in three dimensions. In such cases the surface need not meet itself, but it does need a thickness (a piece of paper is not infinitely thin). This is in fact another method of designing, but as it happens, the same polynomial schemes described in this thesis can be used for it.

Surface schemes

The Quest is (mainly) about surfaces used as boundary in three dimensions, so let's have a closer look at surfaces in 3D.

Instead of defining the whole surface as one regular area, it is possible to define the surface in pieces. As long as those pieces fit well enough (the boundary may not have any holes), there is no restriction on the pieces.

A logical step after this is increasing the number of different pieces the designer can put into the surface. In fact, increase the number to infinity to gain flexibility. One mathematical tool to do this are polynomial functions[4]. There are infinitely many polynomial surfaces, which vary from perfectly flat to reasonably sharp. To control the shape of the polynomial, **control points** are used. These control points, as well as some other aspects of piecewise surface schemes, are of course subject to some (more or less intuitive) requirements:

1. **Deviation** The result should not deviate too far from the control points the user gives. Some schemes achieve this by running a line/surface actually through the control points. More often the design is guaranteed to be within the convex hull of the control points.

2. **Uniqueness** The uniqueness mentioned in the introduction; this is extended to affine invariance: the design can be translated, scaled and rotated in any direction and as far as desired, without loss of relative shape (when scaling in a certain direction, all relative distances in that direction remain the same).

3. **Smoothness** The ability to design very smooth objects, but with the ability to make sharp corners as well.

4. **Locality** The ability to perform local changes. Perfecting details should not disturb the rest of the design.

5. **Normal** Sometimes, when a design runs through the control points, designers want to be able to specify the normal of the surface at the control points as well.

[4] referred to as "polynomials" from now on.

Conditions 1 and 2 are absolutely required, without them the surface scheme will never be of practical use. Requirement 3 is mathematically called **Geometric Continuity**, denoted by GC^q, which means the q^{th} derivative is continuous. If a curve/surface is GC^0, then at least there is no discontinuity in it, so it can be used as closed boundary. If a curve or surface is GC^1 or GC^2, then the boundary will become smooth. For practical use, GC^2 is already *very* smooth, most often GC^1 is smooth enough. It is not always *easy* to let two pieces meet with a certain smoothness, as will be shown in both chapter 2 and chapter 3. Requirement 4 can be satisfied by allowing movement or addition of control points (which is not always easy either).

Condition 5 is in fact an extension of condition 1. As said under that condition, most schemes approximate the control points. That makes it difficult to interpolate, even worse when the normal is given as well. For Seidel's scheme, this thesis helps.

2 Bézier Patches

In this chapter Bézier patches are defined and examined.

The scheme by The Casteljeau and Bézier uses polynomials over $\mathbb{R}^s$ with values in $\mathbb{R}^t$. $\mathbb{R}^s$ is the parameter space, $\mathbb{R}^t$ is the design space. Without loss of generality, this can be restricted to polynomials over $\mathbb{R}^s$ with values in $\mathbb{R}$ (because the t dimensions in $\mathbb{R}^t$ are independent).

Over $\mathbb{R}^s$ the Bernstein basis is set up, which can construct any desired polynomial. The advantage of the Bernstein basis is that the coefficients can be used as control points in $\mathbb{R}^t$.

The polynomials are restricted to a simplex in $\mathbb{R}^s$, so that in $\mathbb{R}^t$ a Bézier patch is created. These patches can then be connected to form a (hyper)surface. Problems arise when solving the Quest (section 2.3).

First, three basic definitions follow, then the Bézier patches are defined, followed by the De Casteljeau algorithm in section 2.1. All of these are then enlightened with examples and also some general properties are given in section 2.2.

Definition 2.1 (Multi-index) *A multi-index $\beta \in \mathbb{N}_0^s$ is a vector of length s with each index in $\{0, 1, \ldots\}$ The* **weight** *$|\beta|$ of a multi-index is the sum over all indices. The set of multi-indices with length s and weight n is denoted by $\Gamma_{s,n}$.*

So all indices of a multi-index $\beta \in \Gamma_{s,n}$ are in $\{0, 1, \ldots, n\}$. $\Gamma_{s,n}$ contains $\binom{n+s-1}{n}$ multi-indices. Two examples: $(1, 2, 3) \in \Gamma_{3,6}$ and $(0, 1, 0, 1, 0) \in \Gamma_{5,2}$.

Definition 2.2 (Barycentric coordinates) *Let $\triangle = [t_0, t_1, \ldots, t_s]$ be a simplex[5] in s dimensions. Then the barycentric coordinates $\lambda_i(u)$ of $u \in \mathbb{R}^s$ w.r.t. $\triangle$ are uniquely given as:*

$$\sum_{i=0}^{s} \lambda_i(u) t_i = u \tag{1}$$

$$\sum_{i=0}^{s} \lambda_i(u) = 1 \tag{2}$$

These barycentric coordinates can be computed as follows:

$$\lambda_i(u) = \frac{\mathrm{d}\,(t_0, \ldots, t_{i-1}, u, t_{i+1}, \ldots, t_s)}{\mathrm{d}\,(t_0, \ldots, t_s)}$$

where $\mathrm{d}(u_0, \ldots, u_s) = \det \begin{pmatrix} 1 & \ldots & 1 \\ u_0 & \ldots & u_s \end{pmatrix}$, which is a square matrix, because all vectors u_i have length s.

[5] A simplex is the area/volume within a set of $s+1$ affinely independent points in $\mathbb{R}^s$. For example in 2D, three points which are not on one line and thus form a triangle. Another example is a tetrahedron in 3D.

Definition 2.3 (Bernstein Polynomials) *Let $\triangle = [t_0, t_1, \ldots, t_s]$ be a simplex in $\mathbb{R}^s$ and let $\lambda_i(u), i \in \{0, 1, \ldots, s\}$ be the barycentric coordinates of u with respect to $\triangle$. Then the Bernstein polynomials B_β of degree n w.r.t. $\triangle$ are defined as:*

$$B_\beta^\triangle(u) = \binom{n}{\beta} \prod_{i=0}^{s} \lambda_i^{\beta_i}(u)$$

with $\beta \in \Gamma_{s+1,n}$ and $\binom{n}{\beta} ::= \frac{n!}{\prod_{i=0}^{s} \beta_i!}$.

See figure 2 for some examples of Bernstein polynomials with $s = 1$ and $n = 2$.

It can be verified that the Bernstein polynomials adhere to the following recursive property:

Theorem 2.4 (Recursive Bernstein polynomials) *The above-defined Bernstein polynomials $B_\beta^\triangle$, $\beta \in \Gamma_{s+1,n}$ can be constructed as follows:*

$$B_0^\triangle = 1 \tag{3}$$

$$B_\beta^\triangle = \sum_{i=0}^{s} \lambda_i(u) B_{\beta - \hat{e}_i}^\triangle, \ \beta \in \Gamma_{s+1,m}, \ 0 < m \leq n \tag{4}$$

where $\hat{e}_i \in \Gamma_{s+1,n}$ with index $\beta_i = 1$ and the values of all other indices equal to zero.

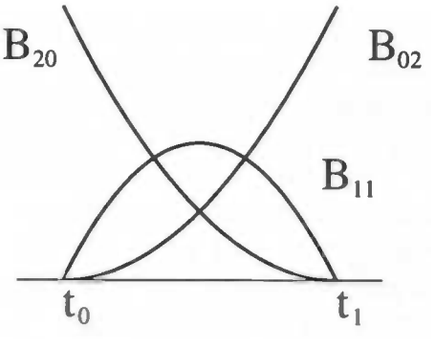

Figure 2: Three Bernstein polynomials: B_{20}, B_{11} and B_{02}, all three patched to $\triangle = [t_0, t_1]$.

These Bernstein polynomials are clearly homogeneous polynomials of degree n. There are exactly $\binom{n+(s+1)-1}{n}$ of them and they are linearly independent. So a basis for polynomials of degree n over $\mathbb{R}^s$ is formed by the Bernstein polynomials[6]. This leads to:

[6]If you have difficulties seeing this: There are $\binom{n+(s+1)-1}{n}$ of these polynomials, because all β's in $\Gamma_{s+1,n}$ have a corresponding Bernstein polynomial (no more, no less). Furthermore, that is precisely the amount of polynomials needed for a basis. Write down a polynomial in its monomial form: as a sum of terms with different amounts of x_i's in it (e.g. $ax_1^2 + bx_1x_2 + cx_2^2 + dx_1 + ex_2 + f$ in two dimensions). This can also be written with extra x_0's inserted when there are not enough x's in the term (thus $ax_1^2 + bx_1x_2 + cx_2^2 + dx_0x_1 + ex_0x_2 + fx_0^2$). It is easily seen that the amount of terms is exactly the same as the amount of polynomials in the basis.

Definition 2.5 (Bézier Representation) *Let $\triangle$ be a simplex in $\mathbb{R}^s$ and let $F : \mathbb{R}^s \to \mathbb{R}^t$ be a polynomial. Then the Bézier representation of F w.r.t. $\triangle$ is:*

$$F(u) = \sum_{\beta \in \Gamma_{s+1,n}} B_\beta^\triangle(u) c_\beta$$

where the $c_\beta \in \mathbb{R}^t$ are **Bézier control points**, which form the **Bézier control net**. A **Bézier patch** is the polynomial F in its Bézier representation restricted to $\triangle$. See figure 3 for two examples of Bézier patches (see appendix B for an explanation of the signs used).

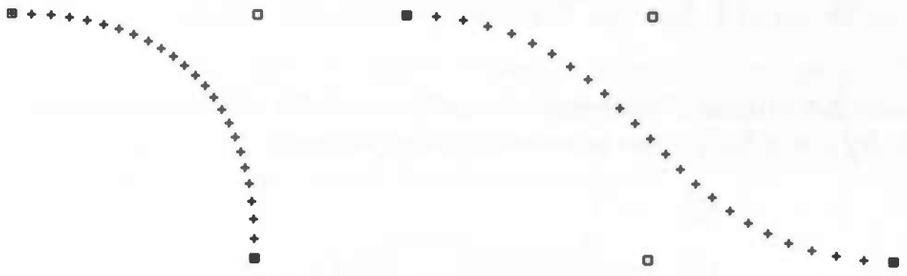

Figure 3: Two examples of two dimensional Bézier patches ($s = 1$ and $t = 2$): (a) second and (b) third degree. The □'s are control points and the +'s are evaluated points on the curve.

The control points are points given by the designer, which the (hyper)surface should approach as good as possible. Due to the choice of the Bernstein polynomials as basis functions, a Bézier patch remains entirely inside the convex hull of its control net. Even better, if you evaluate the polynomial in one of the vertices t_i of the simplex, you end up exactly in a control point. These two properties make Bézier patches so useful.

The maximum of such a Bernstein polynomial B_β lies at $u = \sum_{i=0}^{s} \frac{\beta_i t_i}{n}$. Thus you could **associate** the control point c_β with that point u in the parameter space[7]. See figure 4. From now on, two control points β and γ are called **adjacent** or each others **neighbors** if $\beta + \hat{e}_i - \hat{e}_j = \gamma$, $i \neq j$ (so c_{110} is a neighbor of c_{200}, but c_{020} and c_{011} are not). In figure 4b a set of control points is associated with the simplex of figure 4a and neighbors are connected with a line.

2.1 The De Casteljeau Algorithm

Due to recursive property (2.4) of the Bernstein polynomials, it is possible to set up a recursive scheme for evaluating a point on the surface of a Bézier patch. The control points are used in the basic step and the barycentric coordinates are used in the recursion. The Bernstein polynomials itself do not appear in the algorithm.

[7]This is a very useful association. The control points are regularly associated over the simplex, with s of them exactly on the vertices of the simplex and a series of others exactly on the edges.

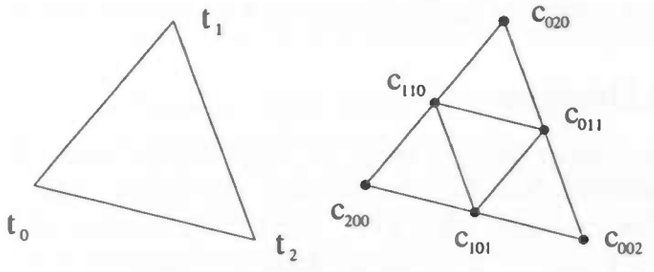

Figure 4: Association of control points with a simplex: (a) the simplex (b) the associated control points. Two control points connected with a line are neighbors.

Theorem 2.6 (De Casteljeau) *Let* $F(u) = \sum_{\beta \in \Gamma_{s+1,n}} B_\beta^\triangle(u) c_\beta$ *be a Bézier patch w.r.t.* $\triangle$. *Then* $F(u) = c_0^n$, *where the sets* $\{c_\beta^l \mid \beta \in \Gamma_{s+1,n-l}\}$ *are recursively determined for* $0 < l \leq n$ *by:*

$$c_\beta^0 = c_\beta, \ \beta \in \Gamma_{s+1,n} \tag{5}$$

$$c_\beta^l = \sum_{i=0}^{s} \lambda_i(u) c_{\beta+\hat{e}_i}^{l-1}, \ |\beta| + l = n. \tag{6}$$

This has been proven several times in several ways. See [Sei90] for an elegant proof using **blossoming**[8].

Note that the control points are used, the Bernstein polynomials do not appear at all in the De Casteljeau algorithm. For a demonstration, see figure 5.

