COMENIUS UNIVERSITY, BRATISLAVA
FACULTY OF MATHEMATICS, PHYSICS
AND INFORMATICS

From segments to spline curves and patches

(User's manual)

Diploma thesis

2016 **Peter Bezák**

COMENIUS UNIVERSITY, BRATISLAVA
FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS

DEPARTMENT OF ALGEBRA, GEOMETRY
AND DIDACTICS OF MATHEMATICS

From segments to spline curves and patches
(User's manual)

Diploma thesis

Field of study:	1113 Mathematics
Study programme:	Computer graphics and geometry
Department:	Department of Algebra, Geometry and Didactics of Mathematics
Supervisor:	RNDr. Róbert Bohdal, PhD.
Consultant:	RNDr. Soňa Kudličková, CSc.

Bratislava 2016 **Peter Bezák**

UNIVERZITA KOMENSKÉHO
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

KATEDRA ALGEBRY, GEOMETRIE A
DIDAKTIKY MATEMATIKY

Od jednoduchých oblúkov k splajnovým krivkám a záplatám

(Používateľská príručka)

Diplomová práca

Študijný program:	1113 Matematika
Študijný odbor:	Počítačová grafika a geometria
Katedra:	Katedra algebry, geometrie a didaktiky matematiky
Vedúci:	RNDr. Róbert Bohdal, PhD.
Konzultant:	RNDr. Soňa Kudličková, CSc.

Bratislava 2016 **Peter Bezák**

Honour declaration

I declare that I personally prepared this presented diploma thesis and carried out myself activities directly involved with it. I also confirm that I have used no resources other than those declared.

Bratislava, 4.5.2016

....

Student’s signature

Acknowledgment

I would like to express my gratitude to my supervisor Róbert Bohdal for useful tips and advice during designing and developing software application *Splines & Surfaces,* and to my consultant, Soňa Kudličková, for helping me with mathematical description of geometrical objects and valuable comments on theoretical part of this work. Special thanks go to my beloved family and friends for their support.

24440370

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Bc. Peter Bezák
Study programme:	Computer Graphics and Geometry (Single degree study, master II. deg., full time form)
Field of Study:	Mathematics
Type of Thesis:	Diploma Thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: From segments to spline curves and patches (User's manual)

Aim: Implementation (and its user manual) for recent Microsoft Windows operating system

Comment: Vytvorenie programu a užívateľskej príručky pre aktuálny operačný systém Windows. Vyžaduje sa dobrá znalosť programovania.

Supervisor:	RNDr. Róbert Bohdal, PhD.
Consultant:	RNDr. Soňa Kudličková, CSc.
Department:	FMFI.KAGDM - Department of Algebra, Geometry and Didactics of Mathematics
Head of department:	prof. RNDr. Pavol Zlatoš, PhD.
Assigned:	27.11.2013
Approved:	20.03.2014

prof. RNDr. Július Korbaš, CSc.
Guarantor of Study Programme

....

Student Supervisor

Abstract

The purpose of this work is to create software application, which will be used during lectures in Representation of geometric objects. Thus main task for application called *Splines & Surfaces* is to enable user to enter input data and modify arcs, splines, patches and surfaces to accomplish this. We mathematically describe these objects, mention their properties and methods for enumeration. Then we describe how to modify their shape by modification of knot vector, multiplicity of control points or by choice of end condition. Work specifies requirements, goal, and controls for application and data model, input, principle of construction, methods for modifying, output and possible errors for each graphic object. User's guide for *Splines & Surfaces* application is supplied as a part of this work.

Keywords: curves, approximation and interpolation splines, surfaces, continuity conditions, software application, user's guide

Abstrakt

Hlavným účelom tejto práce je vytvorenie softwarovej aplikácie, ktorá bude využívaná na prednáškach predmetu Reprezentácia geometrických objektov. Preto je úlohou aplikácie *Splines & Surfaces* umožniť používateľovi zadávať vstupné údaje a modifikovať oblúky, splajny a plochy. Tieto geometrické objekty matematicky opíšeme a upozorníme na ich vlastnosti a metódy vyčíslenia. Ponúkneme viaceré možnosti modifikovania ich tvaru cez násobné riadiace vrcholy, zmeny v uzlovom vektore alebo rôzne okrajové podmienky. V práci sú uvedené požiadavky, cieľ a ovládanie aplikácie a špecifikácie dátového modelu, vstupu, princípu konštrukcie, možnosti modelovania, výstupu a mońých chýb pre každý grafický objekt. Súčasťou práce je používateľská príručka k aplikácií *Splines & Surfaces*.

Kľúčové slová: krivky, aproximačné a interpolačné splajny, plochy, podmienky spojitosti, softwarová aplikácia, používateľská prírućka

Contents

Table of pictures

Table of list and tables

Introduction

The presented work harmonises two aspects, mathematical representation of geometric objects and manipulation with those objects by means of software application *Splines & Surfaces*.

First chapter focuses on studying planar and spatial curves. It starts from simple arcs - Bézier and Hermite, continues through continuity conditions for splines and then presents representation of individual spline curves. This includes Hermite spline, cardinal spline, approximation and interpolation Bézier spline, Beta spline and B-spline.

To further improve control over shape of curve, rational curves are introduced. They allow shape modification by changing weights of control points and also represent conic sections.

Important de Casteljau and de Boor enumeration algorithms for Bézier curve, B-spline curves and surfaces are used in *Splines & Surfaces* to evaluate points on curve or surface.

Representation of curves and splines naturally leads to representation of surfaces. This includes parametric surfaces, surfaces of revolution and extruded surfaces – affine surfaces, surfaces created from boundary curves – Coons patches and surfaces defined by control net – both integral and rational Bézier and B-spline surfaces.

Essential part of this work is designed and developed software application *Splines & Surfaces*. This application in detail processes individual options for modification of geometric objects, which are described in first chapter and other literature concerning CAGD. With respect to scope of this work, not all modification options are listed in text, but they are implemented in software.

Next chapter states and specifies requirements, and defines how it looks and controls for developed software application. It describes functions for controlling the application and processing input form user and defines data model and specifications for each individual arcs, splines and surfaces – input, how is this input processed, output and possible errors. This includes all possible ways to modify given object.

In each case of curve’s visualisation, after defining a sequence of control points and specifying type of curve, curve and blending functions are rendered. Blending function define shape of curve and its geometric properties. In case of splines choice of end condition is possible – either phantom or multiple vertices, by defining vector first derivative at end points or changes in knot vector, which can be uniform, open or non-uniform.

Application offers variety of options to create a surface. Either creating parametric surface, surfaces that is created from parametrically described profile curve or boundary curves or surfaces defined by control net. Parameters for these curves and control net can be modified. Displayed result is model of surface – representation of biparametric geometric object.

Third chapter is about implementation of software application *Splines & Surfaces*. It describes important functions for processing input from user, displaying enumerated objects. These functions serve as interface between user and hardware. Important content of this chapter is class structure. All classes and their methods are described in detail.

Fourth chapter represents user’s guide for software application *Splines & Surfaces*. It describes all options located in menu and icons beneath it and options for modifying graphic visualisation. Significant part is dedicated to describing controls and modification of created geometric objects. The most important features are illustrated on screenshots from application.

The goal of software application *Splines & Surfaces* is visualisation of certain curves and surfaces in CAGD, which are examined in lecture in Representation of geometric objects [1], which belongs to bachelor degree of education on our university.

1. Chapter – Curve and Surface representation

We can represent curves and surfaces in three different ways: explicitly, implicitly and parametrically.

“Explicit representations of the form $y = f(x)$, although useful in many applications, are axis dependent, cannot adequately represent multiple-valued functions, and cannot be used where a constraint involves an infinite derivative. Hence, they are little used in computer graphics or computer aided design.“[2]

“Implicit representations of the form $f(x, y) = 0$ and $f(x, y, z) = 0$ for curves and surfaces, respectively, are capable of representing multiple-valued functions but are still axis dependent. However, they have a variety of uses in computer graphics and computer aided design.“[2]

The most important curve and surface representation for this work and computer graphics at all is parametric representation. For curves it has form

$$x = x(t), y = y(t), z = z(t)$$

where $x(t), y(t)$ and $z(t)$ are arbitrary functions and t is parameter from given interval. It can represent multi-valued functions and this representation is independent of choice of coordinate axes. Values of x, y and z designate coordinates of a point on the curve or surface. We will use Cartesian coordinate system and define position vector as

$$P(t) = \left(x(t), y(t), z(t)\right)$$

In this chapter we will first describe integral and then rational curves and surfaces.

1.1. Arcs

The simplest curve, or arc, is a straight line segment, given by equation

$$c(t) = V_0 + t(V_1 - V_0)$$

where $t \in [0,1], c(t)$ is a position vector and V_0 and V_1 are two different points in space. This curve has degree of one.

1.1.1. Bézier arc

Bézier curve is probably the best known curve model. Bézier curve of degree n is defined by sequence of $V_0, \dots, V_n$ points in space. These points are called control points and define control polygon. We can define Bézier curve of degree n as

$$c(t) = \sum_{i=0}^{n} B_i^n(t) V_i$$

where $t \in [0, 1], V_i$ stands for i–th control point and $B_i^n(t)$ is i–th Bernstein polynomial of degree n. (Figure 1-1)

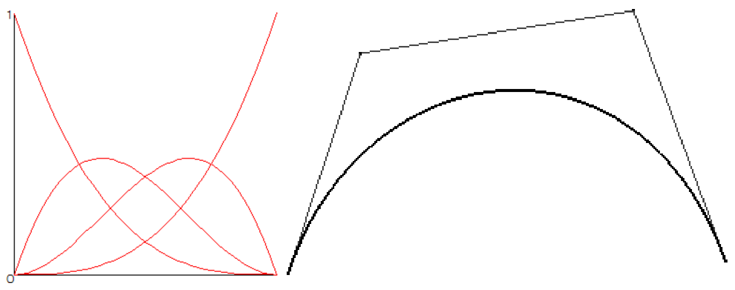


Figure 1-1 Bernstein polynomials and Bézier arc for degree $n = 3$

Bézier curve uses Bernstein polynomials as blending functions, which are defined for degree of n as

$$B_i^n(t) = \begin{cases} \binom{n}{i} t^i (1-t)^{n-i} & i = 0, \dots, n \\ 0 & otherwise \end{cases}$$

Bernstein polynomials have a number of properties:

- Non-negativity:

$$B_i^n(t) \geq 0, t \in [0,1]$$

- Symmetry:

$$B_i^n(t) = B_{n-i}^n(1-t), i = 0, \dots, n$$

- Partition of unity:

$$\sum_{i=0}^{n} B_i^n(t) = 1, t \in [0,1]$$

- Maximum: $B_i^n(t)$ has exactly one maximum on interval $[0, 1]$ at $t = \frac{i}{n}$.
- Basis: $B_i^n(t)$ form basis for polynomials of degree n.
- Recursion:

$$B_i^n(t) = (1-t)\, B_i^{n-1}(t) + t\, B_{i-1}^{n-1}(t), i = 0, \dots, n$$
$$B_{-1}^{n-1}(t) = B_n^{n-1}(t) = 0$$

- Derivative:

$$\frac{d}{dt} B_i^n(t) = n\left(B_{i-1}^{n-1}(t) - B_i^{n-1}(t)\right), i = 0, \dots, n$$

- Integral:

$$\int_0^1 B_i^n(t)\, dt = \frac{1}{n+1}, i = 0, \dots, n$$

Proofs of these properties can be found in [5] and [9].

Properties of Bézier curves are [1]:

- Bézier curve $c(u)$ generally follows shape of the control polygon $V_i, i = 0, \dots, n$.
- Endpoint interpolation property: Curve interpolates points V_0 and V_n

$$c(0) = V_0$$

$$c(1) = V_n$$

- Convex hull property: Every point of Bézier curve lies inside the convex hull of its defining control points.
- Symmetry property:

$$\sum_{i=0}^{n} B_i^n(t)V_i = \sum_{i=0}^{n} B_i^n(1-t)V_{n-i}$$

- Shape of curve does not change if we reverse order in control points' sequence and parameter t will change its value from 1 to 0.
- Invariance under parameter transformation:

$$\sum_{i=0}^{n} B_i^n(t)V_i = \sum_{i=0}^{n} B_i^n\left(\frac{u-a}{b-a}\right)V_i$$

for $t \in [0,1]$ and $u \in [a,b]$.

- Invariance under affine transformation:

$$T(c(t)) = T\left(\sum_{i=0}^{n} B_i^n(t)V_i\right) = \sum_{i=0}^{n} B_i^n(t)\, T(V_i)$$

where T is an affine transformation.

- Variation diminishing property: Intersection of control polygon and any hyperplane creates at least same number of points of intersection as intersection of that hyperplane and Bézier curve.
- Pseudo local control property: Bézier curves cannot be locally modified. Each change of position even of only one control vertex changes whole curve. By moving vertex V_i, the largest change in shape occurs for parametric values close to $t = i/n$.
- Derivatives [10]:

$$\frac{d^k}{dt^k}c(t) = n \cdots (n-k+1) \sum_{i=0}^{n-k} B_i^{n-k}(t)\, \Delta^k V_i$$

where $\Delta^k V_i$ is a vector defined recursively as

$$\Delta^0 V_i = V_i$$

$$\Delta^1 V_i = V_{i+1} - V_i$$

$$\Delta^k V_i = \Delta^1\left(\Delta^{k-1} V_i\right)$$

Now we can obtain tangents at endpoints

$$\frac{d}{dt}c(0) = n(V_1 - V_0)$$

$$\frac{d}{dt}c(1) = n(V_n - V_{n-1})$$

- Degree elevation [10]: Because $c(u)$ is a polynomial function of degree n, we can express it as polynomial of degree $n+1$. Thus

$$\sum_{i=0}^{n} B_i^n(t)V_i = \sum_{i=0}^{n+1} B_i^{n+1}(t)W_i$$

where W_i are control new points determined by equations

$$W_0 = V_0$$

$$W_i = \frac{i}{n+1}V_{i-1} + \left(1 - \frac{i}{n+1}\right)V_i, i = 1, \dots, n$$

$$W_{n+1} = V_n$$

This operation is called degree elevation and can be useful for more precise control of curve's shape.

- Degree reduction [10]: Reversing of degree elevation is called degree reduction. Because in general it is not possible to represent polynomial of degree n by polynomial of degree $n-1$, we are only approximating shape of original curve. Exact representation of the same curve with decreased degree is possible only in the case when degree was elevated beforehand. Let $W_i, i = 0, \dots, n+1$ be control points of Bézier curve of degree $n+1$. We construct two sequences of new control points A_i and B_i.

$$A_i = \frac{n+1}{n+1-i}\, W_i - \frac{i}{n+1-i}\, A_{i-1}, i = 0, \dots, n$$

$$B_i = \frac{n+1}{i+1}\, W_{i+1} - \frac{n-i}{i+1}\, B_{i+1}, i = n, \dots, 0$$

$$V_i = (1 - \alpha_i)\, A_i + \alpha_i\, B_i, i = 0, \dots, n$$

Now we have to blend these two sequences together to obtain control points for Bézier curve of degree n. There are multiple ways how to blend them, most intuitive is to choose $\alpha_i = \frac{i}{n}$. We define Bézier curves defined by control polygons $A_0, \dots, A_n$ and $B_0, \dots, B_n$, as $A(t)$ and $B(t)$, respectively.

"This method obviously has serious inaccuracies, particularly in the middle range (as could be anticipated from the above discussion which indicated that both $A(t)$ and $B(t)$ were most satisfactory near their starting points). We look to Matthias Eck for a refinement of reverse-elevation method." [6]

We minimise this error by constructing new control vertices by using "left" half of A_i and the "right" half of B_i

$$V_i = (1 - \alpha_i)A_i + \alpha_i\, B_i, for\ i = 0, \dots, n, \alpha_i = \begin{cases} 0\ if\ i < \frac{n}{2} \\ \frac{1}{2} if\ i = \frac{n}{2} \\ 1\ if\ i > \frac{n}{2} \end{cases}$$

Another approach is to define α_i as in [6]

$$\alpha_i = 2^{1-2n} \cdot \sum_{j=0}^{i} \binom{2n}{2j}, i = 0, \dots, n-1$$

Control points obtained by this method minimise error function

$$d(W,V) = \max\{|W(t) - V(t)|, t \in [0,1]\}$$

where $W(t)$ and $V(t)$ are Bézier curves defined by control polygons $W_0, \dots W_{n+1}$ and $V_0, \dots V_n$, respectively.

Enumeration of Bézier curves is mostly done by de Casteljau algorithm. It is based on recursive definition of Bernstein polynomials

$$B_i^n(t) = (1-t)\, B_i^{n-1}(t) + t\, B_{i-1}^{n-1}(t), i = 0, \dots, n$$

We start by defining

$$v_i^0(t) = V_i, i = 0, \dots, n$$

Then we compute points

$$v_i^k(t) = (1-t)\, v_i^{k-1}(t) + t\, v_{i+1}^{k-1}(t), k = 1, \dots, n, i = 0, \dots, n-k$$

In the last step we obtain point

$$v_0^n(t) = c(t)$$

which lies on curve and it is the same point that we will obtain if we enumerate $c(t)$.

Degree elevation can be also use to rasterize Bézier curve, as new vertices are approximating original control polygon. However this approximation is very slow compared to subdivision. This approach divides curve in two ones in such a way, that these two new curves joined together form original curve. Both of these curves are again Bézier curves of same degree as initial curve, so we can repeat subdivision process as many times as we need.