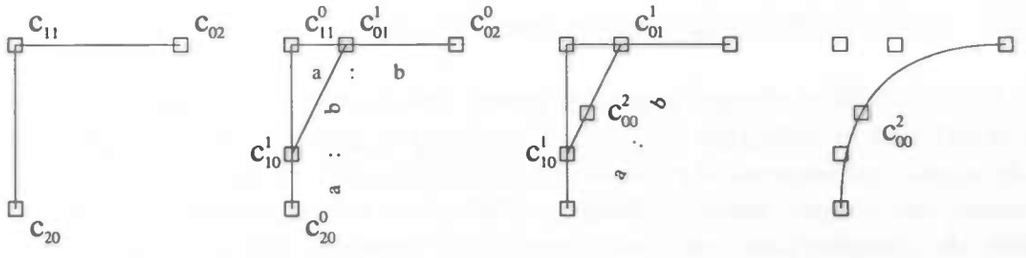

Figure 5: A demonstration of the De Casteljeau algorithm; (a) the control points (b) c_β^1 derived from c_γ^0 (c) c_0^2 derived from c_β^1

The De Casteljeau algorithm can also be used for subdivision, differentiation and computing the tangent plane (see [Far93]). Only the last one is given here:

[8]The mathematical name is **polarization**. If you are not familiar with this, look in [Far93] for a proof.

Corollary 2.7 (Tangent Plane) *The tangent plane at $F(u)$ is spanned by the points $c_{\hat{e}_i}^{n-1}, 0 \leq i \leq s$, as computed by De Casteljeau's algorithm (theorem 2.6).*

2.2 Useful Designs

Now how do these Bézier patches look? $\mathbb{R}^t$ is the design space. $\mathbb{R}^s$ is the parameter space, which is not visualized, but helps to compute the design. Also, s is the dimension of the designed item! For example, when $s = 1$ and $t = 2$ curves in two dimensions are modeled, whereas with $s = 1$ and $t = 3$ curves in three dimensions are produced and whereas with $s = 2$ and $t = 3$ surfaces in three dimensions are made. See for examples respectively figures 3, 6 and 8.

Note that with these settings of s and t all useful designs for humans can be made. If $s \geq t$ the results are so trivial (the convex hull of the resulting object *is* the object) other methods of design are easier. For $s = 0$ only points are modeled and for $t \geq 4$ design in four or more dimensions is done, which humans can not use.

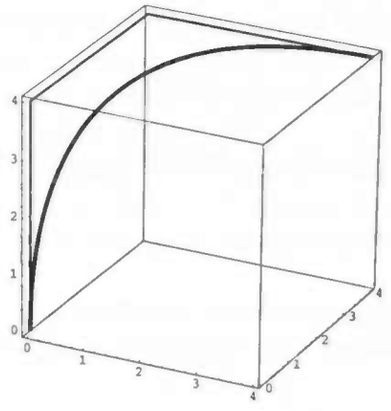

Figure 6: A Bézier patch over $\mathbb{R}^1$ with values in $\mathbb{R}^3$.

In about 99% of all cases (educated guess), the designer is not interested in a polynomial as mentioned in definition 2.5. If he[9] wanted the polynomial, he would have used the polynomial, not some set of points which describe it, even though it is a perfect description. Instead, the designer wants some curve, of which he knows more or less how it should look. Then the designer sets some control points to approach this curve and adjusts what he does not like. The fact that this *can* be represented as polynomials afterwards does not interest the designer (but it does interest the implementor of any computer program for CAGD).

[9]Replace with "she" or "he/she" if you like throughout this thesis

2.2.1 (Hyper)surfaces built from Bézier patches

As mentioned in chapter 1, it is possible to use surfaces as the boundary of a three dimensional object.

A Bézier patch is just a polynomial in $\mathbb{R}^s$ restricted to a simplex with values in $\mathbb{R}^t$. As (hyper)surface, this is in no way a complete boundary of an object. Several patches are to be stitched together (have to be GC^0) in order to close the boundary. Fortunately, this can be done easily: the edges of two Bézier patches are exactly the same when all control points for those edges are used by both patches (and in the same order). This comes from the following theorem:

Theorem 2.8 (Boundary Patches) *Given a Bézier patch over $\triangle = [t_0, t_1, \ldots, t_s]$. The restriction of that Bézier-Patch to the face $[t_0, \ldots, t_{i-1}, t_{i+1}, \ldots, t_s]$ is a Bézier patch with control points c_β with $\beta_i = 0$. This Bézier patch is one dimension lower than the original patch.*

The proof of this is the fact that $\lambda_i(u) = 0$ when u is restricted to the mentioned face, so $B_\beta(u) = 0$ if $\beta_i \neq 0$. In figure 8 the thick drawn "diagonal" curve is such a boundary patch.

Look at two adjacent Bézier patches over $\mathbb{R}^s$ with values in $\mathbb{R}^t$. When they use the same control points associated with their common face, the same Bézier patch (which can be constructed over $\mathbb{R}^{s-1}$ with values in $\mathbb{R}^t$) forms their boundary. Thus two Bézier patches connect in a GC^0 fashion.

2.2.2 Computing Bézier patches

Neither the recursive definition of Bernstein polynomials, nor the evaluation by De Casteljeau is of particular interest for an implementation, because (re)evaluating a polynomial can be done much faster than going through the whole recursion (See for example [HB94]). However, both recursions provide many clues as to how Bézier patches behave. They are very useful recursions for the researchers.

2.3 The Quest solved for Bézier patches

The method to connect two Bézier patches from theorem 2.8 does not provide any smoothness better than GC^0. The following two subsections describe GC^1-continuity requirements for adjacent Bézier patches. Also, these requirements are fit into a method to use the Bézier patches for interpolation with given normal.

Note that the Geometric Continuity is required in design space. Since patches are directly related to the function values of the Bernstein-Bézier representation, you might expect there has to be some sort of continuity in the parameter space as well. Within one simplex (and thus within one patch), this is satisfied by definition. Where two Bézier patches are connected, there happens to be no requirement on the continuity in the parameter space (this is obvious, because a totally different set of basis-polynomials is used

when going from one simplex to another, which would be discontinuous from the set in the neighbor-patch if laid out in a triangulation like in definition 3.6).

The requirements are given for one and two dimensional parameter space separately. The distinction between dimensions of the parameter space has to be made, because no general condition for smoothness is known. Note that the requirements are on the control points, which are in design space!

2.3.1 One dimensional parameter space

In the one dimensional parameter space ($s = 1$), the user gives a series of $n+1$ control points to create a Bézier patch of degree n. This patch runs through the begin- and endpoints given by the user. Moreover, at the begin- and endpoint the tangent line is equal to the line through the first and second respectively last and pre-last points (as follows from corollary 2.7).

The property about the tangent line can be used to solve the Quest for Bézier patches in a one dimensional parameter space: If the designer/user gives two points with tangent line through them, set the first and last control points to be those two user-defined points and lay the second and pre-last control points on these tangent lines. The obtained four control points define a third degree polynomial curve in design space, which runs through the user-defined points with specified tangent line.

For the special case where the design space is two dimensional, there is a high probability it is possible to let the second and pre-last control point be the same, because two the two tangent lines cross almost always[10], thus reducing the curve to a second degree polynomial. See figure 7.

Figure 7: (a) Given two points with tangent line, (b) extend the lines until they meet, (c) the (three) resulting control points.

2.3.2 Two dimensional parameter space

Like for one dimensional parameter space, the tangent plane through control points associated with the the corner vertices is the plane spanned by that control point itself and its direct neighbors (it can be computed that the $c^{n-1}_{\hat{e}_j}$ from corollary 2.7 evaluated in $u = t_i$ are the control points $c_{(n-1)*\hat{e}_i+\hat{e}_j}$).

[10]There is a possibility the curve runs through the control point in the opposite direction: if one of the two arrows in figure 7 points in the other direction. If the two arrows are parallel, there is no second degree solution.

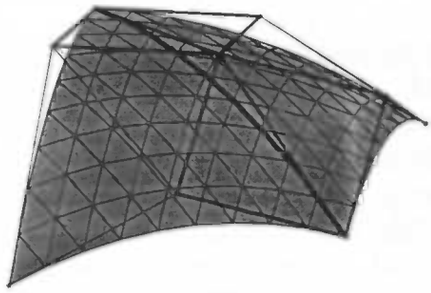

Figure 8: Two adjacent Bézier patches, which are connected in a GC^1 fashion. Thicker lines are used to connect the control points that have to be in one plane (four by four, not all seven) and an even thicker line is used for the common border of the two patches.

This property and some research, done by Farin (see [Far93]) resulted in the complete requirement for GC^1 in two dimensional parameter space:

> Consider two adjacent control points, both associated with the common edge of two triangles. This pair of control points has two common neighbors, one in each triangle sharing the edge. The two Bézier patches are GC^1 if the four control points lie in one plane.

For three dimensional design space, see an example in figure 8, where the thicker part of the control net exists of two airs of coplanar triangles.

However, when this method is used to make a complete surface, this forces the Bézier patches to be of third degree if there is no normal specified and even fourth degree when the normal *is* specified; see again [Far93]. The computations to construct these patches are tedious.

2.4 Conclusion

Bézier patches form a partially useful scheme:

- When curves in arbitrary dimensional design spaces are used, all requirements mentioned in chapter 1 can be satisfied.
- When surfaces are used, the given control points can be approximated very well.

However, when surfaces are used and smoothness is required (GC^1 or better) Bézier surfaces of degree three are needed and the construction of these takes a lot of complicated computations. Worse, when a normal of the surface is specified by the designer, the degree of the Bézier patches rises to four and construction becomes even more complex.

The next chapters will show that there are solutions to the Quest, based upon other schemes, which produce piecewise polynomial surfaces of a degree lower than four.

3 B-Patches

A generalization of Bézier patches is formed by B-Patches. Instead of using plain simplices in parameter space, clouds of points are assigned to the vertices of the simplices, thus creating a knot arrangement. From these knot arrangements polynomials are constructed in a way similar to the Bernstein polynomials. These polynomials again form a basis, so that B-Patches can be built. The hope is to gain flexibility w.r.t. layout of the control points by using these clouds, may be improving the smoothness properties of generated surfaces. But as will be found, the smoothness properties of B-Patch surfaces is not better than for Bézier surfaces.

The De Casteljeau algorithm generalizes to the De Boor algorithm, explained in section 3.2. In section 3.3 the B-Patches are analyzed and examples are given.

3.1 Assigning clouds to points

When trying to assign a cloud to a vertex of a simplex, surprisingly few restrictions are necessary, namely one.

Definition 3.1 (Knot Arrangement) *Let $\triangle = [t_0, t_1, \ldots, t_s]$ be a simplex in $\mathbb{R}^s$. Label each t_i now $t_{i,0}$ and add n other knots (points) $\{t_{i,1}, \ldots, t_{i,n}\}$ to $t_{i,0}$ to form $s+1$ clouds. This set of clouds is a knot arrangement $\mathcal{A}$ assigned to $\triangle$ when all sets of points $\{t_{0,\beta_0}, t_{1,\beta_1}, \ldots, t_{s,\beta_s}\}, \beta \in \Gamma_{s+1,m}, 0 \leq m \leq n$ are affinely independent.*

It is possible to assign closely spread clouds to vertices, but also very widely spread. An example of a knot arrangement with both closely and widely spread clouds is given in figure 9.

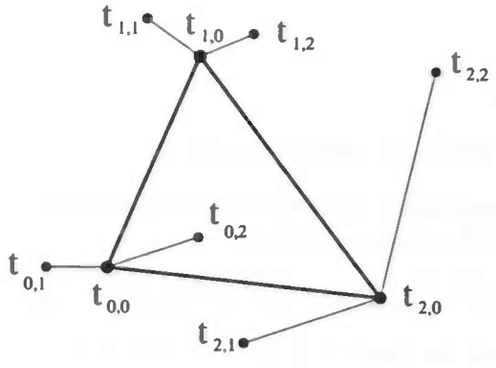

Figure 9: A knot arrangement. The cloud assigned to t_1 is less spread than the cloud assigned to t_2.

Definition 3.2 (Normalized B-Weights) *Let $\lambda_\beta(u)$ be the barycentric coordinates of u w.r.t. the simplex $\triangle_\beta = [t_{0,\beta_0}, t_{1,\beta_1}, \ldots, t_{s,\beta_s}], \beta \in \Gamma_{s+1,m}, 0 \leq m \leq n$. The normalized B-Weights over the knot arrangement $\mathcal{A}$ are the polynomials $B^{\mathcal{A}}_\beta(u)$, which are defined recursively by*

$$\begin{aligned} B^{\mathcal{A}}_0(u) &= 1, \\ B^{\mathcal{A}}_\beta(u) &= \sum_{i=0}^{s} \lambda_{\beta-\hat{e}_i,i}(u) B^{\mathcal{A}}_{\beta-\hat{e}_i}(u), \ \beta \in \Gamma_{s+1,m}, m \leq n \end{aligned} \tag{7}$$

where terms with a negative index in their multi-indices are set to zero.

The normalized B-Weights $B^{\mathcal{A}}_\beta$ are clearly homogeneous polynomials of degree n in the barycentric coordinates of u. There are exactly $\binom{n+(s+1)-1}{n}$ of these, which are linearly independent[11]. Thus the normalized B-Weights form a base for the space of polynomials of degree n (like in the Bézier case, theorem 2.3). This leads to the following definition:

Definition 3.3 (B-Patch Representation) *Let $F : \mathbb{R}^s \to \mathbb{R}^t$ be a polynomial of degree n. Let $\mathcal{A}$ be a knot arrangement assigned to $\triangle$. Then the B-Patch representation of F w.r.t. $\mathcal{A}$ is the unique decomposition of F w.r.t. the basis $\{B^{\mathcal{A}}_\beta \mid \beta \in \Gamma_{s+1,n}\}$*

$$F(u) = \sum_{\beta \in \Gamma_{s+1,n}} B^{\mathcal{A}}_\beta(u) c_\beta$$

The coefficients $c_\beta \in \mathbb{R}^t$ are called **B-Patch control points**, which form the **B-Patch control net**. The restriction of the representation to $\triangle$ is called a **B-Patch**.

Even though the maxima of the B-Weights in the basis are not perfectly regularly spread, like the maxima of the Bernstein polynomials, we still hold to the association and adjacency of control points with a simplex. See page 12.

3.2 The De Boor algorithm

For B-Patches, the De Casteljeau algorithm for evaluation generalizes to the De Boor algorithm:

Theorem 3.4 (De Boor Evaluation) *Let $F(u) = \sum_{\beta \in \Gamma_{s+1,n}} B^{\mathcal{A}}_\beta(u) c_\beta$ be a B-Patch over $\mathcal{A}$ and let $u \in \mathbb{R}^s$. Let $\lambda_\beta(u)$ be the representation of u in barycentric coordinates w.r.t. $\triangle_\beta$. Then $F(u) = c^n_0$, where the sets $\{c^l_\beta \mid \beta \in \Gamma_{s+1,n-l}\}$ are determined for $l = 0, \ldots, n$ as follows:*

$$c^0_\beta = c_\beta, \ \beta \in \Gamma_{s+1,n} \tag{8}$$

$$c^l_\beta = \sum_{i=0}^{s} \lambda_{\beta,i}(u) c^{l-1}_{\beta+\hat{e}_i}, \ |\beta| + l = n. \tag{9}$$

[11] Due to the requirement on the layout of the knots in the knot arrangement, the barycentric coordinates $\lambda_{\beta-\hat{e}_i}$ are uniquely defined. Because of that, the B-Weights are uniquely defined. Also, no two barycentric coordinates $\lambda_{\beta,i}$ and $\lambda_{\gamma,j}$, $i \neq j$ are the same, which guarantees linear independence.

Just as De Casteljeau's method, this algorithm comes in several flavors: next to the given version for evaluation, there are versions of this algorithm that can be used for subdivision and computing the tangent plane of the B-patches.

Corollary 3.5 (Tangent Plane) *The tangent plane through $F(u)$ is spanned by the control points $\mathbf{c}_{\hat{e}_i}^{n-1}$ evaluated at u, as computed by De Boor 3.4.*

This is exactly the same as the tangent plane for Bézier patches (corollary 2.7), which could be expected, since c_0^n is a point on the patch as well (theorem 2.6 versus 3.4).

3.3 Useful Designs

This B-Patch representation looks a lot like the Bézier representation (2.5), but with knot arrangement $\mathcal{A}$ instead of just a simplex (in parameter space) and normalized B-Weights instead of Bernstein polynomials. In fact, when the clouds are contracted to one point, the Bézier representation is regained.

The more a cloud is spread, the more the B-Patch deviates from the Bézier patch. See figure 10, which has the same setup as the Bézier patch in figure 3a. In the first picture the clouds are only a little spread, which is visible because the line that connects some of the evaluated points has its endpoints close to the control points. In the second picture more widely spread clouds are used: The endpoints are clearly further from the control points. Further research can be done about the behavior of B-Patches with the use of varying clouds.

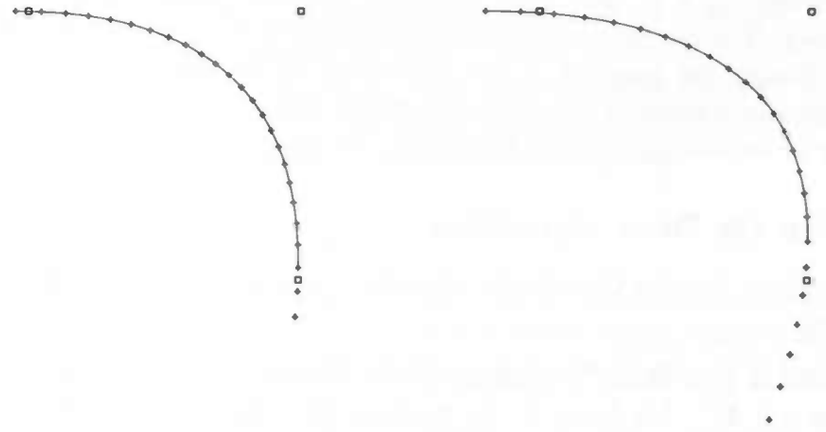

Figure 10: Two B-Patches with the same control points as figure 3a: (a) with closely spread clouds, $\{1, 2, 3\}$ and $\{23, 24, 25\}$; (b) with widely spread clouds, $\{1, 4, 7\}$ and $\{19, 22, 25\}$.

3.3.1 Geometric Continuity

There has not been done much research on the problem of connecting B-Patches together to a curve or surface. What *has* been found until now, will be repeated here, to show what the problems are.

One of the problems is the loss of clear and sharp edges of the patch. The edges are not clear, because a B-Patch does not run through its control points associated with the vertices of the simplex like a Bézier patch.

The edges are not sharp, because the clouds assigned to the vertices of the simplex diffuse the edges of the simplex and thus of the patch (in figure 10 the evaluated points which have no line drawn through them, are still evaluated in parameter values inside the knot arrangement $\mathcal{A}$ assigned to simplex $\triangle$, be it outside $\triangle$ itself). Even though the edges have been *defined* strictly w.r.t. $\triangle$ in definition 3.3, I am not sure if that definition will be *practical.*

A third problem is the assignment of clouds: for two adjacent patches, should the clouds that are used by their associated simplices be of the same layout or is that irrelevant? If they have to be the same, it is convenient to define a triangulation[12]:

Definition 3.6 (Triangulation) *A set of simplices* $\mathcal{T} = \{\triangle_I = [t_{i_0}, t_{i_1}, \ldots, t_{i_s}] \mid I = (i_0, i_1, \ldots, i_s) \in \mathcal{I} \subset \mathbb{N}_0^{s+1}\}$ *is a triangulation of a bounded domain* $D \subset \mathbb{R}^s$ *when*

$$\bigcup_{\triangle \in \mathcal{T}} \triangle = D \quad \textit{and} \quad \forall_{I \neq J \in \mathcal{T}} I \bigcap J = \emptyset \textit{ or a common face.}$$

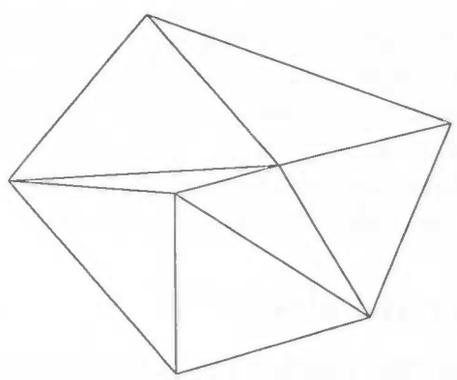

Figure 11: A triangulation.

This is convenient, because it forces the simplices associated with the patches to be adjacent as well, thus they will use the same vertices and clouds.

So now for all adjacent pairs of patches in a surface that represents a closed boundary, there has to be an associated pair of adjacent simplices in the triangulation. This effectively means the triangulation has to be the same in some topological fashion. The exact shape of the simplices does in no way need to be the same as the shape of the patches, because they

[12] simplexification does not sound very well, does it?

are placed in different spaces. Thus there is a lot of freedom to set up the triangulation, but not always enough[13]. See also section 6.1.4 and remark (7.5).

Now let's go back to what *has* been found out about placement and smoothness of patches.

As we can read in [Sei90]:

> Suppose that the simplex $\triangle \subseteq \triangle_\beta \forall \beta$ [...], then if $u \in \triangle$, then $F(u)$ lies inside the convex hull of the associated control net.

In words: if the clouds assigned to the vertices of the simplex lie outside that simplex, then $F(u)$ lies inside the convex hull of the control net.

This seems useful for the designer, because this is the B-Patch version of the property on page 11 that holds for all points $u \in \triangle$ for Bézier patches. That property guarantees that the curve or surface approaches the control points well enough. It appears to hold as well for B-Patches, but: for adjacent simplices, if the clouds used for the corner vertices are the same, these clouds can not possibly be outside these adjacent simplices as well.

Then perhaps different clouds should be used? An immediate problem then is about the edges of the patches in design space. Where are those edges exactly? The corners of the patches are *not* the control points associated with the corner vertices and it is unknown yet, where the edges are. This makes it impossible for now to let two patches even be GC^0...

For know it seems to me that using the same cloud for vertices shared by simplices poses more problems than are solved. By using different clouds, the need for a triangulation vanishes. For one dimensional parameter space I could provide a solution, more or less analogous to what I found in chapter 7. This returns Bézier patches, however, be it described in another fashion than just converging clouds, so just use Bézier patches. I have no idea about solutions for higher dimensional parameter spaces.

3.4 The Quest for B-Patches

An analogue for gluing patches together like in section 2.2 for Bézier patches is omitted in [Sei90]. Considering the statements above, this is understandable; there is almost nothing known. It is a little strange in the light of the coming theories about B-Splines, though, which are built upon B-Patch theories.

However, in [Sei89] Seidel *does* mention a condition (for $\mathbb{R}^2 \to \mathbb{R}^t$ only!) for GC^1. It is states that if:

> F and G are B-Patches with common boundary $[t_0, t_1]$ and with knots $t_{0,0}, \ldots, t_{0,n}, t_{1,0}, \ldots, t_{1,n}$ which all lie on line L [...],

[13] For example, I tried to model a sphere by setting up four patches forming a tetrahedron. This tetrahedron can not be projected on a Cartesian plane which is periodic enough to work (try for yourself, I would be interested if you succeed: `kero@dds.nl`). Perhaps it is possible to use the surface of a sphere for it (not very satisfying, using a sphere to model a sphere...)? The fact that my package does not process this, does not mean it does not work in general...

is used as precondition, then the same holds as in section 2.3.2.

This seems useful for the designer again, but the preliminary condition that $2n$ knots should all lie on the same line makes this condition useless. GC^q continuity is not even possible for the other edges of such a patch anymore, because if $t_{1,0}, \ldots, t_{1,n}$ and $t_{2,0}, \ldots, t_{2,n}$ are on one line to guarantee continuity for the second edge, the three clouds do not form a knot arrangement! The only way around this problem seems to be to put the knots of one cloud not only on the same line but also on the same points, but this refers the designer back to chapter 2. This also applies to the one dimensional parameter space.

Remark 3.7 (Boundary Curve) *The condition that both clouds are on one line, effectively brings the edge of the B-Patch to a known area: the restriction of the patch to the edge, is (like in the analogue for Bézier, remark 2.8) a B-Patch of one dimension (parameter space) lower.*

These can be used by two adjacent B-Patches, using additional constraints on the placement of the control points to connect GC^1.

As said, only for one edge per B-Patch, thus the Quest remains unsolved for B-Patches.

3.5 Conclusion

In comparison with Bézier patches, B-Patches do not provide many new and useful features. Due to the lack of continuity requirements, not much of a smooth surface is left for any designer, unless the special case of the Bézier patches itself is used.

As in the conclusion of [Sei89], I would say "further research is necessary".

4 B-Splines

Both Bézier patches and B-Patches yield problems when it comes to smoothness. For Bézier patches there is a simple solution in one dimensional parameter space, but for higher dimensions solutions become cumbersome. B-Patches are worse, because their continuity does not extend itself further than one edge of a patch.

A solution can be searched for in several directions. Hans-Peter Seidel came up with a new basis [DMS90], made up from blended functions which satisfy the continuity and smoothness requirements by definition. Reasonably smooth piecewise polynomials are used, which can be glued together to construct patches. These patches can be connected without any losses of smoothness.

This chapter will treat single B-Splines, chapter 5 will treat a set of them as a basis for a single patch. The connection of several patches to a surface will be done in chapter 6.

Definition 4.1 (Multivariate B-Spline) *Let $V = \{t_0, t_1, \ldots, t_m\}$ be a finite multi-set*[14] *of points in $\mathbb{R}^s (m \geq s)$. The multivariate B-Spline $M(u \mid V)$ is defined recursively as follows:*

$$M(u \mid t_0, t_1, \ldots, t_s) = \frac{\chi_{[t_0,t_1,\ldots t_s)}}{|\mathrm{d}(t_0, t_1, \ldots, t_s)|},$$

where $\chi_{[t_0,t_1,\ldots t_s)}$ is the characteristic function over the half open convex hull $[t_0, t_1, \ldots t_s)$ and d() *(as defined on page 9) unequal to zero, which means $\{t_0, t_1, \ldots, t_s\}$ is an affinely independent set of points (if the set is not affinely independent, then $M(u \mid V) = 0$). For $V = \{t_0, t_1, \ldots, t_m\}, m > s$:*

$$M(u \mid V) = \sum_{d=0}^{s} \lambda_{i_d}(u) M(u \mid t_0, \ldots, t_{i_{d-1}}, t_{i_{d+1}}, \ldots, t_m),$$

where $W = \{t_{i_0}, t_{i_1}, \ldots, t_{i_s}\}$ is any *subset of affinely independent points in V and $u = \sum_{d=0}^{s} \lambda_{i_d}(u) t_{i_d}$ is the representation of u in barycentric coordinates w.r.t. W (if such a W does not exist, $M(u \mid V) = 0$).*

Quite a definition. Let's have a thorough look at it. We start with the shape of the B-Splines in section 4.1, followed by explanations of some technical details in the definition. The last section (4.5) is about the computation of B-Splines on computers.