It can be proved that control vertices for these new curves are points obtained from de Casteljau algorithm. Vertices for first curve are $v_0^0, \dots, v_0^n$ and for second curve $v_0^n, \dots, v_n^0$. Subdivision mostly occurs at $t = 1/2$ and for rough idea how the curve looks like two subdivisions are sufficient for planar curve and three for curve in space. Otherwise we terminate subdivision process when the length of longest line segment is smaller than specified constant. This is also a way to model more complex curves that need more control vertices.

Modelling Bézier curves can be done only by moving control points in space, or changing multiplicity of these points.

1.1.2. Hermite arc

Cubic Hermite arc is determined by two control points V_0, V_1 and two tangents at these points $\overrightarrow{q_0}$ and $\overrightarrow{q_1}$. Formal definition as in [1] is

$$c(t) = H_0^3(t)V_0 + H_1^3(t)\overrightarrow{q_0} + H_2^3(t)\overrightarrow{q_1} + H_3^3(t)V_1$$

where $t \in [0, 1]$. (Figure 1-2) Blending functions $H_i^3(t), i = 0, 1,2, 3$ are called Hermite cubic polynomials. These functions can be derived from cubic Bernstein polynomials as

$$H_0^3(t) = B_0^3(t) + B_1^3(t) = 1 - 3t^2 + 2t^3$$

$$H_1^3(t) = \frac{1}{3}B_1^3(t) = t - 2t^2 + t^3$$

$$H_2^3(t) = -\frac{1}{3}B_2^3(t) = -t^2 + t^3$$

$$H_3^3(t) = B_2^3(t) + B_3^3(t) = 3t^2 - 2t^3$$

as in [3]. Cubic Hermite polynomials have following properties:

$$H_i^3(t) \geq 0, i = 0,1,3, H_2^3(t) \leq 0, t \in [0, 1]$$

$$H_0^3(t) + H_3^3(t) = 1$$

$$H_0^3(0) = H_3^3(1) = H_1^{3\prime}(0) = H_2^{3\prime}(1) = 1$$

$$H_0^3(1) = H_3^3(0) = H_1^{3\prime}(1) = H_2^{3\prime}(0) = 0$$

$$H_1^3(0) = H_1^3(1) = H_2^3(0) = H_2^3(1) = 0$$

$$H_0^{3\prime}(0) = H_3^{3\prime}(1) = H_0^{3\prime}(1) = H_3^{3\prime}(1) = 0$$

It can be shown using these properties that cubic Hermite curve interpolates both control points and its tangents at these points are equal to $\overrightarrow{q_0}$ and $\overrightarrow{q_1}$.

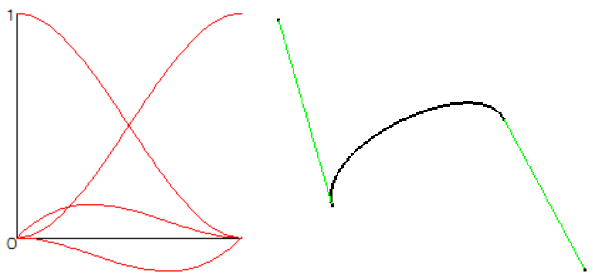


Figure 1-2 Cubic Hermite polynomials and Hermite arc

We can also define quintic Hermite curve for two control vertices, two tangents $\overrightarrow{q_0}, \overrightarrow{q_1}$ and two second derivatives $\overrightarrow{r_0}$ and $\overrightarrow{r_1}$ at these points as

$$c(t) = H_0^5(t)V_0 + H_1^5(t)\overrightarrow{q_0} + H_2^5(t)\overrightarrow{r_0} + H_3^5(t)\overrightarrow{r_1} + H_1^5(t)\overrightarrow{q_1} + H_5^5(t)V_1$$

where $t \in [0, 1]$ and $H_i^5(t), i = 0, \dots, 5$ are quintic Hermite polynomials. (Figure 1-3) They can be derived from Bernstein polynomials of degree 5 by equations

$$H_0^5(t) = B_0^5(t) + B_1^5(t) + B_2^5(t)$$

$$H_1^5(t) = \frac{1}{5}(B_1^5(t) + 2B_2^5(t))$$

$$H_2^5(t) = \frac{1}{20}B_2^5(t)$$

$$H_3^5(t) = \frac{1}{20}B_3^5(t)$$

$$H_4^5(t) = -\frac{1}{5}(2B_3^5(t) + B_4^5(t))$$

$$H_5^5(t) = B_3^5(t) + B_4^5(t) + B_5^5(t)$$

as in [3]. This curve interpolates both control points, first derivatives at these points are $\overrightarrow{q_0}$ and $\overrightarrow{q_1}$ and second derivatives $\overrightarrow{\mathrm{r}_0}$ and $\overrightarrow{\mathrm{r}_1}$.

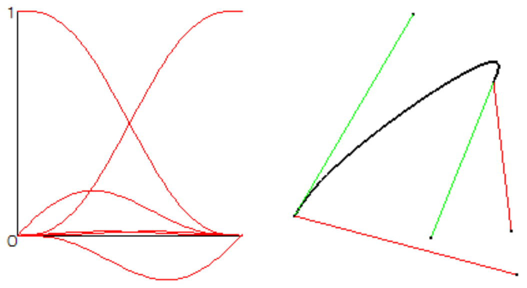


Figure 1-3 Quintic Hermite polynomials and Hermite arc

For calculating point on given curve we can simply calculate value of $c(t)$ or we use Horner's scheme. It is based on gradual factorisation and consists of n steps for polynomial of degree n. Each step includes one multiplication and one summation.

Modelling Hermite arc can be done via changing position of one or both control points. We can also change orientation and magnitude of tangents and in case of quintic arc also vectors for second derivative.

1.2. Splines

Working with curves of high order is not appropriate for tasks such as modelling or interpolation, because single change in even one control point results in change of whole curve. Instead a spline curve is used. It has various uses in mathematics, informatics, CAD, approximation theory, design and many others. Splines are divided into two major groups, interpolation and approximation splines. Interpolation splines are generally used for image digitalization or describing animation paths and approximation splines for designing and modelling.

Spline is a piecewise curve formed by number of segments. These segments itself are curves of different types. They are joined at joining points and continuity at these points can be changed to suit our needs. General definition of spline curve in n dimensional space is

$$s(u) : [a,b] \to \mathbb{R}^n$$

where $u \in [a,b]$ and this interval is composed of subintervals $[u_i, u_{i+1}]$ such that $a = u_0 < u_1 < \cdots < u_{n-1} < u_n = b$. We call parameter u global parameter and local parameter $t \in [0,1]$ is defined as

$$t = \frac{u - u_i}{u_{i+1} - u_i}$$

Each subinterval has its length denoted by $\Delta^i = u_{i+1} - u_i$. Restriction of spline s on interval $[u_i, u_{i+1}]$ is segment

$$s_i : [u_i, u_{i+1}] \to \mathbb{R}^n$$

so that

$$s(u) = s_i(t)$$

where $u \in [u_i, u_{i+1}], t \in [0,1]$ and maximum degree of individual segments equals degree of spline. Points that satisfy equations $s_i\,(u_{i+1}) = s_{i+1}\,(u_{i+1})$ are called joining points. If each subinterval has equal length, for example $u_i = i/n$, then we call spline uniform, otherwise it is non-uniform spline.

1.2.1. Continuity conditions for splines

We distinguish two types of continuity, parametric and geometric continuity. Firstly, we examine parametric continuity. C^0 means that position vector for points on curve changes smoothly along whole curve. For two given two curve segments s_i and s_{i+1} on intervals $[u_i, u_{i+1}]$ and $[u_{i+1}, u_{i+2}]$, respectively, we define C^0continuity as $s_i\,(u_{i+1}) = s_{i+1}\,(u_{i+1})$. General definition of C^n continuity requires fulfilling C^{n-1}continuity conditions and thus $s_i^{(n)}(u_{i+1}) = s_{i+1}^{(n)}\,(u_{i+1})$. In practice, $C^1$and C^2 continuity are mostly used. C^1 continuity ensures that tangent vector at point corresponding to $u = u_{i+1}$ is the same for both segments and thanks to C^2 continuity curvature of curve changes smoothly. Example from real life for C^2 continuity can be path of an object, which should move with without discontinuous changes in position, speed and acceleration, otherwise it will look unnatural.

Geometric continuity is less demanding. G^0 stays the same as C^0 but G^1 is different. It requires G^0 continuity and condition $s_{i+1}'\,(u_{i+1}) = \beta_1\, s_i'\,(u_{i+1})$ where $\beta_1 > 0$ must be fulfilled. So tangent vector at $u = u_{i+1}$ needs to be only positive scalar multiple of tangent vector of another segment at the same point. For G^2 continuity curve must satisfy G^1 continuity and following equation $s_i''\,(u_{i+1}) = \beta_1^2 s_{i+1}''\,(u_{i+1}) + \beta_2\, s_{i+1}'\,(u_{i+1})$ where $\beta_2 \in R$.

“Although many application find G^1 continuity adequate, for applications that depend on the fairness or smoothness of a curve or surface, especially those that depend on a smooth transition of reflected light, e.g., automobile bodies, G^1 or even G^2 continuity is not adequate. For these applications, at least C^2 continuity is required to achieve the desired results.” [4]

Most splines usually leave some degree of freedom for user to modify shape of curve by changing end conditions, which can be defined in various ways, for example:

- Clamped splines has fixed tangent vector at the end points.

$$s_0'\,(u_0) = \overrightarrow{q_0}, s_{n-1}'\,(u_n) = \overrightarrow{q_n}$$

- Natural or relaxed spline has zero curvature at the end points.

$$s_0''(u_0) = \vec{0}, s_{n-1}''(u_n) = \vec{0}$$

- Cyclic spline has equal first and second derivatives at the end points.

$$s_0'(u_0) = s_{n-1}'(u_n), s_0''(u_0) = s_{n-1}''(u_n)$$

- Acyclic spline has equal but opposite derivatives at the end points.

$$s_0'(u_0) = -s_{n-1}'(u_n), s_0''(u_0) = -s_{n-1}''(u_n)$$

- Quadratic condition for splines is defined as equality of second derivative at first two and last two points.

$$s_0''(u_0) = s_1''(u_1), s_{n-2}''(u_{n-1}) = s_{n-1}''(u_n)$$

1.2.2. Cubic Hermite spline

Cubic Hermite spline is a spline composed from cubic Hermite arcs, so it is interpolation spline by its definition [1]. For given control points $V_0, \dots, V_k, k \geq 2$, we need to calculate tangents at these points in order to construct C^2 continuous spline. Formal definition of i-th segment is

$$s_i(u) = H_0^3(t)V_i + H_1^3(t)\vec{q_i} + H_2^3(t)\vec{q_1} + H_3^3(t)V_{i+1}, i = 0, \dots, k-1$$

where $u \in [a, b], t = \frac{u-u_i}{u_{i+1}-u_i}$ and $H_i^3(t), i = 0,1,2,3$ are cubic Hermite blending functions. (Figure 1-4)

After expanding equations $s_i''(1) = s_{i+1}''(0)$, we obtain system of $k-1$ equations for $k+1$ variables. In order to solve this system we choose one end condition. Then it is possible to find all tangents and compute each Hermite segment.

Hermite spline is also called global spline, because each change of control vertices modifies whole curve. We describe this property of spline as not having local control. Another way to modify this kind of spline is to change end condition.

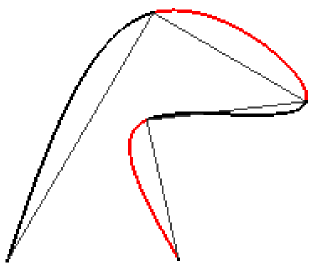


Figure 1-4 Relaxed Hermite spline

1.2.3. Cardinal spline

Cardinal spline is a modification of Hermite spline, in which all tangent vectors at control points are defined by adjacent points [1]. Because of this, we do not have to choose end condition. Again, let $V_0, \dots, V_k, k \geq 3$ be sequence of points in space and we want to interpolate these vertices with Hermite cubic spline. Cardinal spline segment can we expressed as

$$s_i(u) = \sum_{j=-1}^{2} C_j(t)V_{i+1+j}\,, i = 0, \dots, k-3$$

where $u \in [a,b], t = \frac{u-u_i}{u_{i+1}-u_i}$ and $C_j(t), j = -1,0,1,2$ are cardinal blending functions. (Figure 1-5)

Each Hermite cubic spline segment is defined by two points and two tangent vectors at these points. We define tangent vector $\overrightarrow{q_i} = s\,(V_{i+1} - V_{i-1}), s > 0$, so now each segment is defined for foursome of points $V_{i-1}V_iV_{i+1}V_{i+2}$. Parameter s is called proportional coefficient as its value represents ratio of magnitude of tangent vector $\overrightarrow{q_i}$ and length of line segment between V_{i-1} and V_{i+1}. Now we define cardinal blending function $C_i(t), i = -1, 0, 1, 2$, as

$$C_{-1}(t) = -st + 2st^2 - st^3$$

$$C_0(t) = 1 + (s-3)t^2 + (2-s)t^3$$

$$C_1(t) = st + (3-2s)t^2 + (s-2)t^3$$

$$C_2(t) = -st^2 + st^3$$

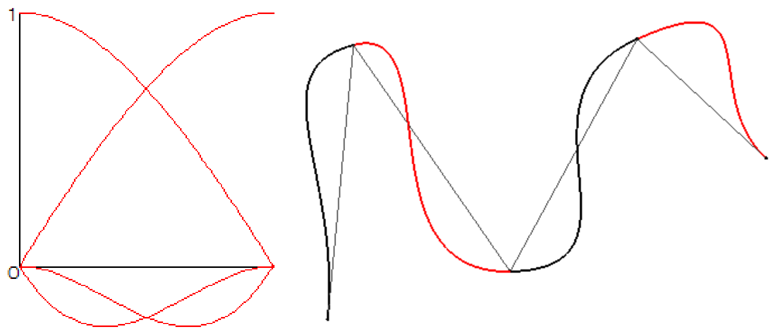


Figure 1-5 Cardinal polynomials and spline with double endpoints for $s = 1{,}6$

These functions have property of partition of unity and cardinal spline is $C^0$and C^1 continuos, but conditions for C^2 nor G^2continuity are not fulfilled.

Cardinal spline interpolates vertices $V_i, i = 1, \dots, k-1$. Points V_0 and V_k are free ends of curve. We can interpolate these free ends in two ways. First method is called multiple vertices. We add two vertices $V_{-1} = V_0$ and $V_{k+1} = V_k$, so we can create two new curve segment for points $V_{-1}V_0V_1V_2$ and $V_{k-2}V_{k-1}V_kV_{k+1}$.

Following method give us better options for modelling shape of curve. This one is called phantom vertices, because we once again add two vertices V_{-1} and V_{k+1} but this time position of these vertices is not fixed. For clamped spline we have given first derivatives at end points so we can express V_{-1} and V_{k+1} as

$$V_{-1} = V_0 - \frac{1}{s}\,\overrightarrow{q_0}$$

$$V_{k+1} = V_k + \frac{1}{s}\,\overrightarrow{q_k}$$

In case of relaxed spline second derivatives at end points are equal to zero vector. Equation for new phantom points are

$$V_{-1} = V_1 - \frac{3}{2s}(V_1 - V_0) + \frac{1}{2}(V_2 - V_0)$$

$$V_{k+1} = V_{k-1} + \frac{3}{2s}(V_k - V_{k-1}) + \frac{1}{2}(V_k - V_{k-2})$$

Modelling cardinal spline is done mostly by changing proportional coefficient s, which is global parameter, as change in its value modify tension and thus shape of whole curve. This can cause undesirable changes in shape of curve mainly if distance between adjacent control points are significantly different. Raising value of s causes curve to deviate less from tangent vectors. These curve has low tension and can create loops. Splines with lower values of s deviate more from tangent vector and the largest deviation occurs a $s = 0$, when spline segment s_i is line segment $V_i V_{i+1}$. We define tension parameter T as $s = (1-T)/2$ so $T = 1 - 2s$. Spline with proportional coefficient $s = 1/2$ has tension $T = 0$ so it is called a spline with zero tension or Catmull-Romm spline or Overhauser spline.

1.2.4. Bézier spline

Bézier spline is a curve, which is composed of Bézier curve segments and can be constructed both as interpolation and approximation spline. We assume these segments are cubic and are joined together to be C^2 continuous.

Definition of Bézier spline s of degree n for points $V_0, \dots, V_k, k \geq 2$ as in [5] is

$$s_i(u) = \sum_{j=0}^{n} B_j^n(t) V_{i,j}, i = 0, \dots, k-1$$

where $u \in [a,b], t = \frac{u-u_i}{u_{i+1}-u_i}, V_{i,j}$ are control points and $B_j^n(t)$ is j-th Bernstein polynomial of degree n. Algorithm for constructing new control points $V_{i,j}$ depends on type of spline. (Figure 1-6)

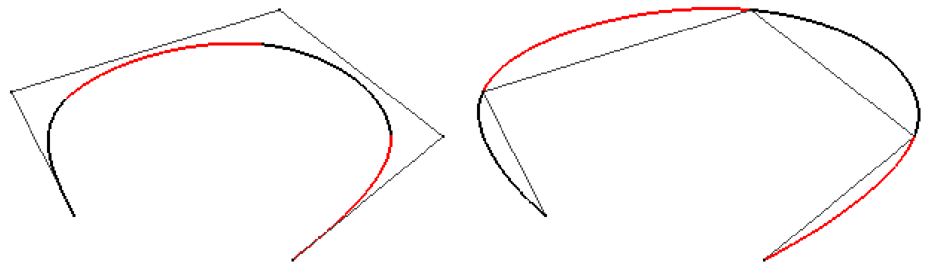


Figure 1-6 Approximation and interpolation cubic Bézier spline

Assume that we have i-th cubic Bézier curve segment s_i defined with control points $V_i, V_{i+1}, V_{i+2}, V_{i+3}$ and we want to add $i+1$-th segment s_{i+1} with control vertices $V_{i+4}, V_{i+5}, V_{i+6}, V_{i+7}$, such that the two curves will be C^2 continuous. By solving continuity conditions we obtain equations for control points of $i+1$-th segment as in [1]:

- C^0 continuity:

$$s_i(u_{i+1}) = s_i(u_{i+1})$$

and thus

$$V_{i+3} = V_{i+4}$$

- C^1 continuity:

$$V_{i+5} = V_{i+3} + \frac{\Delta_{i+1}}{\Delta_i}(V_{i+3} - V_{i+2})$$

  and thus $ratio(V_{i+2}, V_{i+3}, V_{i+5}) = \frac{\Delta_i}{\Delta_{i+1}}$.

- C^2 continuity: We need to construct point D_i

$$D_i = V_{i+2} + \frac{\Delta_{i+1}}{\Delta_i}(V_{i+2} - V_{i+1})$$

  for which $ratio(V_{i+1}, V_{i+2}, D_i) = \frac{\Delta_i}{\Delta_{i+1}} = ratio(D_i, V_{i+5}, V_{i+6})$ thus point D_i is an intersection of lines defined by V_{i+1}, V_{i+2} and V_{i+5}, V_{i+6}. We use it in computation of control point

$$V_{i+6} = V_{i+5} + \frac{\Delta_{i+1}}{\Delta_i}(V_{i+5} - D_i)$$

Vertex V_{i+7} can be positioned freely without compromising C^2 continuity. If Δ_i is constant for $i = 0, \dots, k-1$ then equations for control vertices V_{i+4}, V_{i+5} and V_{i+6} are simplified to

$$V_{i+5} = V_{i+3} + (V_{i+3} - V_{i+2})$$

$$V_{i+6} = V_{i+5} + (V_{i+5} - D) = V_{i+1} + 4(V_{i+3} - V_{i+2})$$

Use of Bézier approximation spline is generally better approach than constructing single Bézier curve of high degree, mainly because this single curve does not approximate control vertices very well.

In this case, points $V_{i,j}$ are derived from control polygon formed by vertices $V_0, \dots, V_k$. In this work we use cubic Bézier spline, so $n = 3$, and $\Delta_i = 1, i = 0, \dots, k-1$. We divide each line segment V_iV_{i+1} by points $V_{i,j}$ so $ratio(V_i, V_{i,1}, V_{i,2}) = ratio(V_{i,1}, V_{i,2}, V_{i+1}) = 1$. Now we have to define position of vertices $V_{i,0}$ and $V_{i,3}$ to position suitable for securing C^2 continuity.

If $i = 0$ or $i = k-1$, we set $V_{i,0} = V_0$ or $V_{i,3} = V_k$, respectively. In order to secure C^0 continuity at joining points we have to set $V_{i,3} = V_{i+1,0}$. C^1 continuity will hold if

$$V_{i,3} = V_{i+1,0} = \frac{1}{2}V_{i,2} + \frac{1}{2}V_{i+1,1}$$

Finally from C^2 continuity we have

$$V_{i+1,2} = V_{i+1,1} + (V_{i+1,1} - V_i)$$

where $V_i = D_i$.

In the case of interpolation Bézier spline it is little more complicated. Once again let $n = 3$ be degree of spline and $\Delta_i = 1, i = 0, \dots, k-1$. First we need to compute coordinates of points D_i. We do it by expanding equations $s_i''(1) = s_{i+1}''(0)$ and obtaining linear system of $k-1$ equations for $k+1$ variables. We need to specify points $V_{0,1}$ and $V_{k-1,2}$ via end condition.

Now points D_i are known and we use same procedure for computing points $V_{i,j}$ as before, but now with input consisting of points D_i. Then have to change position of points $V_{i,0}$ and $V_{i,3}$ (which now have same coordinates as D_0 and D_k) to

$$V_{i,0} = V_0, V_{i,3} = V_k$$

in order to interpolate these vertices.

We rasterize Bézier spline with de Casteljau algorithm. Each segment has its own control polygon and we use its control points for algorithm.

Approximation Bézier spline is a spline with local control. Change in position of control point V_i modifies $i-2, i-1, i$ and $i+1$-th segments. Interpolation Bézier spline is a global one.

1.2.5. Beta-Spline

Beta spline is a cubic curve on a given sequence of control vertices $V_0, \dots, V_k, k \geq 4$ in space composed of segments defined in [1] as

$$s_i(\beta_1, \beta_2, t) = \sum_{j=0}^{3} Q_j(\beta_1, \beta_2, t)\, V_{i+j}, i = 0, \dots, k-3$$

where $u \in [a, b], t = \frac{u - u_i}{u_{i+1} - u_i}$. (Figure 1-7) Blending functions $Q_j(\beta_1, \beta_2, t)$ are defined in a way to secure G^2 continuity for a whole curve in [5] as

$$Q_0(t) = \frac{2{\beta_1}^3 - 6{\beta_1}^3 t + {\beta_1}^3 t^2 - 2{\beta_1}^3 t^3}{\delta}$$

$$Q_1(t) = \frac{(4{\beta_1}^2 + 4\beta_1 + \beta_2) + (6{\beta_1}^3 - 6\beta_1)t - 3(2{\beta_1}^3 + 2{\beta_1}^2 + \beta_2)t^2 + 2({\beta_1}^3 + {\beta_1}^2 + \beta_1 + \beta_2)t^3}{\delta}$$

$$Q_2(t) = \frac{2 + 6\beta_1 t + 3(2{\beta_1}^2 + \beta_2)t^2 - 2({\beta_1}^2 + \beta_1 + \beta_2 + 1)t^3}{\delta}$$

$$Q_3(t) = \frac{2t^3}{\delta}$$

$$0 \neq \delta = 2{\beta_1}^3 + 4{\beta_1}^2 + 4\beta_1 t^2 + \beta_2 + 2$$

If $\beta_1 > 0$ and $\beta_2 \geq 0$, then these functions have property partition of unity and non-negativity, thus whole curve lies inside convex hull of its defining control polygon.

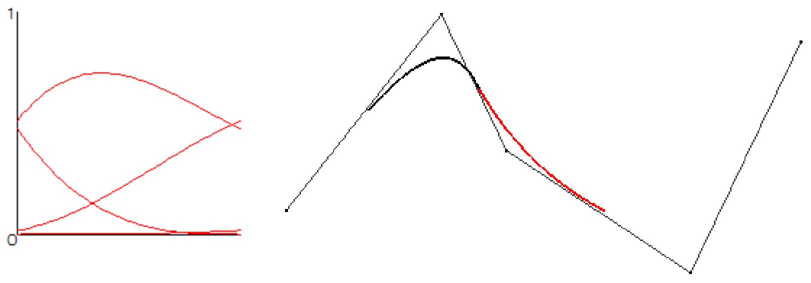


Figure 1-7 Beta polynomials and Beta spline for $\beta_1 = 3$ *and* $\beta_2 = 10$

End points of curve segment s_i satisfy equations

$$s_i(\beta_1, \beta_2, 0) = \frac{1}{\delta}\left[2{\beta_1}^3 V_i + \left(4{\beta_1}^2 + 4\beta_1 + \beta_2\right)V_{i+1} + 2\, V_{i+2}\right]$$

$$s_i(\beta_1, \beta_2, 1) = \frac{1}{\delta}\left[2{\beta_1}^3 V_{i+1} + \left(4{\beta_1}^2 + 4\beta_1 + \beta_2\right)V_{i+2} + 2\, V_{i+3}\right]$$

so for $\beta_1 > 0$ and $\beta_2 \geq 0$ they are inside of triangles formed by V_i, V_{i+1}, V_{i+2} and $V_{i+1}, V_{i+2}, V_{i+3}$, respectively. Spline's starting and ending points are then inside of triangles formed by V_0, V_1, V_2 and V_{k-2}, V_{k-1}, V_k, respectivelly.

Once again we can change spline's end by multiple vertices method or by phantom vertices. Multiple vertices work by adding new control points $V_{-1} = V_0$ and $V_{k+1} = V_k$ creating new ending points

$$s_{-1}(\beta_1, \beta_2, 0) = V_0 + \frac{1}{\delta}[V_1 - V_0]$$

$$s_{k-2}(\beta_1, \beta_2, 1) = V_k + \frac{2{\beta_1}^3}{\delta}[V_{k-1} - V_k]$$

or by adding new control points $V_{-2} = V_0, V_{-1} = V_0, V_{k+1} = V_k$ and $V_{k+2} = V_k$. The result is curve interpolating control points V_0 and V_k.

Phantom vertices method adds new control points V_{-1} and V_{k+1} in such way, that

$$s_0{}'(\beta_1, \beta_2, 0) = \overrightarrow{q_0}$$
$$s_{k-3}{}'(\beta_1, \beta_2, 1) = \overrightarrow{q_1}$$

or adds another points V_{-2} and V_{k+2} to create relaxed spline - curve with zero curvature

$$s_0{}''(\beta_1, \beta_2, 0) = \vec{0}$$

$$s_{k-3}{}''(\beta_1, \beta_2, 1) = \vec{0}$$

We can change shape of Beta-spline by modifying β_1 and β_2 parameters. Parameter β_1 is called bias and we can express it as

$$\beta_1 = \frac{|s_{i+1}{}'(\beta_1, \beta_2, 0)|}{|s_i{}'(\beta_1, \beta_2, 1)|}$$

For parameter $\beta_1 > 1$ is $i + 1$-th segment more attracted to common tangent than i-th segment and for $0 < \beta_1 < 1$ vice versa. If $\beta_1 = 1$ then deviation from said tangent is the same for both segments. Parameter β_2 affects how much joining point S_i of two following segments i and $i + 1$ are attracted to corresponding control vertex V_{i+2}.

$$s_i(\beta_1, \beta_2, 1) = \frac{1}{\delta}\left[2{\beta_1}^3 V_{i+1} + \left(4{\beta_1}^2 + 4\beta_1 + \beta_2\right)V_{i+2} + 2\, V_{i+3}\right] = S_i$$

Now if we define $k = \delta - \beta_2$ and

$$K_i = 2{\beta_1}^3 V_{i+1} + \left(4{\beta_1}^2 + 4\beta_1\right)V_{i+2} + 2\, V_{i+3}$$

we obtain equation

$$S_i - V_{i+2} = \frac{1}{k + \beta_2}[K_i + kV_{i+2}]$$

For parameter $\beta_2 > 0$ point S_i is approaching point V_{i+2}, for $\beta_2 < 0$ we get same effect but some of blending functions can be negative and thus curve will not be in convex hull formed by control polygon. If $\beta_2 \to -k$ then point S_i is pushed away from point V_{i+2}. We can conclude that parameter β_2 changes tension of spline so we call it tension parameter.

1.2.6. B-spline

B-spline is approximation curve for given control polygon and important role in modelling of this type of curve is played by knot vector. We can define B-spline of degree k for given control vertices $V_0, \dots, V_n$ and knot vector $U = \{u_i\}_{i=0}^{m}$ where $m = n + k + 1$ as in [10] by equation

$$s(u) = \sum_{i=0}^{n} N_i^k(U, u)V_i$$

where $u \in [u_k, u_{m-k})$ and $N_j^k(U, u)$ are B-spline basis functions. (Figure 1-8)

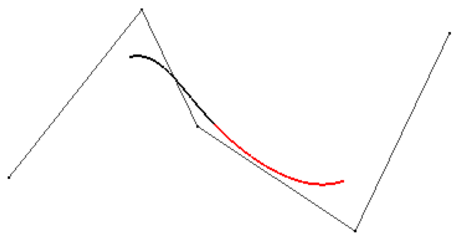


Figure 1-8 B-spline of degree $n = 3$ with uniform knot vector

Knot vector U is a non-decreasing sequence of numbers u_i such that $U = \{u_i\}_{i=0}^{m}$. Elements u_i are called knots. For each u_i we define $j = \arg min_{l \le i}\{u_l = u_i\}$ and $k = \arg max_{l \ge i}\{u_l = u_i\}$. We call number $\mu(u_i) = k - j + 1$ multiplicity of a knot u_i. Knots $u_0, \dots, u_{\mu(u_0)}$ and $u_{m-\mu(u_m)}, \dots, u_m$ are called external knots and all knots between them are called internal knots. Most used types of knot vector are open, uniform or non-uniform.

Uniform knot vector, sometimes also called periodic, has all or only internal knots equally spaced and usually begins at zero. It also can be normalized, so each $u_i \in [0, 1]$. Non-uniform knot vector is unevenly spaced or has multiple knots. Open knot vector has external knots of multiplicity $k+1$.

Assume k is a given degree of B-spline basis function $N_i^k(U,u)$ and $U = \{u_i\}_{i=0}^{m}$, $m = n + k + 1$, is knot vector in which every knot has maximum knot multiplicity of $k+1$. We define functions $N_i^k(U,u)$ recursively as in [10] by equations

$$N_i^0(U,u) = \begin{cases} 1 & if\ u \in [u_i, u_{i+1}) \\ 0 & otherwise \end{cases}, i = 0, \dots, m-1$$

$$N_i^l(U,u) = \begin{cases} \dfrac{u - u_i}{u_{i+l} - u_i} N_i^{l-1}(U,u) + \dfrac{u_{i+l+1} - u}{u_{i+l+1} - u_{i+1}} N_{i+1}^{l-1}(U,u) & u \in [u_i, u_{i+l+1}) \\ 0 & otherwise \end{cases}$$

for $i = 0, \dots, m-l-1$ and $l = 1, \dots, k$. To avoid zero denominator and thus zero division problem we consider this summand equal to zero. In case of $u_i = u_{i+1}$ we define $N_i^0(U,u) = 0$. (Figure 1-9 and Figure 1-10)

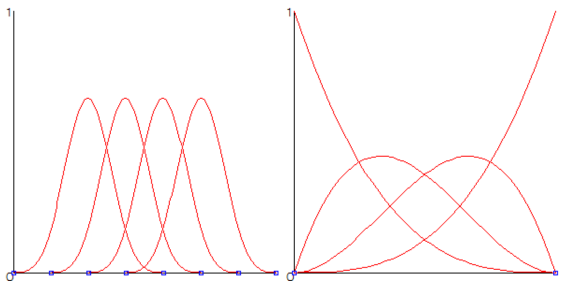


Figure 1-9 B-spline polynomials of degree $n = 3$ for uniform knot vector $U = \{0,1,2,3,4,5,6,7\}$ and for open knot vector $U = \{0,0,0,0,1,1,1,1\}$, respectively

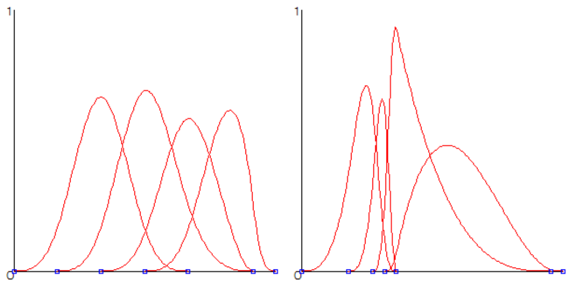


Figure 1-10 B-spline polynomials of degree $n = 3$ for knot vector $U = \{0,1,2,3,4,5.5,5.5,6\}$ and for knot vector $U = \{0,1,1.5,1.75,2,2,5.25,5.5\}$, respectively

B-spline basis functions have these properties:

- Non-negativity:

$$N_i^k(U,u) \geq 0, u \in R, i = 0, \dots, m-k-1$$

- Local support:

$$N_i^k(U,u) = 0, u \notin [u_i, u_{i+k+1})$$
$$N_i^k(U,u) \geq 0, u \in [u_i, u_{i+k+1})$$

for $i = 0, \dots, m-k-1$.

- Partition of unity:

$$\sum_{i=0}^{m-k} N_i^k(U,u) = 1, u \in [u_n, u_{m-k})$$

- Derivative:

$$\frac{d}{du} N_i^k(U,u) = k\left(\frac{1}{u_{i+k} - u_i} N_i^{k-1}(U,u) - \frac{1}{u_{i+k+1} - u_{i+1}} N_{i+1}^{k-1}(U,u)\right)$$

  for $i = 0, \dots, m-k-1$ if derivative at u exists.

- Linear independence: Functions $N_i^k(U,u)$ are linear independent for $i = 0, \dots, m-k-1$.
- Uni-modality: Function $N_i^k(U,u)$ has exactly one maximum for $i = 0, \dots, m-k-1$ and $k \geq 1$
- Periodicity: This property is valid only for uniform knot vector.
$$N_i^k(U,u) = N_{i-1}^k(U, u-d) = N_{i+1}^k(U, u+d)\,, d = u_i - u_{i-1}$$
- Recursive property:
$$N_i^k(U,u) = \frac{u - u_i}{u_{i+k} - u_i} N_i^{k-1}(U,u) + \frac{u_{i+k+1} - u}{u_{i+k+1} - u_{i+1}} N_{i+1}^{k-1}(U,u)$$
  for $i = 0, \dots, m-k-1$

Proofs of these properties can be found in [10].