4.1 What do B-Splines look like?

The recurrence starts with the characteristic function over a (half open) simplex. This function has a value of one inside the half open simplex and zero everywhere else, so it is a discontinuous zeroth degree piecewise polynomial. For every step in the recursion, the degree is raised by one. So $m+1$ points in s dimensions give an $(m+1) - (s+1)$ (because the simplex uses $s+1$ points) $= m - s$ degree piecewise polynomial. Some examples in figure 12 show this.

[14] The t_i need not necessarily be different.

Note that there is always precisely one maximum in the B-Spline, which is roughly at the median of the points.

Furthermore, the characteristic function as well as the barycentric coordinates are functions of $\mathbb{R}^s \to \mathbb{R}$ (there will be no $\mathbb{R}^t$ involved until chapter 5).

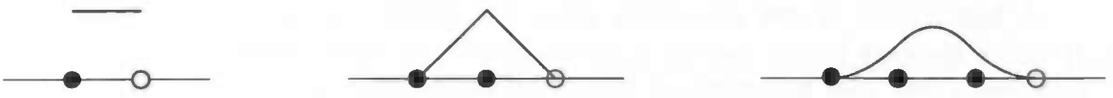

Figure 12: A zeroth, first and second degree B-Spline $\mathbb{R}^1 \to \mathbb{R}^1$

As can be seen in the pictures also, the first degree B-Spline is C^0, the second degree B-Spline is C^1 (C^q is a little stronger then GC^q. C^q will be used in parameter space, GC^Q in design space). This is not merely a coincidence, as follows from the next theorem:

Theorem 4.2 (Continuity of B-Splines) *If at most k knots of a B-Spline are collinear, then that B-Spline is* C^{m-k-1}.

where "collinear" is dimension-dependent: "coincide" for $s = 1$, "collinear" for $s = 2$, "coplanar" for $s = 3$ and so forth. Thus when at most s points in a set are collinear (which is the minimum required), then the B-Spline is C^{m-s-1}, where $m - s$ is the degree of the B-Spline. Thus an n^{th} degree B-Spline is C^{n-1} in that case.

4.2 Half open convex hulls

Due to the half open simplices started with, all convex hulls stitched together are half open. Thus at the edges where they are glued together in the recursive construction, only one half-open convex hull is counted, just as anywhere else. Note that it does not matter which half of the simplices is open and which closed, as long as its the same throughout the recursion. The place where the convex hulls are to be glued have the same function value, just as long as only one is counted, everything is OK. Also note that at the face of the spline, its value is zero, whether the face is closed or open (only in severely degenerate cases ($k = m$ in theorem 4.2) a discrepancy can appear).

4.3 ... *any* subset...

The proof of the fact that any simplex W which is not collinear can be used is too big to show here. To feel intuitively why we may choose any such W: the affinely independent set W determines the barycentric coordinates (if W were not affinely independent, the barycentric coordinates are not uniquely determined); if W is changed, so are the barycentric coordinates, which thus leaves the B-Spline resulting from recurrence 4.1 the same.

4.4 How many pieces are there?

A B-Spline is built up from barycentric coordinates, which are polynomials, and characteristic functions, which are compactly supported polynomials. The result is a **piecewise polynomial**, that is, a function consisting of different polynomials for different parts of the domain.

A first glance in one dimension gives reasonable amounts of pieces: A first degree B-Spline consists of two pieces, a second degree B-Spline contains 3 pieces, but in the recurrence four may be evaluated (the middle piece twice).

However, in two dimensions the numbers grow rapidly. A first degree B-Spline has four pieces, a second degree B-Spline eleven and a third degree B-Spline up to twenty-five![15]

4.5 Computing B-Splines

I do not know about a simple data structure to store piecewise polynomials in a computer.

That is, of course separate polynomials with boundaries can be used, but that does not feel coherent to me: the continuity (be it only C^0) is not clear in any way. On the other hand, it is possible to pre-compute the polynomials for all pieces of a B-Spline (or even better, just compute a polynomial when you need it and reuse that polynomial if you need it again).

Of course it is possible to evaluate the B-Splines according to recurrence 4.1 (like I did in my package), but that is slow, in the order of $O(e^{|V|})$.

[15] Fourth degree has up to **fifty**. The "up to" is there, because less is possible. Degenerate cases ($s+1$ or more collinear points) have less pieces, but also non-degenerate cases can have less. The maximum is reached for instance when all points are roughly spread on a circle for $s = 2$ (sphere for $s = 3$): no degeneracy and as many lines/planes in between points cross each other as possible, but no more than two lines cross at the same point.

5 B-Spline Basis

The B-Splines from chapter 4 can be used to construct a basis for piecewise polynomials in a way similar to Bernstein polynomials and B-Weights. The result is not a polynomial, but a piecewise polynomial with compact support. This way, there is no need to patch the piecewise polynomial, thus no continuity or smoothness is lost like in the case of Bézier and B-Patches.

This chapter treats the construction of one basis for one piecewise polynomial. Combining multiples of them will be done in chapter 6.

First, a few definitions are given, then the basis is constructed and investigated.

Definition 5.1 (Interior) *Let $\mathcal{A}$ be a knot arrangement. Then the interior of $\mathcal{A}$ is:*

$$\Omega_n = \bigcap_{|\beta| \leq n} \triangle_\beta \tag{10}$$

where $\triangle_\beta = [t_{0,\beta_0}, t_{1,\beta_1}, \ldots, t_{s,\beta_s}]$.

Definition 5.2 (Knot Net) *A knot arrangement $\mathcal{K}$ is called a knot net iff the interior Ω_n over this knot arrangement is non-empty.*

The requirement that the interior Ω_n is nonempty, means that the clouds assigned to the vertices of the simplex $\triangle$ are non-overlapping in a special sense[16]. See figure 13 for an example of a knot net.

Definition 5.3 (Normalized B-Spline) *Let $\mathcal{K}$ be a knot net assigned to the simplex $\triangle = [t_{0,0}, t_{1,0}, \ldots, t_{s,0}]$. For $\beta \in \Gamma_{s+1,n}$ define the sets $V_\beta^\triangle$ and the normalized B-Splines $N_\beta^\triangle(u)$:*

$$V_\beta^\triangle = \{t_{0,0}, \ldots, t_{0,\beta_0}, t_{1,0}, \ldots, t_{1,\beta_1}, \ldots, t_{s,0}, \ldots, t_{s,\beta_s}\}, \tag{11}$$

$$N_\beta^\triangle(u) = |\mathrm{d}(t_{0,\beta_0}, t_{1,\beta_1}, \ldots, t_{s,\beta_s})| \cdot M(u \mid V_\beta^\triangle). \tag{12}$$

These latter functions are called the normalized B-Splines over $\triangle$.

There are exactly $\binom{n+(s+1)-1}{n}$ of those normalized B-Splines $N_\beta^\triangle$. Again, this is precisely the amount that is needed to be able to define a base for polynomials of degree n, just as in (2.5) and (3.3). Also, these normalized B-Splines are linearly independent[17], so they are a candidate for a basis of polynomials of degree n. But: these B-Splines are *piecewise* polynomials and only have finite support: they do not extend themselves very far, instead they become zero outside the convex hull of the combined clouds. That is not enough to define a representation of arbitrary polynomials yet, so that will follow in chapter 6.

[16] There is no proper English word for this. Looking from a certain knot $t_{i,j}$, some parts of the clouds assigned to other vertices may not "hide" behind another cloud (the higher the second index, the more freedom for the knot).

[17] The maxima as briefly mentioned in section 4.1 are always in different parts of the simplex.

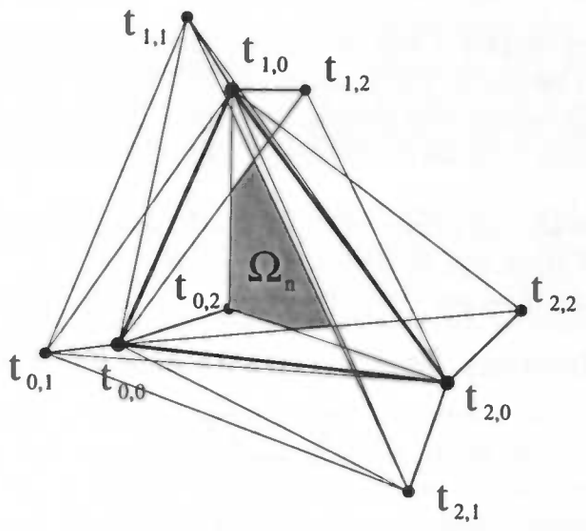

Figure 13: An example of a knot net. The interior is shaded

However, though the compact supports limit the possibilities, the fact that B-Splines are *piecewise* polynomials enhances the possibilities: these normalized B-Splines can be used as basis for piecewise polynomials adhering to some requirements:

$$F(u) = \sum_{i=0}^{s} N_\beta^\triangle(u) c_\beta,$$

where F is any piecewise polynomial with the same piece structure as dictated by the knot net-control points combination and the same support[18].

Again, the $c_\beta \in \mathbb{R}^t$ are called **control points**, which form the **control net**. Seidel doesn't provide any new name for the combination $\sum_{i=0}^{s} N_\beta^\triangle(u) c_\beta$, so I'll have to baptize it myself. Let's call such a combination an **S-Spline**, where the 'S' stands for Seidel (or super, special or superimposed, whatever you like).

The maxima of the normalized B-Splines in the basis are near the median[19] of the B-Spline, which is not exactly the same as for Bernstein polynomials, but close enough. So I'll keep using the association and adjacency relation of figure 4.

5.1 Looks of an S-Spline

Now how does an S-Spline look? Before we turn to complicated S-Splines, let's start with the simplest possible S-Splines and explain the pictures in detail.

[18] This is not a good description, but I know nothing better and I found nothing in the articles by Seidel.

[19] about as many t_i on every side of the maximum, so there are equally many pieces to connect the B-Spline smoothly with the axis where the compact support ends.

The simplest S-Splines are first degree S-Splines from $\mathbb{R}^1$ to $\mathbb{R}^2$. Two examples are drawn in figures 14 and 15.

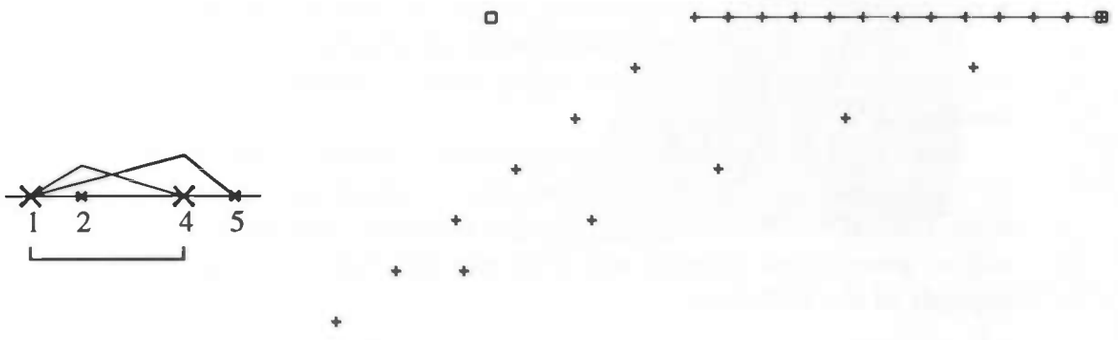

Figure 14: "leftmost" cloud is $\{1, 2\}$, the curve misses the corresponding control point

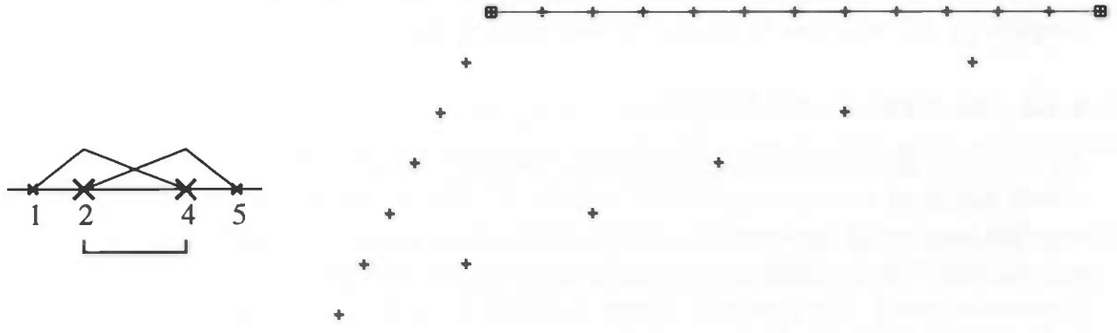

Figure 15: "leftmost" cloud is $\{2, 1\}$, the curve runs through the corresponding control point

Explanation of the picture

The little figure appearing next to the picture is an overview of the clouds and the triangulation in parameter space. These clouds are formed by a group of $\times$'s (the knots), where the biggest cross indicates t_{i0}, the smaller cross indicates t_{i1} (if there are more than two crosses: the smaller the cross, the higher the second index). Since clouds do not overlap, it should be clear which knots form one cloud.