B-spline curve has following properties [9]:

- B-spline curve $s(u)$ generally follows shape of control polygon $V_i, i = 0, \dots, n$.
- Domain for curve is interval $u \in [u_k, u_{m-k})$.
- Local modelling: Curve segment corresponding to interval $u \in [u_i, u_{i+1})$, $k \leq i \leq m-k-1$ is designated by $k+1$ control vertices $V_{i-k}, \dots, V_i$.
- Convex hull property: For $u \in [u_i, u_{i+1}), k \leq i \leq m-k-1$ point on curve $s(u)$ lies in convex hull formed by vertices $V_{i-k}, \dots, V_i$.
- Variation diminishing [10]: Intersection of control polygon and any hyperplane creates at least same number of points of intersection as number of intersections of said hyperplane and B-spline curve.
- Continuity: B-spline is $C^{k-\mu(u_i)}$ continuous at points $s(u_i)$ and C^∞ continuous everywhere else. If internal knot u_i has multiplicity of $k+1$, then curve can be C^0 uncontinuous.
- Derivatives [9]:
$$\frac{d^l s(u)}{du^l} = k \dots (k-l+1) \sum_{i=0}^{n-l} N_i^{n-l}(U,u) \Delta^l V_i$$

  where $\Delta^l V_i$ is a vector defined recursively as

$$\Delta^0 V_i = V_i$$

$$\Delta^1 V_i = \frac{V_{i+1} - V_i}{u_{i+k+1} - u_{i+1}}$$

$$\Delta^l V_i = \Delta^1\left(\Delta^{l-1} V_i\right) = \frac{\Delta^{l-1} V_{i+1} - \Delta^{l-1} V_i}{u_{i+k+1} - u_{i+l}}$$

- Invariance under affine transformation:

$$T\big(s(u)\big) = T\left(\sum_{i=0}^{n} N_i^n(U,u)V_i\right) = \sum_{i=0}^{n} N_i^n(U,u)T(V_i)$$

where T is an affine transformation.

Cubic B-spline curve segment with periodic knot vector has end points defined by these equations [1]

$$s(u_{3+i}) = \frac{1}{6}(V_i + 4V_{i+1} + V_{i+2}) = V_{i+1} + \frac{1}{3}\left(\frac{V_i + V_{i+2}}{2} - V_{i+1}\right)$$

$$s(u_{3+i+1}) = \frac{1}{6}(V_{i+1} + 4V_{i+2} + V_{i+3}) = V_{i+2} + \frac{1}{3}\left(\frac{V_{i+1} + V_{i+3}}{2} - V_{i+2}\right)$$

for $i = 0, \dots, n-3$. Both points are anti-centroid of triangle formed by $V_iV_{i+1}V_{i+2}$ and $V_{i+1}V_{i+2}V_{i+3}$, respectively. First and second derivatives at end points of segment are defined by following equations

$$s'(u_{3+i}) = \frac{1}{2}(V_{i+2} - V_i)$$

$$s'(u_{3+i+1}) = \frac{1}{2}(V_{i+3} - V_{i+1})$$

$$s''(u_{3+i}) = (V_i - 2V_{i+1} + V_{i+2}) = (V_i - V_{i+1}) + (V_{i+2} - V_{i+1})$$

$$s''(u_{3+i+1}) = (V_{i+1} - 2V_{i+2} + V_{i+3}) = (V_{i+1} - V_{i+2}) + (V_{i+3} - V_{i+2})$$

We can control end points by applying multiple coincident vertices [2]. By defining multiple coincident vertices at endpoint of the B-spline curve resulting curve is pulled toward to vertex with modified multiplicity. If multiplicity of this vertex is k (for B-spline of degree k), then B-spline interpolates this vertex and tangent vector at this point is parallel to first or last adjacent control polygon span with non-zero length. If we set multiplicity of an end point to $k+1$, first or last segment of curve will be a line segment parallel to first or last control polygon span. This can be unpleasant for modelling and use of open knot vector is better for purpose of interpolating starting and ending points.

Process of knot insertion is usually performed to gain better control of curve or to enumerate curve at given parametric value. After knot insertion curve's shape remains the same but control polygon will change and also number of curve segments will be increased by one. If we insert just one knot, then we are talking about knot insertion but if we insert multiple knots then we call it knot refinement.

Let $s(u)$ be a B-spline curve of degree k with control vertices $V_0, \dots, V_n$ and $U = \{u_i\}_{i=0}^{m}, m = n + k + 1$, is its knot vector. We insert new knot $u^* \in \left[u_j, u_{j+1}\right)$ to create new knot vector $U^* = \{u_i^*\}_{i=0}^{m+1}$, such that

$$u_i^* = \begin{cases} u_i & i = 0, \dots, j \\ u^* & i = j+1 \\ u_i - 1 & i = j+2, \dots, m+1 \end{cases}$$

Determining new control vertices for curve of identical shape

$$s(u) = \sum_{i=0}^{n+1} N_i^k(U^*, u) V_i^*$$

yields [9]

$$V_i^* = \begin{cases} V_i & 0 \le i \le j-k \\ (1-\alpha_i)V_{i-1} + \alpha_i V_i & j-k+1 \le i \le j \\ V_{i-1} & j+1 \le i \le n+1 \end{cases}, \alpha_i = \frac{u_j - u_i}{u_{i+k} - u_i}$$

Shown procedure is known as Boehm’s algorithm.

“Unfortunately, this so called knot removal procedure cannot be carried out in general without changing the shape of the spline curve. Obviously, the only exception occurs if a knot has been inserted artificially before or, in other words, if the continuity order at the respective knot is higher than it should be according to its multiplicity. Thus, the necessity to interpret the knot removal process as an approximation process is manifest.” [8]

For given B-spline curve with degree k, we remove knot u_s, such that $u_s \neq u_{s+1}$, which has multiplicity m, then new control points are defined as in [2] by equation

$$V_i^* = \begin{cases} V_i & 0 \le i \le s-k-1 \\ (V_i - (1-\alpha_i)V_{i-1}^*)/\alpha_i & s-k \le i \le c \\ (\beta_i V_i - \beta_i V_i^*)/(1-\beta_i) & c < i \le s-m \\ V_{i+1} & s-m+1 \le i \le n-1 \end{cases} \quad c = \frac{k-m+1}{s-m}$$

$$\alpha_i = \frac{u_j - u_i}{u_{i+k} - u_i}, \beta_i = \frac{u_{j+1} - u_{i+1}}{u_{i+k+1} - u_{i+1}}$$

We use de Boor algorithm for evaluating B-spline curve of degree k. Similarly to de Casteljau algorithm this algorithm makes use of recursive property of B-spline basis function

$$N_i^k(U,u) = \frac{u - u_i}{u_{i+k} - u_i} N_i^{k-1}(U,u) + \frac{u_{i+k+1} - u}{u_{i+k+1} - u_{i+1}} N_{i+1}^{k-1}(U,u)$$

for $i = 0, \dots, m-k-1$. Given parameter $u \in [u_k, u_{m-k})$ we start by finding knot interval $[u_r, u_{r+1})$ such that $u \in [u_r, u_{r+1})$. Then we define

$$r_i^0(u) = V_i, i = r-k, \dots, r$$

and then compute

$$r_i^j(u) = \left(1 - \alpha_i^j\right) r_{i-1}^{j-1}(u) + \alpha_i^j\, r_i^{j-1}(u), i = r-k+j, \dots, r, j = 1, \dots, k$$

where $\alpha_i^j(u) = \frac{u - u_i}{u_{i+k+1-j} - u_i}$. In the last step we obtain point

$$r_r^k(u) = s(u)$$

and it is a point on curve corresponding to parametric value of u.

We can modify shape of B-spline by various ways, most obvious is changing knot vector. We can also change positions and multiplicity of control points or use multiple knot values.

1.3. Rational curves

All curve described so far were subsets of points from E^2 or E^3 and we call them integral curves. Rational curves were introduced to further augment control over shape of curve. Point (x, y, z) is now represented by homogenous coordinates (X, Y, Z, W), where

$$x = \frac{X}{W}, y = \frac{Y}{W}, z = \frac{Z}{W}, W \neq 0$$

This construction is usually visualized as central projection with centre of projection being point with coordinates $(0,0,0,1)$ to hyperplane given by equation $W = 1$. We add weight w_i to each control point V_i such that $V_i^w = (w_i V_i, w_i) = (w_i X_i, w_i Y_i, w_i Z_i, w_i)$.

Now any integral curve of degree n can be defined by equation as in [1]

$$r^w(u) = R(u) = \sum_{i=0}^{n} T_i(u) V_i^w$$

where $T_i(u)$ are blending functions. We project this curve integral $R(u)$ to the hyperplane $W = 1$ and there we can express curve $r(u)$ as

$$r(u) = \frac{R(u)}{W(u)} = \frac{\sum_{i=0}^{n} T_i(u) w_i V_i}{\sum_{i=0}^{n} T_i(u)\, w_i} = \sum_{i=0}^{n} R_i(u) V_i$$

where blending functions $R_i(u)$ defined as

$$R_i(u) = \frac{T_i(u) w_i}{\sum_{j=0}^{n} T_j(u) w_j}$$

are now rational polynomials and hence we call curve $r(u)$ rational curve.

1.3.1. Rational Bézier curve

Rational Bézier curve is expansion of Bézier curve. For given control points $V_0, \dots, V_n$ and their weights $w_i \geq 0$, $i = 0, \dots, n$, we define rational Bézier curve of degree n as

$$c(t) = \frac{C(t)}{W(t)} = \frac{\sum_{i=0}^{n} B_i^n(t) w_i V_i}{\sum_{i=0}^{n} B_i^n(t)\, w_i} = \sum_{i=0}^{n} R_i^n(t) V_i$$

$$R_i^n(t) = \frac{B_i^n(t) w_i}{\sum_{j=0}^{n} B_j^n(t) w_j}$$

where $t \in [0, 1]$ and $B_i^n(t)$ are Bernstein polynomials of degree n. Rational blending functions $R_i^n(t)$ are generalisation of Bernstein polynomials so they have same properties and if all w_i are equal, we obtain Bernstein polynomials. (Figure 1-11)

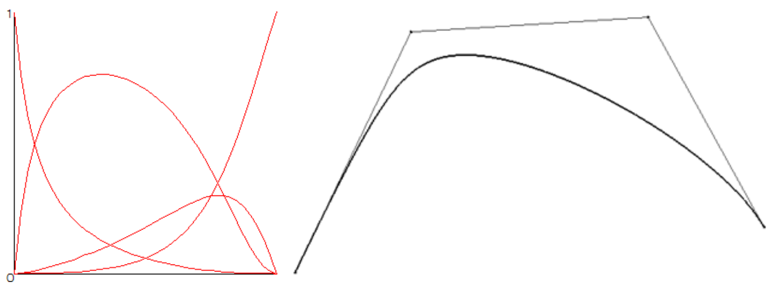


Figure 1-11 Rational Bernstein polynomials and Bézier arc of degree $n = 3$ with weights $w_0 = w_2 = w_3 = 1, w_1 = 4$

Rational Bézier curve has properties as integral Bézier curve apart from these [9]:

- Convex hull property: If $w_i \geq 0, i = 0, \dots, n$ then every point of a rational Bézier curve lies inside the convex hull of its control polygon. If even a single weights is negative, this property does not hold.
- Invariance under projective transformation:

$$P(c(t)) = P\left(\sum_{i=0}^{n} R_i^n(t) V_i\right) = \sum_{i=0}^{n} R_i^n(t) P(V_i)$$

  where P is an projective transformation. This is stronger condition than invariance under affine transformation [3].

- Derivative:

$$\frac{d}{dt} c(t) = \frac{C'(t)}{W(t)} - c(t) \frac{W'(t)}{W(t)}$$

  Tangents at end points are

$$\frac{d}{dt} c(0) = n \frac{w_1}{w_0} (V_1 - V_0)$$

$$\frac{d}{dt} c(1) = n \frac{w_{n-1}}{w_n} (V_n - V_{n-1})$$

- Degree elevation and reduction: Same as for Bézier arc, we can represent curve of degree n as curve of degree $n+1$ or approximate said curve with curve of degree $n-1$. Algorithms for obtaining new control points ate the same, but they work with homogenous coordinates.

Slight modification of de Casteljau algorithm is used for enumerating rational Bézier curve. We can use standard algorithm but now each point considered in algorithm have expanded affine coordinates. Final point on curve is then obtained by projecting calculated point to hyperplane $W = 1$

$$v_0^n(t) = (X, Y, Z, W) \longrightarrow \left(\frac{X}{W}, \frac{Y}{W}, \frac{Z}{W}\right) = c(t)$$

Another approach is to project each point obtained in algorithm right after its enumeration to hyperplane $W = 1$.

By modifying weights we can control how much the curve is pulled toward to control points. Now assume rational Bézier curve with all weights w_i having same non-zero value. Resulting curve has same shape as integral curve regardless of weight value. Increasing ratio of one weight w_i relative to others pulls the curve closer to control vertex V_i and away from other control vertices.

Important property of rational Bézier curve is ability to represent conic sections. Given control vertices V_0, V_1 and V_2 with theirs non-negative weights w_0, w_1 and w_2, quadratic rational Bézier curve is defined by

$$c(t) = \frac{\sum_{i=0}^{2} B_i^2(t) w_i V_i}{\sum_{j=0}^{2} B_j^2(t) w_j} = \frac{(1-t)^2 w_0 V_0 + 2t(1-t) w_1 V_1 + t^2 w_2 V_2}{(1-t)^2 w_0 + 2t(1-t) w_1 + t^2 w_2}$$

Type of conic section is defined by denominator

$$W(t) = (w_0 - 2w_1 + w_2)u^2 + 2(w_1 - w_0)t + w_0$$

Discriminant for this quadratic equation is $w_1^2 - w_2 w_0$ so if

- $w_1^2 - w_2 w_0 < 0$ then type of conic section is ellipse
- $w_1^2 - w_2 w_0 = 0$ then type of conic section is parabola
- $w_1^2 - w_2 w_0 > 0$ then type of conic section is hyperbola

If we choose weights $w_0 = w_2 = 1$ then previous conditions are simplified to:

- $w_1^2 < 1 \Rightarrow w_1 \in (-1,1) \Rightarrow w_1 \in (0,1)$ conic section is ellipse
- $w_1^2 = 1 \Rightarrow w_1 = \pm 1 \Rightarrow w_1 = 1$ conic section is parabola
- $w_1^2 > 1 \Rightarrow w_1 \in (-\infty, 1) \cup (1, \infty) \Rightarrow w_1 > 1$ conic section is hyperbola

It is more convenient for designers to choose position of point on curve for some parametric value, for example $t = 1/2$, rather than weight w_1. This point S is called shoulder point.

$$S = c\left(\frac{1}{2}\right) = \frac{\frac{1}{2}(V_0 + V_2) + w_1 V_1}{1 + w_1} = \frac{1}{1 + w_1} M + \frac{w_1}{1 + w_1} V_1 = (1 - s)M + sV_1$$

where M is midpoint of control points V_0 and V_2 and parameter $s = \frac{w_1}{1+w_1}$. By moving point on curve S along line segment MV_1 designer change value of s and obtain line segment V_0V_2 for $s = 0$, ellipse for $s \in (0, 1/2)$, parabola for $s = 1/2$ and hyperbola for $s \in (1/2, 1)$.

1.3.2. Rational Bézier spline

Rational Bézier spline for control points $V_0, \dots, V_k, k \geq 2$ and their weights $w_0, \dots, w_k \geq 0$ is defined as

$$s_i(u) = \frac{\sum_{j=0}^{n} B_j^n(t) w_{i,j} V_{i,j}}{\sum_{j=0}^{n} B_j^n(t) w_{i,j}}, i = 0, \dots, k-1$$

where $u \in [a, b], t = \frac{u-u_i}{u_{i+1}-u_i}, V_{i,j}$ are control points, $w_{i,j}$ are their weights and $B_j^n(t)$ is j-th Bernstein polynomial of degree n. New control points $V_{i,j}$ are again derived from control points $V_i, i = 0, \dots, k$ depending on type of spline, but using homogenous coordinates.

We use de Casteljau algorithm for rational Bézier arc in order to evaluate point on spline.

1.3.3. Rational B-splines curve - NURBS

"Rational B-splines provide a single precise mathematical form capable of representing the common analytical shapes – lines, planes, conic curves including circles, free form curves, quadric and sculptured surfaces – that are used in computer graphics and computer aided design." [2]

For given control points $V_0, \dots, V_n$, their weights $w_i \geq 0, i = 0, \dots, n$ and knot vector $U = \{u_i\}_{i=0}^m, m = n + k + 1$, we define rational B-Spline curve of degree k

$$s(u) = \frac{S(u)}{W(u)} = \frac{\sum_{i=0}^n N_i^k(U,u) w_i V_i}{\sum_{j=0}^n N_j^k(U,u)\, w_j} = \sum_{i=0}^n R_i^k(u) V_i$$

$$R_i^k(u) = \frac{N_i^k(U,u) w_i}{\sum_{j=0}^n N_j^k(U,u) w_j}$$

where $u \in [u_k, u_{m-k})$ and $R_i^k(U,u)$ is i-th rational B-spline basis function of degree k. Rational B-spline basis functions $R_i^k(U,u)$ are generalisation of B-spline basis and thus inherit nearly all properties of non-rational functions. If all weights w_i have same value then rational B-spline basis functions $R_i^k(U.u)$ become B-spline basis functions $N_i^k(U,u)$. Periodicity for uniform knot vector is preserved only if all weights have same value. (Figure 1-12)

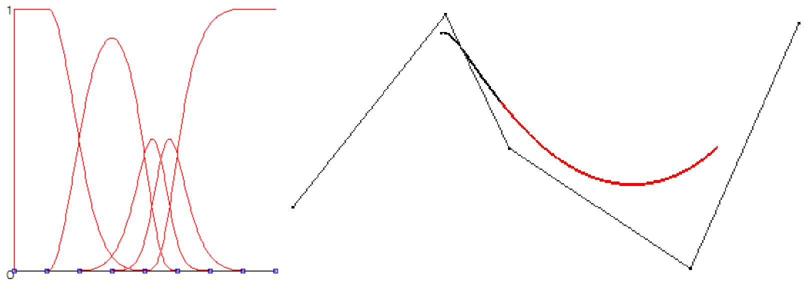


Figure 1-12 Rational B-spline polynomials and NURBS spline of degree $n = 3$ for uniform knot vector and weights $w_i = \{1,\ 4, 1, 1, 4\}$

NURBS Spline has same properties as integral B-spline, apart from following [9]:

- Convex hull property: If $w_i \geq 0, i = 0, \dots, n$ then every point of NURBS curve lies inside convex hull defined by control polygon.
- Invariance under projective transformation:

$$P\big(s(u)\big) = P\left(\sum_{i=0}^k N_i^k(U,u) V_i\right) = \sum_{i=0}^k N_i^k(U,u) P(V_i)$$

where P is an projective transformation. This is stronger condition than invariance under affine transformation [3].