For $s = 1$, that is a one dimensional parameter space, the triangulation is a set of intervals. These intervals are indicated below the axis and numbers: the little vertical lines show where one interval ends and another begins. For $s = 1$ the B-Splines which form the basis are also drawn. The vertical scaling is not

equal to the horizontal scaling. That is done, because the value of a B-Spline never exceeds the value one. Moreover, for this base the values of all B-Splines at one point in parameter space within the triangulation add up to one (except near the border of the triangulation, where the values drop to zero).

For $s = 2$ only the triangulation exists of triangles. It is drawn completely with the knots inside it. The B-Spline basis-functions do not fit in here and are omitted.

The big picture exists of a set of control points (□'s) and evaluated points (+'s). Some of these evaluated points are connected by a line: that line represents evaluation over the interior of the knot net. The lower left point is the origin: every single S-Spline will start and end there, because of the compact support of the B-Splines.

There is a difference between the two pictures: the second S-Spline runs through both control points, whereas the first does not. The cause of the difference of the two splines is in the configuration of the clouds. For the first spline, the "leftmost" cloud $\{t_{00}, t_{01}\}$ is $\{1, 2\}$, for the second spline that is $\{2, 1\}$ (in both cases the "rightmost" cloud is $\{4, 5\}$).

To state the difference in words: in figure 15, the cloud assigned to t_0 is *outside*[20] the simplex $[t_0, t_1]$, whereas in figure 14, the cloud is *inside*.

5.1.1 Second degree S-Splines

An example of a second degree S-Spline from $\mathbb{R}^1$ to $\mathbb{R}^2$ can be found in figure 16. Both clouds assigned to the simplex are outside it. The S-Spline is only drawn for evaluation over the interior of the knot net. The S-Spline does not run through any of its three control points. Another configuration of the knot net would not have changed that (except for the degenerate case). For a second degree S-Spline from $\mathbb{R}^2$ to $\mathbb{R}^3$ the same effect can be seen in figure 17, where the knot net is drawn as well.

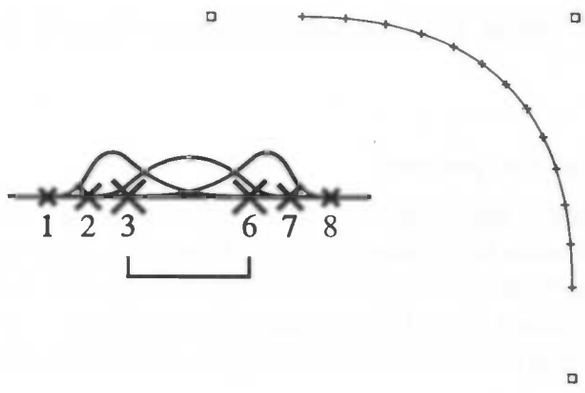

Figure 16: A second degree S-Spline from $\mathbb{R}^1$ to $\mathbb{R}^2$

[20] t_{00} can not be *outside* the simplex, because it is *on* the simplex. Nevertheless, I call this outside because the rest of the knots are outside. The same holds for inside.

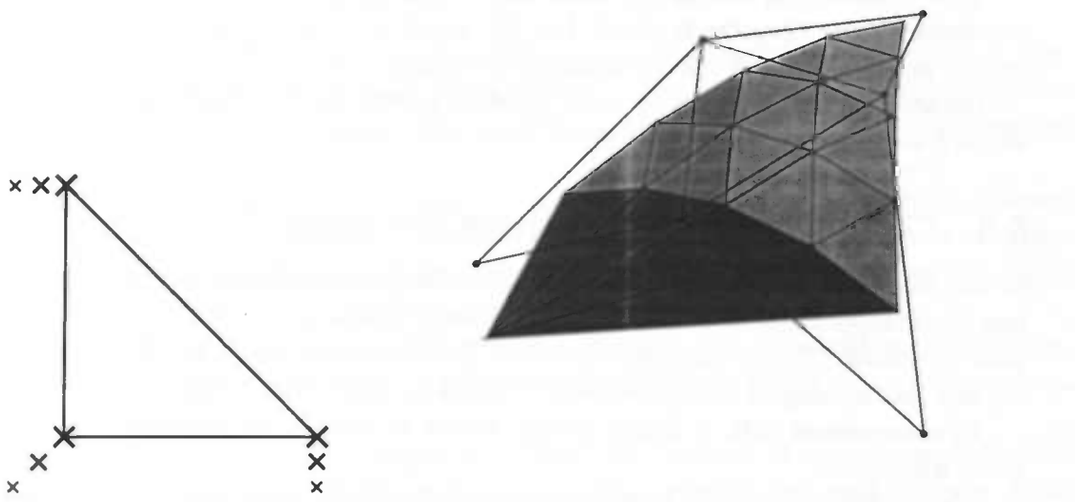

Figure 17: A second degree S-Spline from $\mathbb{R}^2$ to $\mathbb{R}^3$ with (a) its triangulation and clouds and (b) the resulting S-Spline.

5.1.2 Arbitrary degree

Higher degree S-Splines show no new insights. The S-Splines are even smoother and the distance to its control points is a little bigger. I will give some general rules on how these S-Splines behave (which does also hold for second degree):

- The less the clouds assigned to the vertices are spread, the sharper the B-Splines have to bend, the closer the result is to the corner control points and the more it looks like a Bézier patch.
- The more the clouds are spread, the more regular the curvature of the resulting S-Spline.

If clouds are widely spread, the B-Splines which form the basis are made from more widely spread points. This will give more space to a piece of the B-Spline to adjust itself to meet the next piece with proper smoothness. Thus the higher derivatives need not be varying as much.

5.2 Equivalence between B-Weights and B-Splines

Theorem 5.4 (Normalized B-Weights and B-Splines) *Let $\mathcal{K}$ be a knot net and let $\mathcal{A}$ be the knot arrangement obtained by dropping all $t_{i,n}$ from $\mathcal{K}$. Then*

$$N_\beta^\triangle(u) = B_\beta^{\mathcal{A}}(u), \ \forall u \in \Omega_n, \beta \in \Gamma_{s+1,n}. \tag{13}$$

Note that both the normalized B-Weights and the normalized B-Splines are functions from $\mathbb{R}^s$ to $\mathbb{R}$.

This definitely is one of the most important conclusions made by Dahmen, Micchelli and Seidel. The theorem is short, but the proof is long. It does not help us any further, so it is omitted here; it can be found in [DMS90].

Because of the equivalence, it is possible to use the De Boor algorithm for S-Splines, which I will do to solve the Quest.

5.3 Computing B-Spline space elements

In the B-Spline space a second degree piecewise polynomial has up to 11 pieces in 2D. As can be counted in the knot net of figure 13, an S-Spline constructed on that knot net would have 40 pieces. The amount can rise to (# B-Splines)$*$(# pieces in 1 B-Spline)$= 6*11 = 66$ for any second degree B-Spline space element in three dimensions.

In comparison with a Bézier patch, which is built from only one polynomial, this is quite a lot.

Also, considering this, I am not sure what will be more efficient: evaluating a point on the surface recursively, or precomputing the polynomials for all pieces and evaluating them when needed. A memoizing technique might advance the latter method to something useful, because some pieces are *very* small and other pieces (the interior, for example) are big.

6 Combining S-Splines

Like with Bézier and B-Patches, it is possible to combine S-Splines to create a (hyper)surface. This combining is nothing but addition. Thus geometric continuity remains the same as for the separate S-Splines. This provides a scheme which satisfies requirements about smoothness.

There are, however, some restrictions on the addition. For this, a slightly different version of a triangulation as definition 3.6 will be given, because where S-Splines are combined, they overlap for a part and will be added there, while for Bézier patches the sharp border did not have any need for addition like that.

For this, a triangulation is set up, like for the B-Patches, but with half-open simplices. Then each simplex in the triangulation will get its own S-Spline, which can cover any area together.

Again, let's start with some definitions. Then the method of combining S-Splines is given and examined. Examples from first and second degree are used to show what happens.

Definition 6.1 (Triangulation) *A set of vertices $V = \{t_0, t_1, \ldots, t_m\} \in \mathbb{R}^s$ has a triangulation $\mathcal{T}$ iff there is a set of simplices $\{\triangle \mid \triangle = [t_{i_0}, t_{i_1}, \ldots, t_{i_s})$, all $t_{i_d} (0 \leq i_d \leq m)$ are affinely independent$\}$, for which $\bigcup_{\triangle \in \mathcal{T}} = [V)$ and $\triangle_1 \bigcap \triangle_2 = \emptyset$ and two simplices from that set touch nowhere or have a touching face (not a part of a face).*

The difference with the definition 3.6 lies in the half-openness of the simplices. This should be the same half-openness as used in the definition of B-Splines 4.1. Only then the additions can be performed correctly.

To each of the vertices in V a cloud can be assigned, so that for every simplex $\triangle$ a knot net $\mathcal{K}$ is constructed. Then a set of $N_\beta^\triangle$ can be constructed. These sets are linearly independent. Now it is possible to use $\mathbb{R}^s$ as domain (but if $[V)$ is not bounded, the triangulation is infinite), so there is a complete spline space:

Definition 6.2 (B-Spline Representation) *Let $\mathcal{T}$ be a triangulation of a region in $\mathbb{R}^s$. Then every polynomial $F : \mathbb{R}^s \to \mathbb{R}^t$ can be represented as:*

$$F(u) = \sum_{\triangle \in \mathcal{T}} \sum_{\beta \in \Gamma_{s+1,n}} N_\beta^\triangle(u) c_\beta^\triangle.$$

Moreover, every degree n piecewise polynomial with the same piece-structure in the domain[21] can be represented as $\sum_\triangle \sum_\beta N_\beta^\triangle c_\beta^\triangle$.

Theorem 6.3 (Control Points for the B-Spline representation) *Let $\mathcal{T}$ be a triangulation as given in (6.1) of a region of $\mathbb{R}^s$. If for any two adjacent simplices $I, J \in \mathcal{T}$, whose touching face is $[t_0, \ldots, t_{i-1}, t_{i+1}, \ldots, t_s]$ the following holds:*

$$c_\beta^I = c_\beta^J, \ \forall |\beta| = n, \beta_i = 0, \tag{14}$$

[21] This is not a good description of the functions that can be represented, but like the functions represented on page 27, I do not know of any better definition.

Then

$$\sum_{\triangle \in \mathcal{T}} \sum_{|\beta|=n-1} N_\beta^\triangle c_\beta^\triangle = \sum_{\triangle \in \mathcal{T}} \sum_{|\beta|=n} N_\beta^\triangle c_\beta^\triangle, \tag{15}$$

where the control points relate as:

$$c_\beta^\triangle = \sum_{i=0}^{s} \lambda_{\beta,i}(u) c_{\beta+\hat{e}_i}^\triangle, \quad |\beta| = n-1. \tag{16}$$

The requirement (14) is simple: on the touching face of the two S-Splines, use the same control points.

However, recursive evaluation (16) looks the same as theorem 3.4! For the interior of the knot nets assigned to the simplices this is logical, according to theorem 5.4 the B-Splines and B-Weights are the same and so must be the resulting S-Splines and B-Patches. When evaluating *outside* any interior the important difference with B-Patches is that *multiple* S-Splines are evaluated together and added, due to the first summation $\sum_{\triangle \in \mathcal{T}}$.

The preliminary requirements (like one single cloud assigned to a vertex in parameter space) then guarantee that the combination remains inside the convex hull of the combined control points of the multiple S-Splines! This is not trivial and the proof is very complex, see [DMS90].

6.1 Looks of combined S-Splines

To see how combined S-Splines behave, first let's look at a very simple example: combining two first degree S-Splines from $\mathbb{R}^1$ to $\mathbb{R}^2$. This is done in section 6.1.1. Then some second degree S-Spline combinations are examined in section 6.1.2and 21

6.1.1 Combined S-Splines of First Degree

Exactly *how* is this combining done? Look at figure 18.

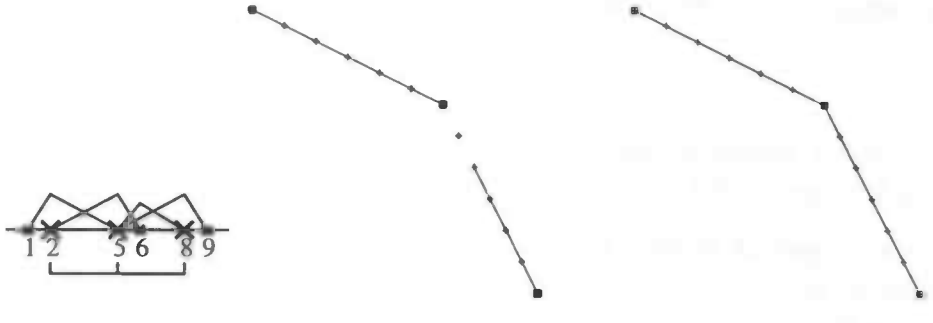

Figure 18: On the left: The parameter space with B-Splines for the picture in the middle. On the right: the combined S-Splines, drawn entirely with a line.

The connecting lines in the middle picture show the evaluation of values in the interior of both intervals ($\{2,5\}$ and $\{6,8\}$). This means that in parameter space from 2 to 8 there is one hole in the connecting line: from 5 to 6. As can be seen in the little left pictures, from 5 to 6 after addition is exactly *one* piece! That means that in the middle picture, the gap has to be filled with one first degree polynomial, matching in a GC^0 fashion. There is only one possibility: that has been added as connecting line to the two S-Splines in the right picture. The result is a GC^0 curve lying inside the convex hull of the control points.