- Derivative:

$$\frac{d}{dt}s(u) = \frac{S'(u)}{W(u)} - s(u)\frac{W'(u)}{W(u)}$$

Derivatives at ending points are given by equations [2]

$$s'(u_k) = (k-1)\frac{w_1}{w_0}(V_1 - V_0)$$

$$s'(u_{m-k}) = (k-1)\frac{w_{n-1}}{w_n}(V_n - V_{n-1})$$

so once again tangent vectors at ending points are parallel to first and last control polygon span.

Algorithms for knot insertion and knot removal are almost the same, but now they work with homogenous coordinates. We use rational de Boor algorithm for enumeration of rational B-spline curve. Similarly to modification of de Casteljau algorithm we choose one of two approaches. First one is to compute all intermediate results in homogenous coordinates and in second one we project these results into hyperplane $W = 1$ after each step.

With rational B-spline curve we can also represent conic sections. We can represent all types of conic sections but with NURBS circle arc construction becomes simpler. We choose NURBS spline with degree 2, because conic sections are described by quadratic equations.

With three vertices $V_0V_1V_2$ forming isosceles triangle and weights $w_0 = w_2 = 1$ and $w_1 = \cos\theta$ where θ is angle between base and side of triangle $V_0V_1V_2$. Obtained circular arc subtend angle 2θ of a full circle. This arc represents one segment of NURBS spline with open knot vector $U = \{0,0,0,1,1,1\}$. Full circle is created by duplicating this procedure to obtain for example three $120°$ or four $90°$ segments. (Figure 1-13)

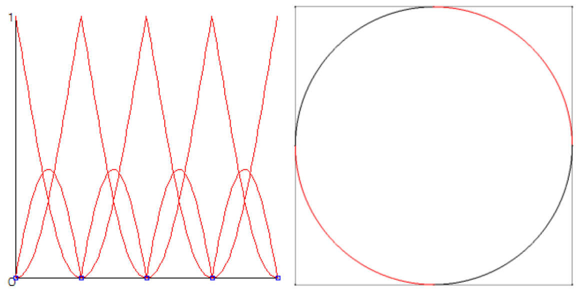


Figure 1-13 Rational B-spline polynomials and NURBS spline of degree $n = 2$ for knot vector $U = \{0,0,0,1,1,2,2,3,3,4,4,4\}$ and weights $w_0 = \{1, \sqrt{2}/2, 1, \sqrt{2}/2, 1, \sqrt{2}/2, 1, \sqrt{2}/2, 1\}$

1.4. Surfaces

“Surfaces are used in almost all branches of modern technologies. Shapes of things we use every day like clothes, cell phones, cars and also buildings are usually designed using some form of computer aided design. We can find analytical form for many surfaces, but these forms often does not exist for more complex surfaces like airplane fuselage and car body.“ [2] Designers create these from simpler surfaces – patches composed of points and curves. The patches are joined together with different continuity conditions and called piecewise surfaces.

“A vector-valued parametric representation is used because it is axis independent, avoids infinite slope values with respect to some arbitrary axis system, allows the unambiguous representation of multivalued surfaces or space functions, facilitates the representation of surfaces in homogeneous coordinates and is compatible with the use of the three-dimensional homogeneous coordinate transformations.“ [2]

If we want to represent surface by parametric representation, we have to use two parameters, u and v. We call such surface bi-parametric and its equations are

$$x = x(u,v), y = y(u,v), z = z(u.v)$$

where $u, v \in U \times V$. If we fix value of u or v, we obtain so-called v-isoparametric or u-isoparametric curves on the surface. If isoparametric curve has fixed parameter of maximum (minimum) value, it forms edge of the surface. Fixing both parametric values results in a point on the surface.

1.4.1. Swept surfaces

“A swept surface is generated when a curve is parametrically translated or rotated. In CAD, a surface is represented by series of curves, which are parametrically generated at various instances.” [11]

While translating or rotating profile curve, we sample curve’s points position. Since both rotation and translation are affine transformations, surface of this type is called an affine surface. Here we will discuss surfaces of revolution and extruded surfaces.

1.4.1.1. Surfaces of revolution

Assume a curve $c(u) = C_0(u)$ in space E^3. If we rotate curve $C_0(u)$ by angle $v \in \langle 0,2\pi\rangle$ around axis of rotation and sample position of $C_0(u)$ while rotating we obtain surface of revolution. We add a condition for planar curve $C_0(u) \subset \alpha$, plane α cannot be perpendicular to axis of rotation. u-isocurve is original curve rotated by an fixed angle v and v-isocurve is a parallel circle in plane $C_0(u) \subset \beta$ perpendicular to axis of rotation. We can use quaternions to simplify rotating around arbitrary axis.

“Quaternions are four vectors (this is why they were given this name), and inherit vector operations including addition, scalar multiplication, dot product and norm, but their multiplication is defined specially, in a way somehow similar to arithmetic of complex numbers, because quaternions can also be interpreted as a generalization of the complex numbers with s as the real part and x, y, z as the imaginary part. Denoting the imaginary axes by i, j and k yields:” [4]

$$q = s \cdot 1 + xi + yj + zk = [s, x, y, z] = [s, \vec{w}]$$

where q is quaternion, $\vec{w}$ is vector (x, y, z) and basis elements i, j and k satisfy equations

$$i^2 = j^2 = k^2 = ijk = -1$$

$$ij = -ji = k$$

$$jk = -kj = i$$

$$ki = -ik = j$$

Operation on quaternions are defined as in [4]

$$q_1 + q_2 = [s_1, \overrightarrow{w_1}] + [s_2, \overrightarrow{w_2}] = [s_1 + s_2, \overrightarrow{w_1} + \overrightarrow{w_2}]$$

$$\lambda q = \lambda[s, \vec{w}] = [\lambda s, \lambda \vec{w}]$$

$$q_1 \cdot q_2 = [s_1, \overrightarrow{w_1}] \cdot [s_2, \overrightarrow{w_2}] = [s_1 s_2 - \overrightarrow{w_1} \cdot \overrightarrow{w_2}, s_1 \overrightarrow{w_2} + s_2 \overrightarrow{w_1} - \overrightarrow{w_1} \times \overrightarrow{w_2}]$$

$$\langle q_1, q_2 \rangle = \langle [s_1, \overrightarrow{w_1}], [s_2, \overrightarrow{w_2}] \rangle = s_1 s_2 + \overrightarrow{w_1} \cdot \overrightarrow{w_2}$$

$$\|q\|^2 = \sqrt{s^2 + x^2 + y^2 + z^2}$$

Inverse quaternion to quaternion q^{-1} is defined as

$$q^{-1} = \frac{[s, \overrightarrow{-w}]}{\|q\|^2}$$

In order to rotate 3D vector $\vec{a}$ around axis $\vec{d}$ by an angle α to new orientation $\vec{b}$, we first extend this vector by an $s = 0$ to create a quaternion $[0, \vec{u}]$. We use properties of quaternion multiplication to obtain $[0, \vec{b}]$ by equation

$$[0, \vec{a}] = q \cdot [0, \vec{b}] \cdot q^{-1}, q = \left[\cos\frac{\alpha}{2}, \sin\frac{\alpha}{2} \cdot \vec{d}\,\right]$$

"Our ultimate objective is to move an object from an orientation represented by q_1 to new orientation of q_2 by an even and uniform motion. If linear interpolation is used to generate the path of orientations between q_1 and q_2, then the angles of the subsequent quaternions will not be constant." [4]

"Instead of linear interpolation, a non-linear interpolation must be found that guarantees the constant angle between the subsequent interpolated quaternions. Spherical interpolation obviously meets this requirement, where the interpolated quaternions are selected uniformly from the arc between $q_1$and q_2. If $q_1$and q_2 are unit quaternions, then all the interpolated quaternions will also be of unit length. Unit size quaternions can be regarded as unit-size four-vectors which correspond to a 4D unit-radius sphere. An appropriate interpolation method must generate the great arc between $q_1$and q_2, and as can easily be shown, this great arc has the following form:

$$q(t) = \frac{\sin(1-t)\theta}{\sin\theta} \cdot q_1 + \frac{\sin t\theta}{\sin\theta} \cdot q_2$$

Where angle $\cos\theta = \langle q_1, q_2 \rangle$. " [4]

Now we can express surface of revolution of $c(u)$ curve around axis $\vec{d}$ by an angle v as

$$[0, S(u, v)] = q(v) \cdot [0, c(u)] \cdot q(v)^{-1}$$

where $u, v \in [a, b] \times [r, s] \subseteq [a, b] \times [0, 2\pi]$ and $q(v)$ is spherical interpolation between quaternions q_1 and q_2 given by equations

$$q_1 = \left[\cos\frac{r}{2}, \sin\frac{r}{2} \cdot \vec{d}\,\right], q_2 = \left[\cos\frac{s}{2}, \sin\frac{s}{2} \cdot \vec{d}\,\right]$$

Modelling of this kind of surface can be achieved by modifying shape of input curve, changing axis and angle of rotation. (Figure 1-14)

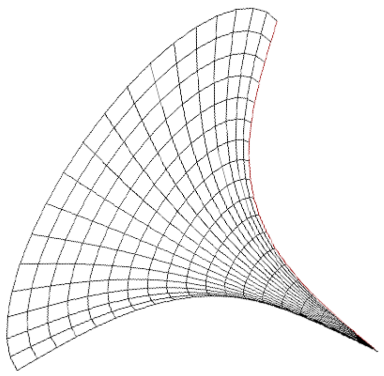


Figure 1-14 Surface of revolution

1.4.1.2. Extruded surfaces

Another type of affine surface is extruded surface. It is also called translational surface. This time we move curve point on curve $C_0(u)$ along another curve $D_0(v)$. Curves $C_0(u)$ and $D_0(v)$ are defined on intervals $[a, b]$ and $[c, d]$ respectively.

$$S(u, v) = C_0(u) + (D_0(v) - D_0(c))$$

u-isocurve is translated curve $C_0(u)$ and v-isocurve is translated curve $D_0(v)$. (Figure 1-15)

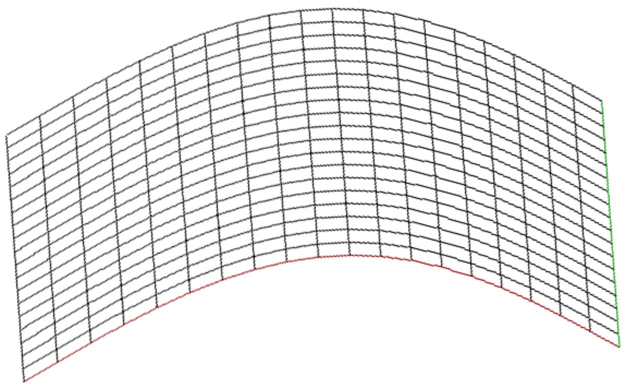


Figure 1-15 Extruded surface

1.4.2. Coons patches

This type of patch takes boundary curves and somehow interpolates them. We can divide Coons patches into categories by number of input curves and the way they are blended to form a patch. Changing shape of these surfaces is done by modifying boundary curves.

1.4.2.1. Ruled Coons patches

We are given two curves in space, $C_0(u)$ and $C_1(u)$, defined on same interval $u \in [0, 1]$. In order to create patch $S(u, v)$ we simply join corresponding points on curve with a line segment. Thus each v-isocurve is a line segment and that is why it is called ruled or linear Coons patch. We can write it down as in [1]

$$S(u, v) = C_0(u) \ + v(C_1(u) - C_0(u))$$

for $u, v \in [0, 1] \times [0, 1]$. (Figure 1-16)

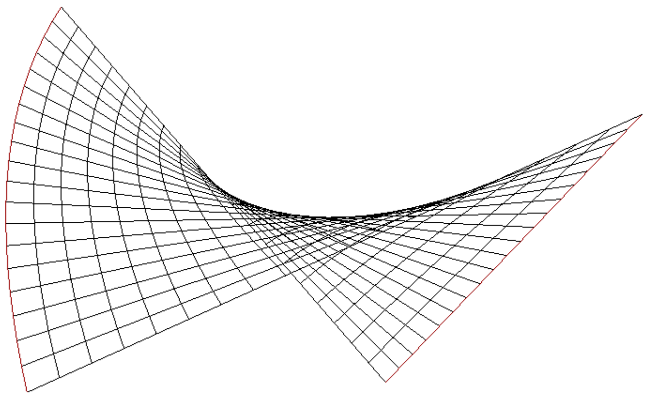


Figure 1-16 Linear Coons patch

1.4.2.2. Bilinear Coons patches

Now we are given four curves in space, $C_0(u), C_1(u)$ $D_0(v)$ and $D_1(v)$ defined on the same interval $u, v \in [0, 1]$. These curves must have common points in corners of patch (they must "meet" in corners):

$$C_i(j) = D_j(i) \qquad i, j = 0,1$$

We call this C^0 compatibility. Then we can define bilinear Coons patch as in [1]

$$S(u, v) = S_c(u, v) + S_D(u, v) - S_{CD}(u, v)$$

where

$$S_c(u, v) = (1 - v)\, C_0(u) + vC_1(u)$$

$$S_D(u, v) = (1 - u)\, D_0(v) + uD_1(v)$$

$$S_{CD}(u, v) = \begin{pmatrix} 1 - u & u \end{pmatrix} \begin{pmatrix} C_0(0) & C_0(1) \\ C_1(0) & C_1(1) \end{pmatrix} \begin{pmatrix} 1 - v \\ v \end{pmatrix}$$

for $u, v \in [0, 1] \times [0, 1]$. Sumand $S_{CD}(u, v)$ is called bilinear interpolant of corner vertices. (Figure 1-17)

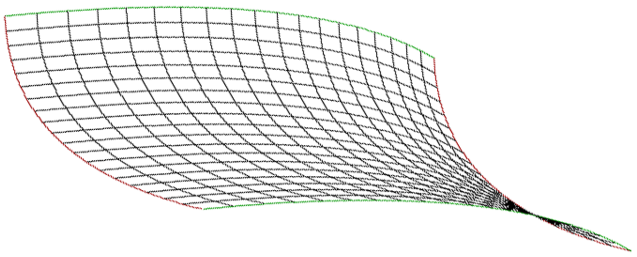


Figure 1-17 Bilinear Coons patch

1.4.2.3. Partially bicubic Coons patches

Blending functions used so far were linear, what can cause problems when joining two linear Coons patches along edges. Therefore we can choose another blending functions, which fulfil these conditions [1]: $f_1(u) + f_2(u) = 1, g_1(v) + g_2(v) = 1, f_1(0) = g_1(0) = 1$ and $f_2(1) = g_2(1) = 0$. In this work, we have chosen Hermite polynomials $H_0^3(t)$ on the position of $f_1(u)$ and $g_1(v)$ and $H_3^3(t)$ on the position of $f_2(u)$ and $g_2(v)$. Now patch can be described as

$$S(u,v) = S_c(u,v) + S_D(u,v) - S_{CD}(u,v)$$

where

$$S_c(u,v) = H_0^3(v)\,C_0(u) + H_3^3(v)C_1(u)$$

$$S_D(u,v) = H_0^3(u)D_0(v) + H_3^3(u)D_1(v)$$

$$S_{CD}(u,v) = \begin{pmatrix} H_0^3(u) & H_3^3(u) \end{pmatrix} \begin{pmatrix} C_0(0) & C_0(1) \\ C_1(0) & C_1(1) \end{pmatrix} \begin{pmatrix} H_0^3(v) \\ H_3^3(v) \end{pmatrix}$$

for $u, v \in [0,1] \times [0,1]$. (Figure 1-18)

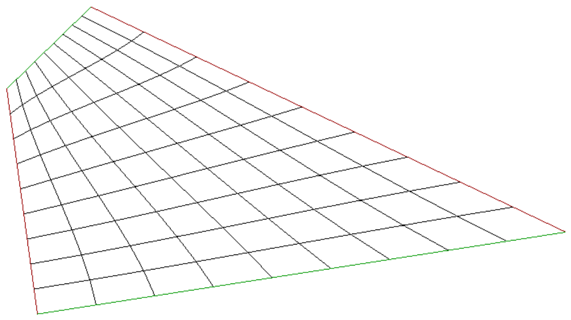


Figure 1-18 Partialy bicubic Coons patch

1.4.2.4. Bicubic Coons patch

In this case each boundary curve is Hermite cubic curve and we use all four Hermite cubic polynomials as blending functions. Curves have to be C^0 compatible. Now we express patch $S(u,v)$ as in [10]

$$S(u,v) = \begin{bmatrix} H_0^3(u) & H_1^3(u) & H_2^3(u) & H_3^3(u) \end{bmatrix} \begin{bmatrix} C_0(0) & \overrightarrow{D_0}(0) & \overrightarrow{D_0}(1) & C_1(0) \\ \overrightarrow{C_0}(0) & \overrightarrow{t_{00}} & \overrightarrow{t_{10}} & \overrightarrow{C_1}(0) \\ \overrightarrow{C_0}(1) & \overrightarrow{t_{01}} & \overrightarrow{t_{11}} & \overrightarrow{C_1}(1) \\ C_0(1) & \overrightarrow{D_1}(0) & \overrightarrow{D_1}(1) & C_1(1) \end{bmatrix} \begin{bmatrix} H_0^3(v) \\ H_1^3(v) \\ H_2^3(v) \\ H_3^3(v) \end{bmatrix}$$

for $u, v \in [0,1] \times [0,1]$. (Figure 1-19)

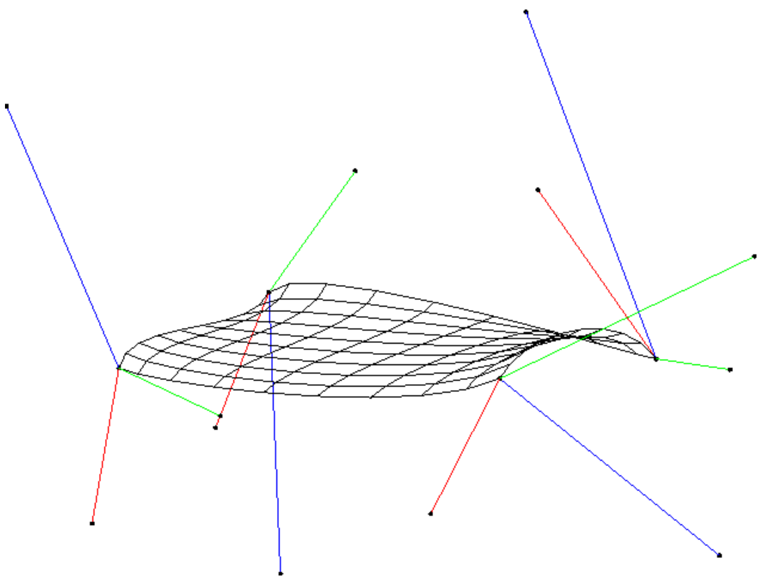


Figure 1-19 Bicubic Coons patch

Isocurves for both parametric directions are cubic Hermite arcs. There are three types of data in 4x4 matrix. First of them are corner vertices, next are tangent vectors in patch's corner points in u and v parametric direction, respectively and last one is composed of mixed derivatives called twist vectors. They are zero vectors for Ferguson patch and non-zero for general bicubic Coons patch. If they are not given, then there are many ways to derive them from boundary curves in order to join patches to form a surface with C^2 continuity.

The simplest of them is to create bilinear patch, where corners are $C_0(0), C_0(1), C_1(0)$ and $C_1(1)$ and boundary curves are line segments. Then twist vector are then defined as

$$\overrightarrow{t_{ij}} = \big(C_0(0) - C_0(1)\big) + \big(C_1(1) - C_1(0)\big), i, j = 0,1$$

We can obtain twist vectors from Adini method [5]. In case of single patch, we create bilinear Coons patch with boundary curves same as for bicubic. Now equation for twist vector is

$$\overrightarrow{t_{ij}} = \left(\overrightarrow{D_1}(j) - \overrightarrow{D_0}(j)\right) + \left(\overrightarrow{C_1}(i) - \overrightarrow{C_0}(i)\right) - \big(C_0(0) - C_0(1)\big) + \big(C_1(1) - C_1(0)\big), i, j = 0,1$$

Another methods can be found in [3] and [5].

1.4.3. Tensor product surfaces

"The product of a column vector and a row vector is an example of the mathematical operation of tensor-product:

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} \otimes [b_0 \quad b_1 \quad b_2 \quad b_3] = \begin{bmatrix} a_0b_0 & a_0b_1 & a_0b_2 & a_0b_3 \\ a_1b_0 & a_1b_1 & a_1b_2 & a_1b_3 \\ a_2b_0 & a_2b_1 & a_2b_2 & a_2b_3 \end{bmatrix}$$

A tensor-product surface $S(u, v)$ is one whose blending functions are products of pairs of univariate blending functions:" [7]

$$S(u, v) = \sum_{i=0}^{k} \sum_{j=0}^{l} P_i^m(u) P_j^n(v) V_{ij} = \sum_{i=0}^{k} \sum_{j=0}^{l} S_{ij}(u, v) V_{ij}$$

$$S_{ij}(u,v) = P_i^m(u)P_j^n(v)$$

where $V_{ij}, i = 0, \dots, k, j = 0, \dots, l$ are control points, $P_i^m(u), P_j^n(v)$ are univariate blending functions of degree m and n, respectively, and $S_{ij}(u,v)$ is bivariate blending function with degree m and n in respective parametric directions. Control points are organized topologically into a rectangular array consisting of k rows and l columns. Two control points V_{ab} and V_{cd} are considered to be neighbours if $|a - c| \leq 1$ and $b = d$ or $|b - d| \leq 1$ and $a = c$. Object formed by control vertices and all line segments, which are joining two neighbour points, is called control net.

1.4.3.1. Tensor product Bézier surface

For given control points $V_{ij}, i = 0, \dots, m, j = 0, \dots, n$ we define tensor product Bézier surface $S(u,v)$ as in [9]

$$S(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} B_i^m(u)B_j^n(v)V_{ij}$$

where $u, v \in [0,1] \times [0,1]$, m and n are degrees of said patch in u and v respective parametric directions and $B_i^m(u)$ and $B_i^n(v)$ are Bernstein polynomials of degree m and n, respectively. (Figure 1-20)

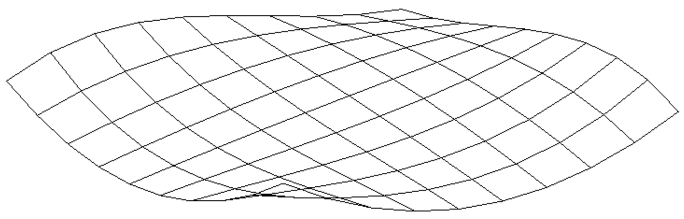


Figure 1-20 Tensor product Bézier patch

Properties of tensor product Bézier surface are similar to those of Bézier curve [9]:

- Bézier surface follows the shape of the control net $V_{ij}, i = 0, \dots, m, j = 0, \dots, n$.
- Convex hull property: Due to

$$B_i^m(u)B_j^n(v) \geq 0$$

$$\sum_{i=0}^{m}\sum_{j=0}^{n} B_i^m(u)B_j^n(v) = 1$$

for $u, v \in [0,1] \times [0,1]$, surface points are located in the convex hull formed by control net.

- Isocurves [10]: Isocurves for both parameters are Bézier arcs of degree m and n, respectively. u-isocurves and v-isocurves are obtained by equations

$$S(u,v^*) = \sum_{i=0}^{m} B_i^n(u) \sum_{j=0}^{n} B_j^n(v^*)V_{ij}$$

$$S(u^*, v) = \sum_{j=0}^{n} B_j^n(v) \sum_{i=0}^{m} B_i^m(u^*) V_{ij}$$

where u^* and v^* are fixed parameter, respectively. Especially all boundary curves are obtained by evaluating $S(u, 0), S(u, 1), S(0, v)$ and $S(1, v)$. Control polygons for these Bézier arcs are subsets of control net. Because of this, patch interpolates all four corner control points.

- Invariance under parameter transformation:

$$\sum_{i=0}^{m}\sum_{j=0}^{n} B_i^m(u) B_j^n(v) V_{ij} = \sum_{i=0}^{m}\sum_{j=0}^{n} B_i^m\left(\frac{s-a}{b-a}\right) B_j^n\left(\frac{t-c}{d-c}\right) V_{ij}$$

where $u, v \in [0, 1] \times [0, 1]$ and $s, t \in (a, b) \times (c, d)$.

- Pseudo local control: As with Bézier arc, tensor product Bézier surface cannot be locally modified. Change of position of point V_{ij} deforms the whole patch and the largest deviation occurs at $u = i/m$ and $v = j/n$.
- Invariance under affine transformation:

$$T(S(u, v)) = T\left(\sum_{i=0}^{m}\sum_{j=0}^{n} B_i^m(u) B_j^n(v) V_{ij}\right) = \sum_{i=0}^{n}\sum_{j=0}^{n} B_i^m(u) B_j^n(v) T(V_{ij})$$

where T is an affine transformation.

- Partial derivative [10]: We define partial derivatives by equation

$$\frac{\delta^{k+l} S(u, v)}{\delta u^k \delta v^l} = m \ldots (m-k+1) n \ldots . (n-l+1) \sum_{i=0}^{m-k} B_i^{m-k}(u) \sum_{j=0}^{n-l} B_j^{n-l}(v)\, \Delta^{k,l}\, V_{ij}$$

Where $\Delta^{k,l}\, V_{ij}$ is a vector defined recursively by following relations

$$\Delta^{0,0}\, V_{ij} = V_{ij}$$

$$\Delta^{1,0}\, V_{ij} = V_{i+1j} - V_{ij}$$

$$\Delta^{0,1}\, V_{ij} = V_{ij+1} - V_{ij}$$

$$\Delta^{k,l}\, V_{ij} = \Delta^{1,0}\left(\Delta^{k-1,l}\, V_{ij}\right), k > 0$$

$$\Delta^{k,l}\, V_{ij} = \Delta^{0,1}\left(\Delta^{k,l-1}\, V_{ij}\right), l > 0$$

- Degree elevation and reduction [10]: We can use similar algorithms as for curves. First one difference is that we have to specify degree to be elevated or decreased and second one is that we have to use Bézier arc degree elevation for each row (column) of control vertices.

- Variation diminishing property: “The surface does not exhibit the variation-diminishing property. The variation-diminishing property for bivariant surfaces is both undefined and unknown.” [2]

Enumerating tensor product Bézier surface can be done in two ways. First of them is to use de Casteljau algorithm for one fixed parameter, thus creating set of Bézier arcs for second parameter and then fixing second parameter and use de Casteljau algorithm for obtained points.

Second one is to use bilinear interpolation [10]. Let

$$b_{ij}^{0}(u,v) = V_{ij}, i = 0, \dots, m, j = 0, \dots, n$$

We define $b_{ij}^{l}(u,v)$ as

$$\begin{aligned} b_{ij}^{l}(u,v) = (1-v)\left((1-u)b_{ij}^{l-1}(u,v) + ub_{i+1j}^{l-1}(u,v)\right) \\ + v\left((1-u)b_{ij+1}^{l-1}(u,v) + ub_{i+1j+1}^{l-1}(u,v)\right) \end{aligned}$$

for $l = 1, \dots, \min(m,n)$, $i = 0, \dots, m-l$ and $j = 0, \dots, n-l$. If $m = n$ then the result is point on patch

$$b_{00}^{m}(u,v) = S(u,v)$$

If this is not the case, then we are left with $|m-n|+1$ points. We use de Casteljau algorithm with these points as input and parameter, in which direction surface has higher degree, to calculate point on patch.

Only way to change shape of patch is to change position of control vertices.

1.4.3.2. Tensor product B-spline surface

We can apply the same idea for creating patch from B-spline curves. Patch is defined as in [2]

$$S(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} N_i^k(U,u)N_j^l(V,v)V_{ij}$$

Where $u, v \in [u_k, u_{m+1}) \times [u_l, u_{n+1}), V_{ij}, i = 0, \dots, m, j = 0, \dots, n$ are control vertices, $U = \{u_i\}_{i=0}^{m+k+1}$ and $V = \{v_i\}_{i=0}^{n+l+1}$ are knot vectors with maximum possible knot multiplicity of $m+1$ and $n+1$, respectively, and $N_i^k(U,u)$ and $N_j^l(V,v)$ are B-Spline basis functions of degree k and l, respectively. (Figure 1-21)

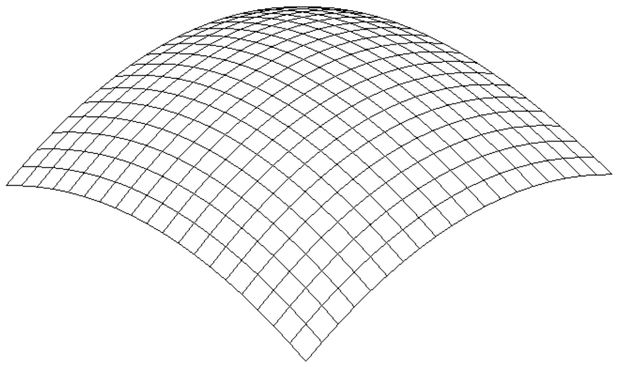


Figure 1-21 Tensor product B-spline patch

"Because the B-spline basis is used both to describe the boundary curves and to blend the interior of the surface, several properties of the B-spline surface are immediately known:" [2]

- B-spline surface follows the shape of the control net $V_{ij}, i = 0, \dots, m, j = 0, \dots, n$.
- Surface domain is $u, v \in [u_k, u_{m-k}) \times [u_l, u_{n-l})$.
- Local modelling: Patch segment corresponding to interval $u, v \in [u_i, u_{i+1}) \times \left[u_j, u_{j+1}\right)$ for $k \leq i \leq m - k - 1, l \leq j \leq n - l - 1$ is designated by $(k+1) \cdot (l+1)$ control vertices $V_{i-kj-l}, \dots, V_{ij}$.
- Convex hull property: Every point of patch is enclosed inside the convex hull formed by control net because
$$N_i^k(U,u) N_j^l(V,v) \geq 0$$
$$\sum_{i=0}^{m} \sum_{j=0}^{n} N_i^k(U,u) N_j^l(V,v) = 1$$
for $u, v \in [u_k, u_{m+1}) \times [u_l, u_{n+1})$.
- Isocurves: u-isocurves and v-isocurves are obtained by equations
$$S(u, v^*) = \sum_{i=0}^{m} N_i^k(U,u) \sum_{j=0}^{n} N_j^l(V, v^*) V_{ij}$$
$$S(u^*, v) = \sum_{j=0}^{n} N_j^l(V,v) \sum_{i=0}^{m} N_i^k(U, u^*) V_{ij}$$
where v^* and u^* are fixed parameter, respectively, hence isocurves are B-spline curves of degree k and l, respectively. Boundary curves can be obtained by evaluating $S(u,0), S(u,1), S(0,v)$ and $S(1,v)$. Subset of control net is control polygon for these curves.
- Invariance under affine transformation:
$$T(S(u,v)) = T\left(\sum_{i=0}^{m} \sum_{j=0}^{n} N_i^k(U,u) N_j^l(V,v)\, V_{ij}\right) = \sum_{i=0}^{m} \sum_{j=0}^{n} N_i^k(U,u) N_j^l(V,v) T(V_{ij})$$
where T is an affine transformation.
- Partial derivative [9]: We define partial derivatives by equation

$$\frac{\delta^{p+q} S(u,v)}{\delta u^p \delta v^q} = m \dots (m-p+1) n \dots . (n-q+1) \sum_{i=0}^{k-p} N_i^{k-p}(U,u) \sum_{j=0}^{l-q} N_j^{l-q}(V,v)\, \Delta^{p,q}\, V_{ij}$$

where $\Delta^{p,q}\, V_{ij}$ is vector defined recursively by following relations

$$\Delta^{0,0}\, V_{ij} = V_{ij}$$

$$\Delta^{1,0}\, V_{ij} = \frac{V_{i+1j} - V_{ij}}{u_{i+k+1} - u_{i+1}}$$

$$\Delta^{0,1}\, V_{ij} = \frac{V_{ij+1} - V_{ij}}{v_{i+l+1} - v_{i+1}}$$

$$\Delta^{p,q}\, V_{ij} = \Delta^{1,0}\left(\Delta^{p-1,q}\, V_{ij}\right) = \frac{\Delta^{p-1,q}\, V_{i+1j} - \Delta^{p-1,q}\, V_{ij}}{u_{i+k+1} - u_{i+p}}, p > 0$$

$$\Delta^{p,q}\, V_{ij} = \Delta^{0,1}\left(\Delta^{p,q-1}\, V_{ij}\right) = \frac{\Delta^{p,q-1}\, V_{ij+1} - \Delta^{p,q-1}\, V_{ij}}{v_{i+l+1} - v_{i+q}}, q > 0$$

- Variation diminishing property: “The variation-diminishing property for bivariate surfaces is both undefined and unknown.” [2]

Boehm’s algorithm for knot insertion and knot removal algorithm can be both easily modified for patch. We need to determine which knot vector will be changed, then increase or decrease number of control points in given parametric direction and lastly compute new control points for each row (column) of control vertices.

Enumeration of tensor product B-spline patch is basically modified de Boor algorithm. First we evaluate de Boor algorithm multiple times for each row or column, depending on chosen parametric direction, of control vertices for first parameter. Obtained points are now used for one more de Boor algorithm but now for second parametric direction. Result is point on patch.

1.4.4. Rational surfaces

Tensor product surfaces described so far have been integral surfaces. A rational surface $S(u,v)$ is defined as a central projection of integral surface $S^w(u,v)$ to the hyperplane $W = 1$.

$$S^w(u,v) = \bar{S}(u,v) = \sum_{i=0}^{k} \sum_{j=0}^{l} P_i^m(u) P_j^n(v) V_{ij}^w$$

where $V_{ij}^w = \left(w_{ij}X_{ij}, w_{ij}Y_{ij}, w_{ij}Z_{ij}, w_{ij}\right), i = 0, \dots, k, j = 0, \dots, l$ are control points, $P_i^m(u)$ and $P_i^n(v)$ are univariate blending functions of degree m and n, respectively. We can express $S(u,v)$ as

$$S(u,v) = \frac{\bar{S}(u,v)}{W(u,v)} = \frac{\sum_{i=0}^{k} \sum_{j=0}^{l} P_i^m(u) P_j^n(v) w_{ij} V_{ij}}{\sum_{s=0}^{k} \sum_{t=0}^{l} P_s^m(u) P_t^n(v) w_{st}} = \sum_{i=0}^{k} \sum_{j=0}^{l} R_{ij}(u,v) V_{ij}$$

$$R_{ij}(u,v) = \frac{P_i^m(u)P_j^n(v)w_{ij}}{\sum_{s=0}^{k}\sum_{t=0}^{l} P_s^m(u)P_t^n(v)w_{st}}$$

where the $V_{ij}, i = 0, \dots, m, j = 0, \dots, n$ are control points, $w_{ij} \geq 0$ are their weights, $P_i^m(u), P_i^n(v)$ are univariate blending functions of degree m and n and $R_{ij}(u,v)$ is bivariate blending function with degree m and n in respective parametric directions. Because each $R_{ij}(u,v)$ is rational polynomial we call this kind of surface rational one.