The fact that there was only one piece in between the interiors is due to the use of the same clouds for one vertex as shared by two simplices. For S-Splines from $\mathbb{R}^1$ of degree n S-Splines, there will always be only one possibility for exactly $n-1$ pieces in between the interiors of two adjacent S-Splines to match in a GC^1 fashion.

The evaluation points of the combined S-Splines figure 18a are perfectly regularly spread over the S-Splines. However, the clouds are not equidistant. This is not a contradiction, because the vertices of the simplices are equidistant: looking at them shows the values 2, 5 and 8, which is indeed perfectly regular. This property holds for higher dimensional first degree S-Splines as well.

Remark 6.4 (Bézier equivalence for first degree S-Splines) *Combinations of first degree S-Splines for* any *value of s and t will be exactly the same as if Bézier patches were used for the same control points.*

This is the case because the combined S-Splines are nothing but straight (hyper)planes connecting adjacent control points.

6.1.2 Combined S-Splines of Second Degree

For first degree S-Splines, the two S-Splines in figure 18 had connected splines in them. They matched exactly GC^0 in the control point. Now take a look at two connected S-Splines of second degree. Both splines are first drawn separately in figure 19, where also some thin lines have been drawn in between some evaluated points (with the same parameter value, but in different S-Splines!) that are to be added. Then they are drawn added together in figure 20. Especially look at the smoothness and the locality of the result. The smoothness is GC^1, and the combined S-Splines remain inside the convex hull of the control points.

6.1.3 Combined S-Splines of arbitrary degree

For higher degree, for higher dimensions, all the above holds as well. Due to the restricted amounts of pieces in the resulting piecewise polynomial (which is a direct consequence of the use of the same clouds), the result is a smooth (hyper)surface which remains inside the convex hull of its control net. For any degree n used, the result can be GC^{n-1} as long as the clouds in parameter space are laid out well.

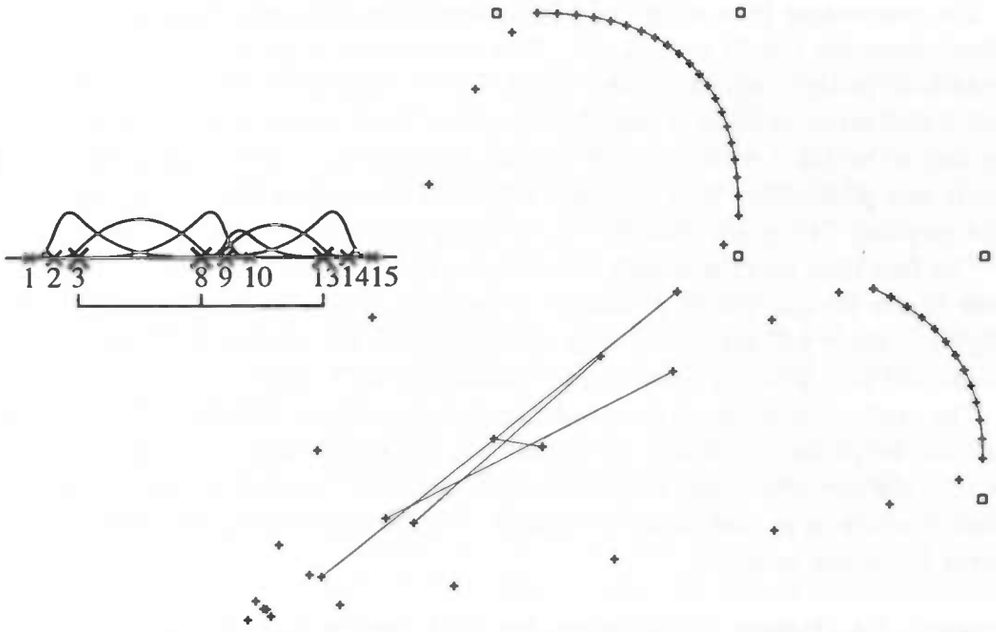

Figure 19: Two second degree S-Splines together in one picture...

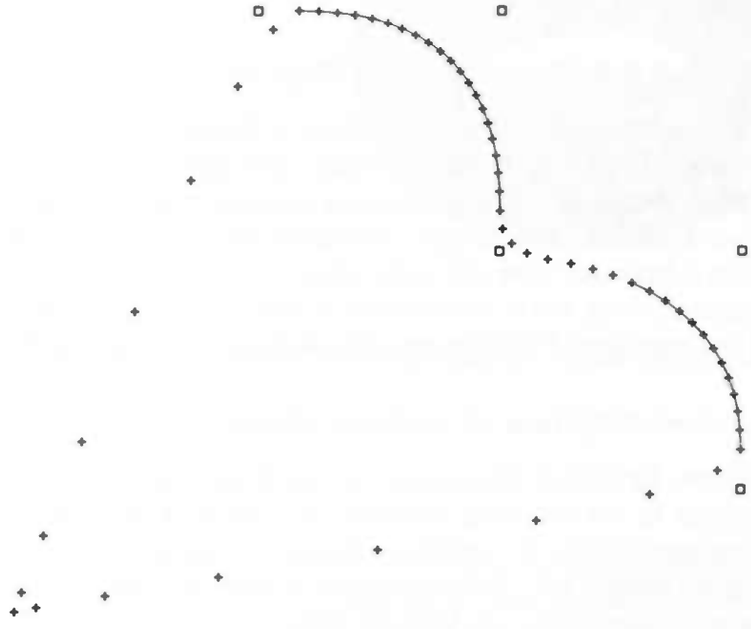

Figure 20: ...and the addition of the two

Figure 21: Two single second degree S-Splines (a) and (b) combined to form a (part of) a surface (c), which is still GC^1 and within the control net.

6.1.4 Using S-Splines as boundary surfaces

When a surface is used as a boundary, the triangulation in parameter space must support this, i.e. be of the same topology, like may be the case for B-Patches. See for example figure 22, where an approximation of a torus has been drawn, with the triangulation next to it. The triangulation is a part of $\mathbb{R}^2$, whereas the torus has its own characteristic shape: not at all the same topology! I have cheated a little: the fact that this triangulation is shaped like a rectangle, makes it easy to "bend" it and glue the upper side to the lower side, as well as the left side to the right side. Note that the edges as well as the clouds as well as the control points used have to be the same! The triangulation then has a torus-like shape itself. The fact that this can be done, though all calculations are still done in the plane is what I call **pseudo-periodic**.

There is another way to help to solve this problem. While a larger design may have a topology unequal to a plane, parts of it (if taken sufficiently small) are always topologically equal to planes. Thus by computing parts of the surface (single S-Splines, for example) and gluing them later, the entire surface can be computed as well. Evaluated points can only be added if the parameter value is "the same"! Also, the clouds used have to be "the same"! This is not always *easy*...

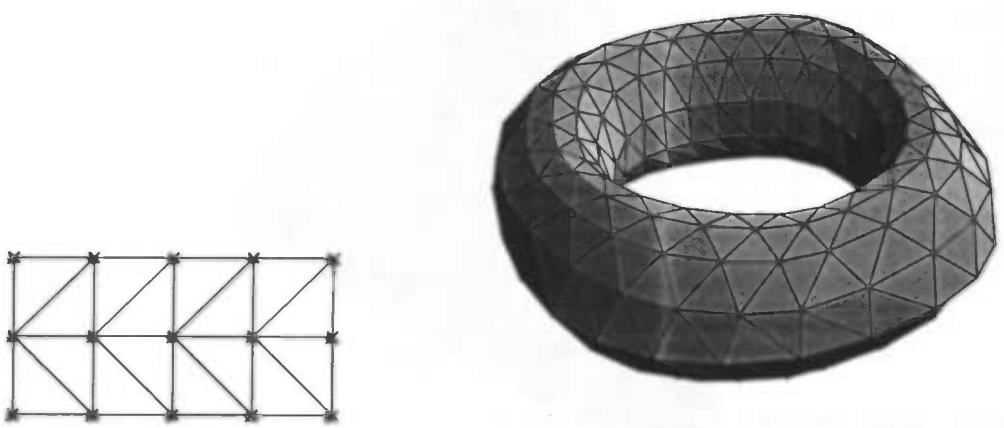

Figure 22: An approximation of a torus, built up from 16 second degree S-Splines: (a) the triangulation in $\mathbb{R}^s$ (b) the torus.

6.1.5 Computing S-Splines

When combining S-Splines, some parts of them overlap. An advanced data structure might optimize for this. In the overlap pieces have the same form, so memory (and computations) can be saved by constructing only one piece instead of multiple pieces. For an extra saving of the amount of pieces see remark 7.3.

The implementation I made simply evaluates all basis-functions at a certain parameter

value. No distinction whatsoever for separate or overlapping S-Splines. I said before it was slow!

6.2 Conclusion

The main advantage of combined S-Splines is the explicit Geometric Continuity: a (hyper)surface can be made GC^{n-1} if its degree is n if the clouds in the parameter space are not degenerate.

Disadvantages are merely computational: there are more polynomials needed to construct the surface and the restrictions on the parameter space complicate the computations.

7 The Quest solved for S-Splines

In this chapter the Quest for interpolating given points with normals is solved. For this, theories and tools from most previous chapters are used. The solution will be step by step:

1. Some special point on a single S-Spline is found with very regular behavior.
2. That special point has a regularly behaving tangent plane as well!
3. Instead of using an S-Spline and finding that special point, the process is reversed: the special point is given and some S-Spline has to be determined which indeed has that special point in it.

For first degree S-Splines there are plenty of such points, but they are not interesting: if interpolating with straight lines or planes is required, Bézier will do fine (or even simpler schemes).

For second degree, a solution will be found. A theorem will be stated and proven about the special point. Because second degree S-Splines are used, the resulting surface will be GC^1 automatically, thus fulfilling smoothness requirements.

Since the special point is in fact the point the designer gave, along with its tangent line/(hyper)plane, it will be called **user-defined point** from now on.

7.1 A special point on an S-Spline

The good observer will see that the endpoints drawn in figure 16 lie *on* the line through two of the three control points. Also, the tangent plane at that point is the same as that line (OK, I admit it was easier to see that when my package produced a series of numbers instead of pictures like this). For this simple case in two dimensions I will show that it actually does, then I will prove it for arbitrary dimensions.

7.1.1 Two dimensional endpoints

Take t_0 to be the endpoint of consideration and let t_1 have a higher value. Then for the cloud assigned to t_0 the additional points are outside the segment between t_0 and t_1, so both t_{01} and t_{02} have a lower value. Now it is allowed to evaluate in t_0 as if it were a B-Patch because of theorem 5.4.

So compute c_{00}^2 in t_0 using De Boor (3.4): $c_{00}^2 = \lambda_{00,0}(t_0)c_{10}^1 + \lambda_{00,1}(t_0)c_{01}^1$. Now $\lambda_{00,0}(t_0) = 1$ and $\lambda_{00,1}(t_0) = 0$, so $c_{00}^2 = c_{10}^1$ which is in turn $\lambda_{10,0}(t_0)c_{20}^0 + \lambda_{10,1}(t_0)c_{11}^0$. Clearly, that point lies on the line between c_{20} and c_{11}.

For the tangent line through c_{00}^2 in t_0 again De Boor is used. It states in corollary 3.5 that it is spanned by the points $c_{\hat{e}_i}^{n-1}$ (evaluated in the proper parameter value), which are c_{10}^1 and c_{01}^1 in this two dimensional case. These can be evaluated for t_0. The first point is actually the point computed for c_{00}^2 in t_0. The second point evaluates to $\lambda_{01,0}(t_0)c_{11}^0 + \lambda_{01,1}(t_0)c_{02}^0$, with the first barycentric coordinate equal to one and the second equal to zero. So the second point is c_{11}. This confirms the claim.

7.1.2 Arbitrary dimensional endpoints

The computations from the previous section can be generalized to second degree S-Splines of arbitrary dimensions. The complete theorem is given and then proven.

Theorem 7.1 (Endpoint and its Tangent Plane) *Let $t_i = t_{i0}$ be on a face of the interior ($\partial\Omega_n$) of the knot net $\mathcal{K}$. Then c_0^2 evaluated for t_i lies in the plane spanned by all $c_{\hat{e}_i+\hat{e}_j}, 0 \leq j \leq s$, which is its tangent plane as well.*

These $s+1$ control points may not be collinear.

Proof: *Since t_i is on the edge of the interior, theorem 5.4 tells us it can be evaluated as an S-Spline, but also as if it were part of a B-Patch. The latter is done now, in particular the De Boor algorithm is used twice.*

Evaluate c_0^2 at t_i: $c_0^2 = \sum_j \lambda_{0,j}(t_i)c_{\hat{e}_j}^1$, where it is very clear that $\lambda_{0,i}(t_i) = 1$ and $\lambda_{0,j}(t_i) = 0, \forall j \neq i$. So the evaluation reduces to $c_{\hat{e}_i}^1$ at t_i: $c_{\hat{e}_i}^1 = \sum_j \lambda_{\hat{e}_i,j}(t_i)c_{\hat{e}_i+\hat{e}_j}^0$. This is a point on the (hyper)plane through the control points $c_{2*\hat{e}_i}$ and $c_{\hat{e}_0+\hat{e}_j}, \ldots, c_{\hat{e}_{i-1}+\hat{e}_j}, c_{\hat{e}_{i+1}+\hat{e}_j}, \ldots, c_{\hat{e}_s+\hat{e}_j}$ (the control points adjacent to $c_{2*\hat{e}_i}$).