1.4.4.1. Rational Bézier surface

Rational Bézier surface is defined by equation as in [5]

$$S(u,v) = \frac{\sum_{i=0}^{m}\sum_{j=0}^{n} B_i^m(u)B_j^n(v)w_{ij}V_{ij}}{\sum_{s=0}^{m}\sum_{t=0}^{n} B_s^m(u)B_t^n(v)\, w_{st}} = \sum_{i=0}^{m}\sum_{j=0}^{n} R_{ij}(u,v)V_{ij}$$

$$R_{ij}(u,v) = \frac{B_i^m(u)B_j^n(v)w_{ij}}{\sum_{s=0}^{m}\sum_{t=0}^{n} B_s^m(u)B_t^n(v)\, w_{st}}$$

where $u, v \in [0,1] \times [0,1]$, $V_{ij}, i = 0, \dots, m$ and $j = 0, \dots, n$, are control vertices, $w_{ij} \geq 0$ are their weights, m and n are degrees of said patch in u and v parametric direction, $B_i^m(u), B_i^n(v)$ are Bernstein polynomials of degree m and n, respectively, and $R_{ij}(u,v)$ is bivariate blending function with degree m and n in respective parametric directions.

Properties of this kind of patch are the same as those of integral Bézier surface apart from these [9]:

- Convex hull property: If $w_{ij} \geq 0, i = 0, \dots, m, j = 0, \dots, n$ then every patch point lies in the convex hull formed by control net.
- Invariance under projective transformation:

$$P(S(u,v)) = P\left(\sum_{i=0}^{m}\sum_{j=0}^{n} R_{ij}(u,v)V_{ij}\right) = \sum_{i=0}^{m}\sum_{j=0}^{n} R_{ij}(u,v)P(V_{ij})$$

  where P is an projective transformation. This is stronger condition than invariance under affine transformation [3].

- Operations of degree elevation and reduction are same as for integral Bézier surface, but they need to work with expanded affine coordinates of control points.

Enumeration of a rational Bézier patch can be done via de Casteljau algorithm or with bilinear interpolation. As with all algorithms for enumeration of rational curves and surfaces, we can either project each intermediate result to hyperplane $W = 1$ or project only final vertex to said hyperplane.

1.4.4.2. Rational B-spline Surface - NURBS

"Rational B-spline surfaces, or NURBS, are the standard for surface modelling in much of computer graphics and computer aided design. Many of the typical surface forms used in computer graphics and computer aided design, such as flat planes and quadric surfaces, e.g., cylinders, spheres, ellipsoids of revolution, as well as more complex fully sculptured surfaces, are easily and accurately represented by rational B-spline surfaces. Thus, a single surface description, with excellent local and global control, can be used in a modeller or computer aided design system rather than having to deal with multiple types of surface descriptions." [2]

We define NURBS – non-uniform rational B-spline surface as in [2]

$$S(u,v) = \frac{\sum_{i=0}^{m}\sum_{j=0}^{n} N_i^k(U,u)N_j^l(V,v)w_{ij}V_{ij}}{\sum_{s=0}^{m}\sum_{t=0}^{n} N_s^k(U,u)N_t^l(V,v)\,w_{st}} = \sum_{i=0}^{m}\sum_{j=0}^{n} R_{ij}(u,v)V_{ij}$$

$$R_{ij}(u,v) = \frac{N_i^k(U,u)N_j^l(V,v)w_{ij}}{\sum_{s=0}^{m}\sum_{t=0}^{n} N_s^k(U,u)N_t^l(V,v)\,w_{st}}$$

where $u,v \in [u_k, u_{m+1}) \times [u_l, u_{n+1}), V_{ij}, i = 0, \dots, m, j = 0, \dots, n$ are control vertices, $w_{ij} \geq 0$ are their weights, $U = \{u_i\}_{i=0}^{m+k+1}$ and $V = \{v_i\}_{i=0}^{n+l+1}$ are knot vectors with maximum possible knot multiplicity of $m+1$ and $n+1$, respectively and $N_i^k(U,u)$ and $N_j^l(V,v)$ are B-Spline basis functions of degree k and l, respectively.

Properties of this patch are same as properties of integral B-spline surface, apart from following [2]:

- Convex hull property: If $w_{ij} \geq 0, i = 0, \dots, m, j = 0, \dots, n$ then all patch point are contained in the convex hull formed by control net.
- Invariance under projective transformation:

$$P(S(u,v)) = P\left(\sum_{i=0}^{m}\sum_{j=0}^{n} R_{ij}(u,v)V_{ij}\right) = \sum_{i=0}^{m}\sum_{j=0}^{n} R_{ij}(u,v)P(V_{ij})$$

where P is an projective transformation.

Knot insertion, knot removal and evaluation process are the same as for tensor B-spline surface, but they use homogenous coordinates of control points.

Modified de Boor algorithm can be used for enumeration of NURBS patch. Algorithm works with homogenous coordinates and can either project each intermediate results to hyperplane $W = 1$ or do it only for last one – point on patch.

2. Chapter – Specification

This chapter defines requirements and specification of software application.

2.1. Goal

Main purpose of this application is visualization of various types of curves (arcs and splines) and surfaces and in some cases also blending functions. User input consists of mainly mouse events and keyboard. User can modify various parameters for curves, splines and surfaces and to some degree also control 3D environment.

Application's name is *Splines & Surfaces* (Figure 2-1). It has to be compatible with 32-bit Windows XP, 64-bit Windows 7 and newer versions. Programming language chosen for this application is C++ and development environment Microsoft Visual Studio 2012. Application has appearance of Windows forms and uses OpenGL library to display 3D graphic objects created from user's input. (Figure 2-2)

Figure 2-1 Logo

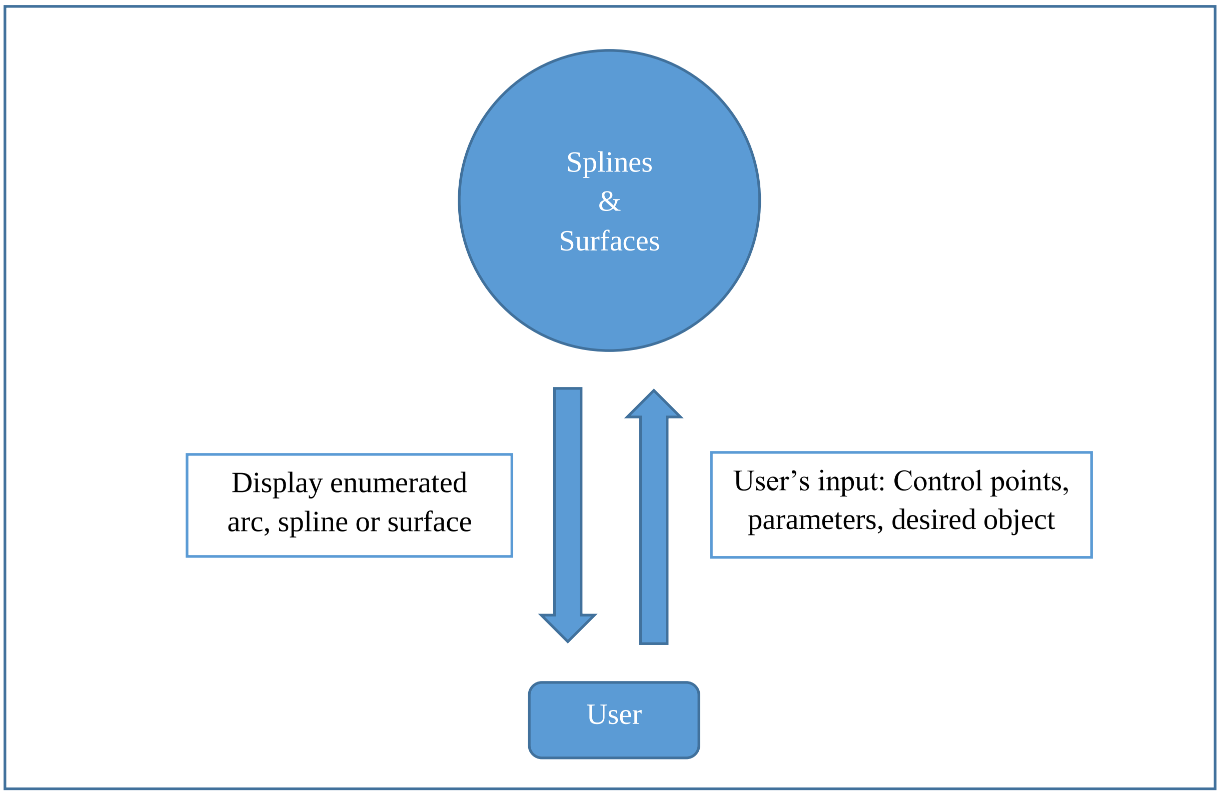

Splines
&
Surfaces
Display enumerated arc, spline or surface
User's input: Control points, parameters, desired object
User

Figure 2-2 Context diagram

2.2. Design and controls

Application has a menu strip, which provides access to groups of functions. First of them is focused on resetting application, saving current object, loading pre-created object or exit application. Next one covers scene, curve and patch options. Last three groups are used to create object form input. They are composed of arcs, splines and surfaces tab, each having multiple choices to choose from. All of them are described further below.

Directly under menu strip buttons with icons are located. These provide accelerated access to functions: New file, save, load, resetting view, top view and all curves and surfaces. Functions for save and load file include dialog, where user can choose location and name of file. If cursor hovers above one of these icons, tool tip with description of selected function is displayed.

The largest part of main window is occupied by OpenGL viewport. Here all 3D graphic objects are displayed. Smaller auxiliary window is used to show blending functions. Elements to control parameters of arc, spline and surface are located below. This form is resizable.

Another window form serves as modifier of display settings. Secondary colour for spline, colour for surface, number of cells for grid, line, curve, patch line width, curve and surface precision can be changed here along with patch display mode. It also offers option to show or hide control vertices, control polygon, grid, coordinate axes, isocurves or point on an object, additional points illustrating enumeration and options to enable point or line smoothing. User can choose between orthographic and perspective projection. Secondary colour for splines and colour for surface can be set via colour dialog. Splines' secondary colour is used for each odd segment to better illustrate segments.

Application has a separate window form to handle operations on knot vector. User can choose whether to modify U or V knot vector and type of operation. User can choose knot's index and value. Options include setting up open and universal knot vector, changing value of a single knot, increasing its multiplicity or inserting and removing knot with given value.

One more window form is used to specify control point. If offers choice whether selected control point will be modified or new control point will be added to sequence. It is also possible to change only selected attributes of control point.

In case of curves and splines, user defines control points (actual points or tangents) by first left mouse click to set x and y coordinate. Next mouse click sets z coordinate to zero if right mouse button is pressed down or to cursor position in case of left button click. If z coordinate is set and now right mouse button is pressed, sequence of control points is complete. In case of surfaces, application itself constructs control points and user can only modify them later. Control net for Bézier and B-spline patches can be initialized either with all points in a same plane, elevated with z coordinate on an arc or random. Number of points in both parametric directions can be changed and scale of control net's actual size is controllable.

To create curve or surface user chooses object type from menu or clicks on button with icon and application then enumerates chosen curve or surface and displays it inside OpenGL viewport. User can change values of parameters modifying shape of curve via numeric boxes. Spline type can be chosen from a list. For some objects blending functions are drawn in auxiliary window. Each segment of spline is drawn with alternating colour. This colour can be changed in display settings. In case of curve, point corresponding to highlight parameter is highlighted and in case of patches it is also highlighted, and isocurves for given highlight parameter are drawn with different colour.

If user wants to change position of any control points, he has to first select a control point. He has to hover mouse cursor around that point and if it is close enough, chosen point is highlighted. Then he can press left mouse button and sets up new coordinates by the same way as in case of a new point, but pressing right mouse button causes that point's x and y coordinate will not change. During either creating new point or moving existing one, mouse cursor creates virtual point. Application shows coordinates of this virtual point.

If application is not in state of defining new point, either x and y or z coordinate, pressing down right mouse button while inside OpenGL viewport opens context menu with options to add new point either at the end or begging of already defined sequence of control points, to specify vertex (opens new window form), to delete selected control point, to change multiplicity or weight of control point or to check one of these values.

In case of surfaces that require formal expressions, user can enter them into text boxes. Text boxes are labelled according to their purpose and if user hovers mouse cursors above any text box, tool tip with description of input for selected text box appears. Generally, expressions for curves have to define points in space, so each of them must contain separate expressions for x, y and z coordinate. Intervals for u, v (for given expressions) and rotation and consist of two separate numbers each. Text box that defines axis of rotation contains three separate numbers.

User can zoom in and out using mouse wheel and rotate view by pressing left mouse button while no point is selected in OpenGL viewport. Pressing arrows causes view to move in desired direction. View rotation is controlled by mouse move and new viewing angle can be accepted by pressing left mouse button once more or by pressing the right mouse button.

Blending functions are displayed in auxiliary window. If spline or patch uses knot vectors, actual knots are shown. User can choose between U or V knot vector and two operations. First one serves for knot value modification. User selects a knot by left mouse click in vicinity of that knot and by next left click defines new value for that knot. Second operation is intended for defining highlight parameter by left mouse click. In order to show form with knot vector options, user have to make right mouse click inside this window.

Application shows error message box every time an error is detected while initializing or working with graphic object. All those error are listed on the pageList 1 Error messages 79. Besides these, application also forbids action if it is not suitable or would cause an error (e.g. deleting point via context menu when no points is selected). Should any event like this happen, message box with error description is shown.

The User input for 3D window is summed up in the table of uses cases on the page 76.

2.3. Data model

Each control points is stored with its x, y, z coordinates, weight and multiplicity. Same structure is used for points on curve or patch. Objects are divided to two main categories, arcs or splines and patches. Both of these classes share basic functions (setting control points, precision, highlight parameter, etc.) and have same output format.

2.3.1. Arcs and splines

Firstly we describe arcs and splines. Input for their evaluation consists of sequence of control points highlight parameter and curve precision, but it can also include other parameters (only these differences are mentioned below). The highlight parameter represents parametric value. If any control point has multiplicity greater than one, application creates new control point with same coordinates and it is added into sequence just behind said control point. Output is the same across all classes. It is sequence of points and each segment has assigned same number of points as defined precision (arcs / curves consists of only one segment). Point on arc or spline that corresponds to the value of highlight parameter is labelled, so it can be highlighted later on during visualization. As part of output, blending functions are calculated and displayed in auxiliary window.

Types of supported curves are:

1. **Parametric curve**

   We evaluate coordinate functions for all three coordinates (defined in 4-th text box).

Input: Three coordinate functions (not sequence of points as all other curves)
Errors: Curve is degenerated to a single point.

2. **Bézier arc**

   We evaluate integral Bézier arc using de Casteljau algorithm. De Casteljau algorithm can be illustrated for given parameter.

Input: No difference.
Errors:Input consists of only 0 or 1 vertex.
Methods: Degree elevation – raises degree of Bézier arc (and computes new control vertices). Degree reduction – decreases degree of Bézier arc (and computes new control vertices).

3. **Rational Bézier arc**

   We evaluate rational Bézier arc using de Casteljau algorithm. De Casteljau algorithm can be illustrated for given parameter.

Input: No difference.
Errors: Input consists of only 0 or 1 vertex.
Methods: Degree elevation – raises degree of rational Bézier arc (and computes new control vertices).
Degree reduction – decreases degree of rational Bézier arc (and computes new control vertices).

4. **Cubic Hermite arc**

   We construct two tangents from input control vertices and then evaluate cubic Hermite arc.

Input: Sequence of 4 control points.
Errors: Input consists of less than 4 vertices.

5. **Quintic Hermite arc**

   We construct two tangents and two second derivatives vectors from input control vertices and then evaluate quintic Hermite arc.

Input: Sequence of 6 control points.

Errors: Input consists of less than 6 vertices.

6. Hermite spline

First we compute tangents at control points according to end conditions (spline type). Then we evaluate Hermite spline. Computed tangents can be illustrated.

Input: End condition (spline type), tangents.
Errors: Input consists of less than 3 vertices. Incompatible end conditions (spline type).

7. Cardinal spline

First we compute new control points according to end condition (spline type). Then we evaluate Cardinal spline with respect to parameter s.

Input: Parameter s, end condition (spline type), tangents.
Errors: Input consists of less than 4 vertices. Incompatible end conditions (spline type).

8. Bézier spline

First we compute new control points according to type of spline. Then we use de Casteljau for each segment of integral cubic Bézier spline. In the case of approximation spline, control points for each segments, and in the case of interpolation spline D_i can be shown.

Input: Type of spline, tangents
Errors: Input consists of less than 3 vertices.

9. Rational Bézier spline

First we compute new control points according to type of spline. Then we use de Casteljau for each segment of rational cubic Bézier spline. In the case of approximation spline, control points for each segments, and in the case of interpolation spline D_i can be shown.

Input: Type of spline, tangents
Errors: Input consists of less than 3 vertices.

10.Beta spline

First we compute new control points according to end condition (spline type). Then we evaluate Beta spline with respect to β_1 and β_2.