As for the tangent plane, this is equal to the (hyper)plane spanned by all $c_{\hat{e}_j}^1$ evaluated at t_i. First, evaluate for $j \neq i$: $c_{\hat{e}_j}^1 = \sum_k \lambda_{\hat{e}_j,k}(t_i)c_{\hat{e}_j+\hat{e}_k}^0$ gives all $c_{\hat{e}_j+\hat{e}_i}, j \neq i$, because only for $k = i$ the barycentric coordinate is one instead of zero. As for the last point: $c_{\hat{e}_i}^1$ is equal to the point which we are determining the tangent plane for, which lies in the searched plane for sure. Also, it is affinely independent from all $c_{\hat{e}_j+\hat{e}_i}, j \neq i$, which completes the proof.

Note that for arbitrary dimensions, great care has to be taken, so that the cloud assigned to t_i does not prevent t_i from being on the edge of the interior! For two dimensions, this will be explained in section 7.2.

7.1.3 Arbitrary Degree S-Splines

Like second degree S-Splines, third, fourth and so forth degree S-Splines do not run through any control point in general. But I expect some kind of trick like 7.1 for the second degree case can be performed for third and higher degree S-Splines. The positioning of the point will not be perfectly in the plane spanned by (some) control points as for the second degree case, but will tend to be further away when the degree rises. The computations will become tedious, but I do not think they become impossible.

More research can be done in that direction, for which I suggest higher derivatives are taken into account for smoothness requirements, because if only the first derivative is considered, the solution is already given for second degree. I can imagine the second (and later higher) derivative should be as regular as possible, but this will not be trivial.

Any S-Spline of degree n matches its neighbors in a GC^{n-1} fashion. Research may try to get as many derivatives involved as possible (up to $n-2$?), though with current requirements by designers, I do not think it will be necessary to look any further than second derivatives.

7.2 Reversion of the Process

Theorem 7.1 can be invoked for the Quest. The trick is that *none* of the control points of the S-Splines is a user-defined point. The user-defined points *will* have one control point nearby, which is associated with a corner vertex of a simplex. All other control points will be used to connect the S-Splines properly.

More formally: Find a second degree S-Spline such that its $c^1_{\hat{e}_i}$ is the user-defined point, with given normal.

Note that the user gives some points, through which the curve should run with a certain normal. The control points and the triangulation plus clouds for the S-Splines should then be computed from that. As made clear in the previous section, the tangent plane is equal to the plane through some control points: for one user-defined point with normal, a part of only one S-Spline is needed, that is to say, $s+1$ control points of the $\binom{s+2}{2}$ that construct one S-Spline.

Let t_i be associated with $c_{2*\hat{e}_i}$, then the exact requirements are:

1. t_i lies on the interior of the knot net assigned to its simplex,
2. $\sum_{j=0}^{s} \lambda_{\hat{e}_i}(t_i) c_{\hat{e}_i+\hat{e}_j}$ is equal to the user-defined point,
3. All $c_{\hat{e}_i+\hat{e}_j}$ in the previous requirement lie in the user-defined tangent plane.

There is a *lot* of freedom left for the choices of the parameter values and the control points, however.

To start with requirement 1, in two dimensional parameter space there is plenty of choice. See figure 23.

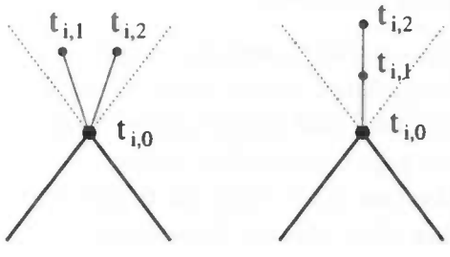

Figure 23: Some positionings for knots in a cloud

The thick lines are edges of a simplex, the dotted lines are extended from these edges. First, it is required that t_{i1} and t_{i2} are inside the angle bounded by the dotted lines. Then the lines from $t_{i,1}$ and $t_{i,1}$ to $t_{j,0}$ do not restrict $t_{i,0}$ from being inside the interior. The clouds in the other vertices of the simplex must be small enough, so that lines from t_{i1} to t_{j1} for any j do not exclude $t_{i,0}$ from the interior, either. For two "regular" solutions for

$s = 2$, see figure 23. There do not rise new problems when $s = 3$ or higher. Keeping the clouds relatively small will help very much to satisfy this last requirement.

The last important thing for the configuration of the clouds is to let t_{i1} and t_{i2} be close enough so that they do not extend themselves to far into the next simplex, where they might diminish (the area of) the interior of that simplex to zero. (When problems might occur due to this, probably it is a better idea to change the triangulation.)

Then there are requirements 2 and 3. These can be satisfied at the same time, namely at the moment the s control points are set. These need to be set in the tangent plane (requirement 3 fulfilled), such that requirement 2 is fulfilled. Since the displacement of the control point $c_{\hat{e}_i}$ w.r.t. the user-defined point is caused by $\lambda_{\hat{e}_i}$ only, it suffices to take that barycentric coordinate into account. Because we started with the layout of the triangulation and the clouds, $\lambda_{\hat{e}_i}$ is set and the relative layout of the $s + 1$ control points w.r.t. the user-defined point is given.

There still remain lots of freedom for the absolute distances between those control points, as well as for the orientation of the set of control points in the tangent plane.

Then there are plenty of control points associated with a simplex left without a placement (those c_β with $\beta_i = 0$). In fact, there is no restriction at all on their placement. But of course some of them might as well be used by adjacent simplices to run through another user-defined point.

To be able to fulfill all requirements, at least as many simplices in the triangulation are needed as there are user-defined points[22]. There still is a lot of freedom left and I suggest to choose the sub-simplices of a simplex in such a way that they have about equal size and that adjacent control points have regular distances.

7.2.1 A few examples

Let's examine some of the freedom. First, take a look at the freedom in parameter space. Change the placement of the clouds and see what happens, see figure 24. It might not be very clear, but the shape of the second picture differs from the first. For that reason, both pictures are joined in figure 25

The consequence of freedom: another choice of parameter spacing (in this case with the same control points) gives another surface. The rightmost half of figure 24b has a bigger interval in parameter space, of which the immediate result is relatively smaller cloud for the "leftmost" control point: (5-6-11) instead of (5-6-8). This influences the barycentric coordinates λ_{20} of the "rightmost" S-Spline and thus the "rightmost" part of the curve. (The "rightmost" cloud has the same relative size, because otherwise the user-defined endpoint would not be met.)

For freedom of choice w.r.t. control points, see figure 27 on page 45. First, the curve does remain inside the convex hull, so movement of control points will likely change the shape of the curve. Second, because the pairs of control points dictating the tangent line are closer to each other in figure 27b than in 27a, there is less space left to bend away, thus

[22] Not necessarily, if you want problems...

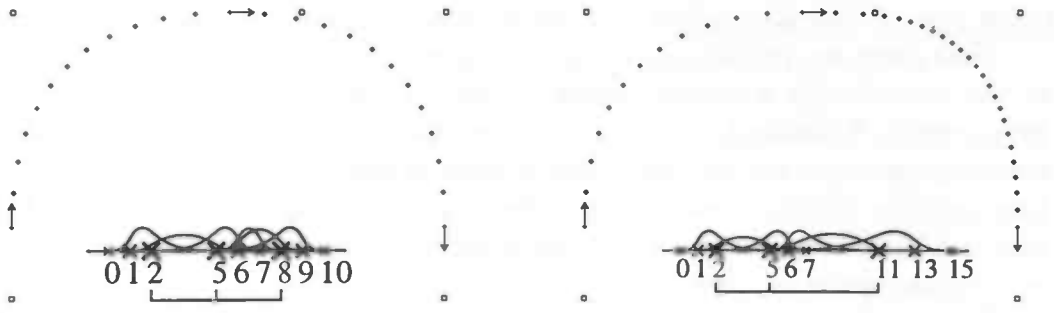

Figure 24: Two S-Splines, (a) with regular spacing in parameter space (2-5-8), (b) with irregular spacing (2-5-11), note that the shape differs!

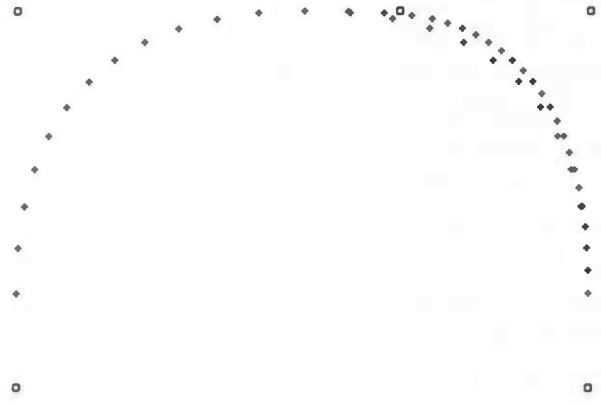

Figure 25: The two splines from figure 24 in one, showing clearly they are *not* the same.

leaving a sharper curvature.

7.2.2 A closed boundary

Let us not forget the requirement of the closed boundary. What should be clear from the above, is that it is possible to end up in a user-defined point with a given tangent plane. Then it is of course possible to end up there from two sides. See figure 26, where this approach has been taken to close a curve in the leftmost user-defined point.

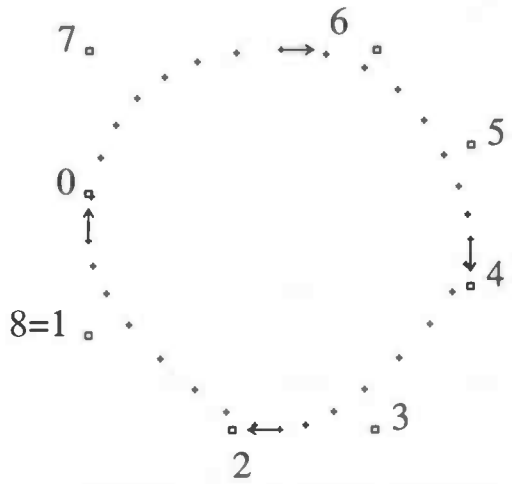

Figure 26: An example of closing a curve in two dimensions

The resulting curve is irregular while the user-defined points with normal were laid out regularly. The problem is that the lay-out of the control points is not regular: when starting with putting control points from 0 to 6, this can be done regularly as is done in figure 26. However, when approaching the leftmost user-defined point from above, the upper left control point 7 can *not* be put regularly, because it has to be on both the tangent line trough the leftmost user-defined point and the upper user-defined point. (One S-Spline (control points 6-7-8) reaches for two user-defined points here. That should have been avoided.) (The fact that control points 1 and 8 are the same is coincidence.)

Even worse, for higher dimensions, this does not work at all! The claim by Seidel (theorem 6.3) is that apart from the same clouds, the same *control points* should be used. That is not the case here, because the same user-defined points are used, for which different control points are needed. Furthermore, it is not known yet how the edge of an S-Spline restricted to $\triangle$ looks. The same solution used to connect two B-Patches (see section 3.4) can be used: put the clouds on one line and lay the control points like for the Bézier solution. If there is one specific boundary in the design which could be used for this (for example, where sharp edges are used instead of smooth connections), then this method can be used.

However, this can be done better. For this the same control points have to be used, along with the same clouds when a curve/surface reaches for a user-defined point from two sides (as required for theorem 6.3). If the clouds are the same, so is the vertex it was assigned

to. In that case for any pair of adjacent S-Splines, there has to be an associated pair of adjacent simplices in parameter space. Like for B-Patches, this means the triangulation has to have the same topology as the resulting surface. The advantage of this method is that the result is more regular, globally. See figure 27. For one dimensional parameter space, it is easy to be obtain the "correct" topology by cheating: instead of an infinite axis, use a part like $[0, n)$, which starts again at 0 when reaching n. For higher dimensions there can rise problems, see section 3.3.1 under triangulation and section 7.5.

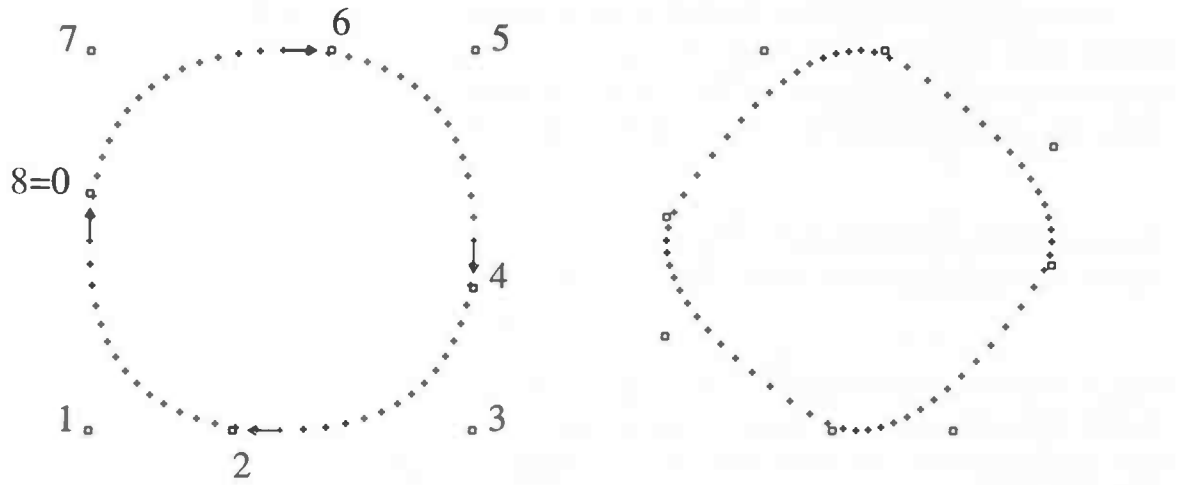

Figure 27: Two "circles", with (a) bigger and (b) smaller spacing between control points. The clouds in parameter space are unchanged.

7.2.3 Disadvantages

As can be seen in figure 27b, two resulting S-Splines do not run away from one user-defined point symmetrically in general (they do in figure 27a, see remark 7.4). See figure 28, where figure 27b and its mirror image are drawn together. For an irregular design, like a dashboard, this should not be a big problem. For regular objects, this is less desirable. To some extent, this can be improved by using small clouds.

7.3 Odds and Ends

This section is just a collection of weird habits of S-Splines, unexpected similarities and other funny things. The first theorem results in a reduction of pieces in S-Splines used for constructing surfaces, so is useful for implementors.

Theorem 7.2 (Indifference of Position of $t_{i,n}$) *When a complete triangulation for a surface is set up, the positioning of the s^{th} knot in any cloud does not matter to some extent.*

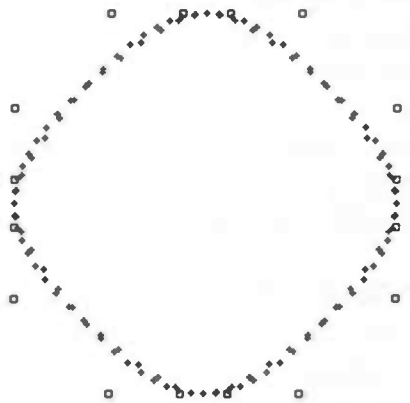

Figure 28: Figure 27b with its mirror image in one picture. They are not the same.

Proof: Let's start in one dimension. Consider vertices t_0, t_1, t_2, which support two simplices. Knot t_{12} only appears in N_{02}^{01} and N_{20}^{12}.

$$\begin{aligned} N_{02}^{01} &= |\mathrm{d}(t_{00}, t_{12})|\, M(u \mid \{t_{00}, t_{10}, t_{11}, t_{12}\}) \\ &= |\mathrm{d}(t_{00}, t_{12})| \left(\frac{\mathrm{d}(u, t_{12})}{\mathrm{d}(t_{00}, t_{12})} M(u \mid \{t_{10}, t_{11}, t_{12}\} + \frac{\mathrm{d}(t_{00}, u)}{\mathrm{d}(t_{00}, t_{12})} M(u \mid \{t_{00}, t_{10}, t_{11}\}) \right) \end{aligned}$$

Similarly,

$$\begin{aligned} N_{20}^{12} &= |\mathrm{d}(t_{12}, t_{20})|\, M(u \mid \{t_{10}, t_{11}, t_{12}, t_{20}\}) \\ &= |\mathrm{d}(t_{12}, t_{20})| \left(\frac{\mathrm{d}(u, t_{20})}{\mathrm{d}(t_{12}, t_{20})} M(u \mid \{t_{10}, t_{11}, t_{20}\} + \frac{\mathrm{d}(t_{12}, u)}{\mathrm{d}(t_{12}, t_{20})} M(u \mid \{t_{10}, t_{11}, t_{12}\}) \right) \end{aligned}$$

Now assume $\mathrm{d}(t_{00}, t_{12})$ and $\mathrm{d}(t_{12}, t_{20})$ have the same sign. For any u, the sum of these two basis-B-Splines then is:

$$\pm \left(\mathrm{d}(t_{00}, u) M(u \mid \{t_{00}, t_{10}, t_{11}\}) + \mathrm{d}(u, t_{20}) M(u \mid \{t_{10}, t_{11}, t_{20}\}) \right)$$

where t_{12} has disappeared completely.

This can be repeated for arbitrary dimensions, where the drop-out of point $t_{i,n}$ will always happen where two faces of simplices touch.

The phrase "to some extent" in the theorem is thus that the normalizing determinants have the same sign, so they cancel the denominator in the barycentric coordinates of the B-Splines. Effectively, this means the position of point $t_{i,n}$ does not matter, as long as it is inside the convex hull of all vertices adjacent to vertex t_i, which it is by definition of the knot nets of the S-Splines sharing t_i!

Remark 7.3 (Reduction of the Number of Pieces) *Using theorem 7.2, you can put any $t_{i,n}$ in the same place as $t_{i,0}$, $t_{i,1}$, ..., $t_{i,n-1}$, which reduces the amount of pieces with different polynomials greatly.*

For the Quest, this means that a single S-Spline has a maximum of 13 pieces, of which 12 are shared with neighbor S-Splines (only the interior is not shared). This brings the scheme much closer towards real time usage.

Remark 7.4 (Bézier Resemblance) *In one dimensional parameter space, when all the clouds are regularly spread and the control points are on the crossing of the two tangent lines through the user-defined points (like with the 2^{nd} degree Bézier solution), the* same curve *is produced as with the Bézier solution.*

For some unknown reason the parts of the curve where the S-Splines are glued sums up perfectly and become part of a Bézier patch. The B-Patch parts of the S-Splines are Bézier patches, too. See for a perfect example figure 24a.

Remark 7.5 (Triangulation of parameter space) *If a triangulation produces a surface, then problems occur when the surface represents a closed boundary of a 3D object. Because of the sharing of the clouds, the triangulation must have the same topology, that is, for each simplex in the triangulation, there must be adjacent simplices, which associate with adjacent S-Splines in the object space.*

For a torus, that is not a problem: take a rectangle in parameter space, make a cylinder of it and then curve it, so the shape of a torus is obtained.

For a sphere, nothing rectangular gives an easy solution. Perhaps that rectangle could give a solution for a part of an icosahedron, which could then be rotated twenty times, thus giving a sphere. It will probably be easier to use the surface of a sphere as parameter space in some way. The topology requirement can then be satisfied, but in what way is a simplex correct? Its edges are not really straight anymore...

8 Conclusion

For surfaces (globally 2D in a 3D space), Seidel's scheme is very good, since it provides many good features as mentioned in the introduction. Let us go step by step through the requirements in chapter 1:

1. **Deviation** S-Splines deviate more from their control points then Bézier patches. The bigger the clouds in parameter space used, the bigger the deviation in the design space. However, these clouds can be kept small enough in applications. Also, S-Spline surfaces remain inside the control net.

2. **Uniqueness** As soon as the clouds and control points have been determined, the definition of a surface built up from S-Splines is unique.

3. **Smoothness** By default, S-Spline surfaces are GC^{n-1} when n degree polynomials are used. The user should be able to define a sharp edge of a design instead of the smooth polynomials normally used. The CAGD-program can implement this easily by converging the clouds in parameter space to one point.

4. **Locality** Local changes are easy. Movement of a control point does not matter at all, addition of a control point just gives a couple of new simplices/S-Splines.

5. **Normal** With the method given in section 7.2, the user can now define points with gradients, where the S-Spline surface will run through in the correct direction.

Advantages

The biggest advantage of the S-Splines is the continuity. By just using this scheme, GC^{n-1} can be obtained by just using n^{th} degree S-Splines.

For the Bézier solution, the surface runs through the control points; to be precise: when a point with given normal is used, multiple Bézier patches reach for that point with proper smoothness. For the S-Spline scheme, only one S-Spline reaches out and the rest is automatically correct. This is an advantage of about a factor of six in the $s = 2$ case[23].

Another advantage is the implicit high amount of pieces. This offers flexibility to make an interpolating surface smooth through *any* combination of user-defined points, which is much more difficult for the Bézier scheme.

Disadvantages

When user-defined points are given, the user will not have the guarantee the surface runs through that point symmetrically. This can be a disadvantage.

[23] As a triangulation grows to infinity, the average amount of triangles that meet in a vertex approaches six. [Theorem by ???]

As well as an advantage, the amount of pieces in the S-Splines is a disadvantage. More pieces means more different polynomials means more memory and more complex computations.

Future Research

You can think of the following:

- Is there a way to assign clouds to vertices fully automatically? Given the triangulation, I think so, but it will not be easy, because the triangulation can be severely irregular. Also, taking a look at non-$\mathbb{R}^2$ parameter spaces (like sphere-coordinates) might be interesting.
- Is there a layout of the clouds which will create symmetric surfaces always?
- Is there a way to produce GC^2 surfaces through user-defined points? I think so, but I expect higher degree S-Splines to be necessary, accompanied by more complicated computations.

I wish you fun and luck solving these!

A Notation used in this thesis

β and other small Greek characters are multi-indices. Also, the 0 appears as multi-index with all indices equal to zero.

c is a control point in $\mathbb{R}^t$ (unfortunately, this t has nothing to do with the next item).

t_x is a knot in $\mathbb{R}^s$.

$\triangle$ A simplex in parameter space; built from three knots (t_i in the Bézier case, t_{i0} otherwise).

$\lambda_i(u)$ are barycentric coordinates of $u \in \mathbb{R}^s$ w.r.t. some simplex.

Mostly, only '+'s are found where the splines have been evaluated. Lines/curves running through these pluses have been added by hand later.

B Legend of the pictures

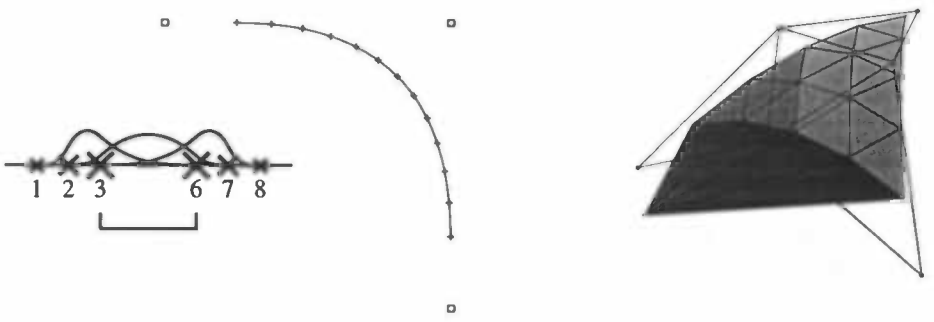

Figure 29: Three pictures: (a) the parameter space (b) the 2D results (c) the 3D results

B.1 Legend of the pictures in parameter space

— Either the square cup at the bottom (1D parameter space) or all the lines drawn (2D parameter space) represent the triangulation.

× A knot. The biggest knots have second index zero (t_{i0}); the smaller the cross, the higher the second index in (t_{ij}, j higher).

⌒ A B-Spline basis-function (1D parameter space only).

B.2 Legend of the S-Splines (2D)

□ The squares indicate the control points c_β. Note that the there is always one control point in every cloud that is shared by all simplices sharing that cloud (i.e. in these 2D pictures every c_{0n} is the same as the c_{n0} of the next S-Spline).

\+ The little plus signs are points on the spline. Usually there are enough of them in the picture to see what the curve looks like.

— A line indicates the part of a patch which is restricted to

- $\triangle$ in the case of B-Patches,
- the interior in the case of S-Splines.

These lines are added by hand, they are *not* not automatically generated.

B.3 Legend of the S-Splines (3D)

$\lhd$ Filled triangles represent a part of the surface; the vertices of these triangles are points on the surface itself.

— The knot net.

• A control point.

References

[Far86] Gerald Farin. *Triangular Bernstein-Bézier Patches.* Dep. of mathematics, University of Utah. *Computer Aided Geometry Design 3*, Elsevier Science Publishers BV, North-Holland, 1986.

[Sei88] Hans-Peter Seidel *A new Multiaffine Approach to B-Splines.* Wilhelm-Shickard-Institut für Informatik, Universität Tübingen. *Computer Aided Geometry Design 6*, Elsevier Science Publishers BV, North-Holland, 1988.

[Sei89] Hans-Peter Seidel. *Symmetric Recursive Algorithms for Surfaces: B-patches and the De Boor Algorithm for Polynomials over Triangles.* Universität Tübingen. *Constructive Approximation*, Springer-Verlag NY Inc, New York, 1989.

[DMS90] Wolfgang Dahmen, Charles A. Micchelli & Hans-Peter Seidel. *Blossoming Begets B-spline Bases Built By Better B-patches. Mathematics of Computation V59 no199*, 1990. *The main theories. Thorough.*

[Sei90] Hans-Peter Seidel. *Polar Forms and Triangular B-Spline Surfaces.* Universität Erlangen. *3D version of [DMS90], reads easier.*

[Sei92] Philip Fong & Hans-Peter Seidel. *An Implementation of Triangular B-Spline Surfaces over Arbitrary Triangulations.* University of Waterloo/Universität Erlangen. *Computer Aided Geometry Design 10*, Elsevier Science Publishers BV, North-Holland, 1992. *Despite its name, nothing about the implementation itself is revealed.*

[Far93] Gerald Farin. *Curves and Surfaces for CAGD. A practical guide. Computer science and scientific computing*, Academic Press, Inc, Boston, 1993. *Good book to start with.*

[Veg97] Gert Vegter. *Apolar Bilinear Forms and Multivariate B-Splines.* Dep. of Mathematics and Computing Science, Rijksuniversiteit Groningen. 1997. *Thorough.*

[HB94] Donald Hearn & M. Pauline Baker. *Computer Graphics* Prentice Hall, 1994, second edition *Wonderful overview of Computer Graphics*

[Str91] Bjarne Stroustrup. *The C++ Programming Language* Addison-Wesley 1991, second edition *Complete, since written by the inventor of C++*