Input: Parameter β_1, parameter β_2, end condition (spline type), tangents.
Errors: Input consists of less than 4 vertices. Incompatible end conditions (spline type).

11. B-spline

Depending on type of knot vector, we either create uniform or open knot vector. In the case of user defined vector we check this vector for size, knot multiplicity and property of non-decreasing sequence. We solve any of these problems. Then we use de Boor algorithm for evaluation of point on B-spline. This algorithm can be illustrated for given parameter.

Input: Knot vector, type of knot vector, degree.
Errors: Input consists of less vertices than value of degree + 1. Enumeration interval is empty.
Methods: Knot insertion – inserts new knot value into knot vector (and computes new control vertices). Knot removal – removes knot value from knot vector (and computes new control vertices) using Boehm’s algorithm.

12. Rational B-spline - NURBS

Depending on type of knot vector, we either create uniform or open knot vector. In the case of user defined vector we check this vector for size, knot multiplicity and property of non-decreasing sequence. We solve any of these problems. Then we use de Boor algorithm for evaluation of point on rational B-spline. This algorithm can be illustrated for given parameter.

Input: Knot vector, type of knot vector, degree.
Errors: Input consists of less vertices than value of degree + 1. Enumeration interval is empty.
Methods: Knot insertion – inserts new knot value into knot vector (and computes new control vertices). Knot removal – removes knot value from knot vector (and computes new control vertices) using Boehm's algorithm.

2.3.2. Patches

Secondly we describe patches. Again, input consists of few common arguments – two dimensional array of control vertices forming control net, highlight parameter (now two dimensional) and patch precision. Any differences will be mentioned below as before. Each one of classes have same output. It is two dimensional array of patch points and number of points in both dimensions equals to defined precision. Point on patch that corresponds to highlight parameter is labelled, so it can be highlighted later on during visualization.

Types of supported patches:

1. **Parametric surface**
   We evaluate coordinate functions for all three coordinates (defined in 9-th, 10-th and 11-th text box).
   Input: Three coordinate functions (not sequence of points as in case of all other surfaces)
   Errors: Surface is degenerated to a single point (empty enumeration interval or 9-th, 10-th and 11-th text box defines only one points).

2. **Extruded surface**
   Function evaluates extruded surface. All points of the first curve are translated by vector defined by second curve.
   Input: Two dimensional array of points, which are on two curves (defined via 5-th and 7-th text box or one is modelled by user using curve modelling tools and one defined via 7-th text box), domain for these text box definitions (two numbers in 1-st text box and 2-nd text box).
   Errors: At least one curve is degenerated to a single point (empty enumeration interval or 5-th and 7-th text box defines only two points).

3. **Surface of revolution**
   Function evaluates surface of revolution by rotating curve around axis with respect to given angle.
   Input: Sequence of points, which are on curve (defined via 5-th text box or is created by user using curve modelling tools), domain for this text box definition (two numbers in 1-st text box), interval of revolution (two numbers in 3-rd text box), axis of rotation (3 numbers in 4-th text box).
   Errors: Whole curve is in plane perpendicular to rotation axis. Interval for revolution is empty. Curve is degenerated to a single point (empty enumeration interval or 5-th text box defines only one point).

4. **Linear Coons patch**
   Function evaluates linear Coons patch. Displays blending functions.

Input: Two dimensional array of points, which are on two boundary curves (defined via 5-th and 6-th text box), domain for these text box definitions (two numbers in 1-st text box).
Errors: Both curves are degenerated to a single point (empty enumeration interval or 5-th and 6-th text box defines only one point each).

5. **Bilinear Coons patch**
   Function evaluates bilinear Coons patch. Displays blending functions.

Input: Two dimensional array of points, which are on four boundary curves (defined via 5-th to 8-th text boxes), domain for these text box definitions (two numbers in 1-st and 2-nd text box).
Errors: Both boundary curves for u and/or v parametric direction are degenerated to a single point (empty enumeration interval or at least one pair of text boxes define only one point each). Boundary curves do not fulfil C^0 compatibility.

6. **Partially bicubic Coons patch**
   Function evaluates bilinear Coons patch. Displays blending functions.

Input: Two dimensional array of points, which are on four boundary curves (defined via 5-th to 8-th text boxes), domain for these text box definitions (two numbers in 1-st and 2-nd text box).
Errors: Both boundary curves for u and/or v parametric direction are degenerated to a single point (empty enumeration interval or at least one pair of text boxes define only one point each). Boundary curves (points on them) do not fulfil C^0 compatibility.

7. **Bicubic Coons patch**
   Function evaluates bicubic Coons patch. When it is initialised, control points are organised in a square with selectable size. Displays blending functions.

Input: Two dimensional array of points, which consists of 4 control points, 8 tangents both for u and v parametric direction and 4 twist vectors. Method for computing twist vectors.

8. **Bézier patch**
   We evaluate Bézier patch using bilinear interpolation. Blending functions are calculated and displayed in auxiliary window. When it is initialised, control points are organised in a square with selectable size and have z coordinate either set to 0, random value or points lie on half sphere. Displays blending functions.

Input: No difference.
Methods: Degree elevation – raises degree of Bézier patch in given parametric direction (and computes new control vertices). Degree reduction – decreases degree of Bézier patch in given parametric direction (and computes new control vertices).

9. **Rational Bézier patch**
   We evaluate rational Bézier patch using bilinear interpolation. Blending functions are calculated and displayed in auxiliary window. When it is initialised, control points are organised in square with selectable size and have z coordinate either set to 0, random value or points lie on half sphere. Displays blending functions.

Input: No difference.
Errors: Input consists of less than 3 non-collinear vertices with non-zero weight or all points with non-zero weights are in the same row or column.
Methods: Degree elevation – raises degree of Bézier patch in given parametric direction (and computes new control vertices). Degree reduction – decreases degree of Bézier patch in given parametric direction (and computes new control vertices).

10. B-spline patch

Depending on type of knot vectors, we either create uniform or open knot vectors. In the case of user defined vector we check this vector for size, knot multiplicity and property of non-decreasing sequence. We solve any of these problems. Then we use de Boor algorithm for evaluation of point on B-spline surface. Blending functions are calculated and displayed in auxiliary window. When it is initialised, control points are organised in square with selectable size and have *z* coordinate either set to 0, random value or points lie on half sphere. Displays blending functions.

Input: Knot vectors, degree for both parametric directions, type of knot vectors.
Errors: Input consists of less vertices than value of degree + 1 in either parametric direction. Enumeration interval is empty in either parametric direction.
Methods: Knot insertion – inserts new knot value into knot vector for given parametric direction (and computes new control vertices). Knot removal – removes knot value from knot vector (and computes new control vertices) using Boehm’s algorithm.

11. Rational B-spline patch - NURBS

Depending on type of knot vectors, we either create uniform or open knot vector. In the case of user defined vector we check this vector for size, knot multiplicity and property of non-decreasing sequence. We solve any of these problems. Then we use de Boor algorithm for evaluation of point on rational B-spline surface. Blending functions are calculated and displayed in auxiliary window. Displays blending functions.

Input: Knot vectors, degree for both parametric directions, type of knot vectors. When it is initialised, control points are organised in square with selectable size and have *z* coordinate either set to 0, random value or points lie on half sphere.
Errors: Input consists of less vertices than value of degree + 1 in either parametric direction. Enumeration interval is empty in either parametric direction. Input consists of less than 3 non-collinear vertices with non-zero weight or all points with non-zero weights are in the same row or column.
Methods: Knot insertion – inserts new knot value into knot vector for given parametric direction (and computes new control vertices). Knot removal – removes knot value from knot vector (and computes new control vertices) using Boehm’s algorithm.

3. Chapter – Implementation

This chapter describes implementation of *Splines & Surfaces* application. First we describe implementation of forms and then classes for curves and surfaces.

3.1. Forms

Main component in the main form *MainForm* is *3D window*. It consists of embedded OpenGL viewport. Its class can be downloaded from [14]. This viewport applies settings from *DisplayOptionsForm*. Projection type, point or line smoothing, line's width, size of points, enumeration precision for objects, secondary spline and surface colour, patch display mode (point cluster, wireframe, shaded and shaded with reflections) and visibility of vertices, control polygon, grid, coordinate axes, isocurves, illustrating points or point on an object can be changed here. If any change is made, it takes effect after user clicks on *Apply* or *Apply and close* button. *MainForm* can be resized by user.

All points, lines and objects are visualised with method `void UpdateWindow()`, which calls whichever function it needs from following list:

- `void DrawAxes()`
- `void DrawGrid(int n, double size)`
- `void ShowXcoordinate(TVector point)`
- `void ShowYcoordinate(TVector point)`
- `void ShowZcoordinate(TVector point)`
- `void ShowCoordinates(TVector point)`
- `void ShowVertex(TVector points, GLfloat size, GLfloat r, GLfloat g, GLfloat b)`
- `void ShowLine(TVector point1, TVector point2, GLfloat linewidth, GLfloat r, GLfloat g, GLfloat b)`
- `void ShowPolyLine(vector<TVector> points, GLfloat linewidth, GLfloat r, GLfloat g, GLfloat b)`
- `void ShowCurve(vector<TVector> points, bool smoothing, GLfloat linewidth, GLfloat r, GLfloat g, GLfloat b, GLfloat r2, GLfloat g2, GLfloat b2)`
- `void ShowVertices(vector<TVector> points, GLfloat size, GLfloat r, GLfloat g, GLfloat b)`
- `void ShowPatchPoints(vector< vector<TVector> > points, GLfloat size, GLfloat r, GLfloat g, GLfloat b)`
- `void ShowPatchPointsFergusson(vector< vector<TVector> > points, GLfloat size, GLfloat r, GLfloat g, GLfloat b)`
- `void ShowControlNet(vector< vector<TVector> > points, GLfloat linewidth, GLfloat r, GLfloat g, GLfloat b)`
- `void DrawPatch(vector< vector<TVector> > points, vector< vector<TVector> > normals)`
- `void DrawWireFrame(vector< vector<TVector> > points, GLfloat linewidth, GLfloat r, GLfloat g, GLfloat b)`
- `void DrawIsoCurves(vector< vector<TVector> > points, GLfloat linewidth, GLfloat r, GLfloat g, GLfloat b)`

View rotation is processed in `void UpdateProjectionMatrix()` and projection type is set in `void SetProjectionType()`.

Application handles mouse events for both mouse click and mouse move according to table of use cases on the page 76. It has integer values assigned to each system state, curve or surface type and others.

Creating coordinates for defining and moving points is done with `TVector RayIntersection(TVector RayDirection, TVector RayOrigin, TVector PlaneNormal, TVector PlaneOrigin)` method. Functions

- `tuple<int, int, double> NearestVertex(vector<TVector> Points, TVector RayDirection, TVector RayOrigin, double MaxRadius)`
- `tuple<int, double> NearestTangent(TVector RayDirection, TVector RayOrigin, double MaxRadius)`
- `tuple<int, int, double> NearestVertex(vector< vector<TVector> > Points, TVector RayDirection, TVector RayOrigin, double MaxRadius)`

depending on type of object, find closest point to mouse cursor. If it is closer than given threshold, it will become selected.

By clicking on whichever object type in menu or tool strip, either `void ComputeCurve(int type)` or `void ComputeSurface(int type)` are called. Both initialise chosen object and use its enumeration function create array of curve or surface points together with point on object and isocurves for surface. In some cases blending functions are also drawn inside *Auxiliary window* with `void DrawPolynomials(int type)` function. This function calls subroutines for enumeration of blending function in respective class. If rational Bézier or B-spline surface is modelled and weight of control point in i-th row and j-th column is changed, rational blending functions are displayed either for i-th row or j-th column for parametric direction u and v, respectively. Numeric box for changing i or j, respectively, manually is located inside *Control Box*.

Control Box groups various functionality. It offers selection of parametric direction, value of highlight parameter, degree for both parametric directions, multiplicity, weight and β_1, s and β_2 parameters. Values of multiplicity and weight are used in procedure of modifying control point via *Context menu*. Other items include text boxes and group of parameters for influencing initialisation of control net. If user wants construct affine surface from modelled curve, he has to check check box with this functionality.

Parameters for curves and patches, namely parameter s, β_1, β_2, degrees n and m and values of highlight parameter, multiplicity and weight of control vertex are being changed in numeric up and down boxes. These have set reasonable minimum and maximum value (e.g. parameters s, β_1, multiplicity and weight cannot be negative etc.). Knot vectors are stored in dynamically allocated array.

Initialisation of control net for tensor surfaces and bicubic Coons patch is modified by parameters in *Control net* panel. Created control net is uniformly sampled square shaped object with either z coordinate of its control points $z = 0$ for planar, z elevated to a half-sphere or z is random. Number of points in both parametric directions is controlled by respective numeric boxes and size of this net is controlled by *Scale* numeric box. For bicubic Coons patch, only this value is used.

For surfaces, normals are calculated in `void CalculateNormals(vector< vector<TVector> > points)` method with surface points as parameter. If object is already created, and parameters or position of control points are changed, method `void Recompute()` is called, which subsequently call functions for computing curve or surface, depending on object type.

Inside *3D window*, *Context menu* can be displayed. It offers options to add new control points on the end or begging of control points' sequence, to specify vertex, which will open *SpecifyVertexForm*, to delete control point, to change or to show its multiplicity or weight.

SpecifyVertexForm enables user to change position and other attributes of point with much higher precision than just by mouse click. Specified vertex could be either already created, or new one. Checked checkboxes select which values will be used during specification. Subroutine `void SpecifyVertex()` is called to update or to create new point with specified attributes.

Save file function `void SaveFile()` saves current graphic object as ".txt" file. Load function `void LoadFile()` processes loaded file token by token and creates new object from read data. This functions calls `void UpdateUI()` method to update all numeric boxes with loaded values.

KnotVectorOptionsForm is a form, which enables user to control knot vector. Form is resizable, so in the case of long knot vector it is possible to show all of its content. Numeric boxes for specifying index, its value, choice of parametric direction and options for setting knot vector to open, uniform, changing value specified by index, increasing multiplicity of knot specified by index and inserting or removing knot with specified value. Each change of knot index or value causes option labels to refresh with current index or value. Every operation has to be confirmed by *Apply* or *Apply and close* button. After mouse cursor leaves *Apply*, modified knot vector is refreshed in respective label.

Function `void ModifyKnotVector()` is called in both cases to handle chosen operation. In the case of inserting or removing knot, it first verifies if value of knot is in suitable interval and if value of removed knot is in knot vector. Then it calls subroutines

- `void UpdateKnotVector(int type, int i, double value)`
  `type:` defines parametric direction
  `i:` specifies index of modified knot
  `value:` new value of specified knot
  Updates knot with specified index to new value.
- `void IncreaseKnotMultiplicity(int type, double value)`
  `type:` defines parametric direction
  `value:` new value of specified knot
  Increases multiplicity of specified knot.
- `void InsertKnot(int type, double value)`
  `type:` defines parametric direction
  `value:` value of inserted knot

Calls appropriate subroutine inside **TNURBSpline** or **TNURBSurface** class to insert new knot specified by value to knot vector.

- `void RemoveKnot(int type, double value)`
  `type:` defines parametric direction
  `value:` value of removed knot
  Calls appropriate subroutine inside **TNURBSpline** or **TNURBSurface** class to remove knot specified by value from knot vector.
- `void SetKnotVector(int type)`
  `type:` defines type of knot vector
  Changes type of knot vector to uniform, open or custom.

All these functions call `void SetKnotStrings()` after successful operation. It refreshes strings of knots for *KnotVectorOptionsForm*.

During changing knot value or inserting new knot, knot value is tested if it knot with same value already has maximum allowed multiplicity. It is performed by method

`bool CheckKnotMultiplicity(int type, double value)`
`type:` defines parametric direction
`value:` value of tested knot

In the case of increasing multiplicity of knot, it is tested if knot vector contains this knot.

Text boxes are evaluated by `bool EvaluateTextBox(bool parametricsurface, bool parametriccurve)` (where variables `parametricsurface and parametriccurve` specifies, which text boxes are enumerated) function for certain types of surfaces and parametric curve. It returns `true` if all content of all text boxes is properly defined and `false` otherwise. Application uses external DLL library, *muparser*. It can be downloaded from [15]. This library offers functionality needed to convert expression in a form of string to mathematical expression, which can be evaluated later. Result of this operation is sequence of points that are later used as boundary curves or axis of rotation. It is also used to convert contents of text boxes with intervals for boundary curves.

Input may consists of any combination of addition, subtraction, multiplication, division of numbers or functions, all of which have predefined keyword. They are listed in table of keywords on the page 78. Library recognizes constants π and e, which have keywords *pi* and *e*, respectively.

Application detects errors for graphic objects specified earlier. Whenever it happens or whenever user tries to make invalid operation, message box with appropriate text is displayed, informing user about error. List of error and information messages is on the page 79 and 80, respectively.

Form *AboutForm* has only purpose to show information about version of software and its author. Icons for menu strip were created using screenshots from application. Only three icons for new, save and load file functions were downloaded from [16].

3.2. Classes

TVector class is used for storing points. It has double type variables for x, y, z, w coordinate and integer variable for multiplicity. Methods of this class include addition, subtraction, multiplication by scalar, multiplication by another vector both scalar and cross product, division by scalar, normalization and method for obtaining magnitude of vector. Actual sequence of points (used as input, output, and private variables inside individual classes) is stored in dynamically allocated array of points. In case of patches, this array is two dimensional.

TQuaternion class is similar to **TVector**, it has four double type variables for coordinates. Methods include addition, subtraction, multiplication by scalar and by another quaternion, division by scalar and methods for obtaining magnitude, inverse and normalized quaternion.

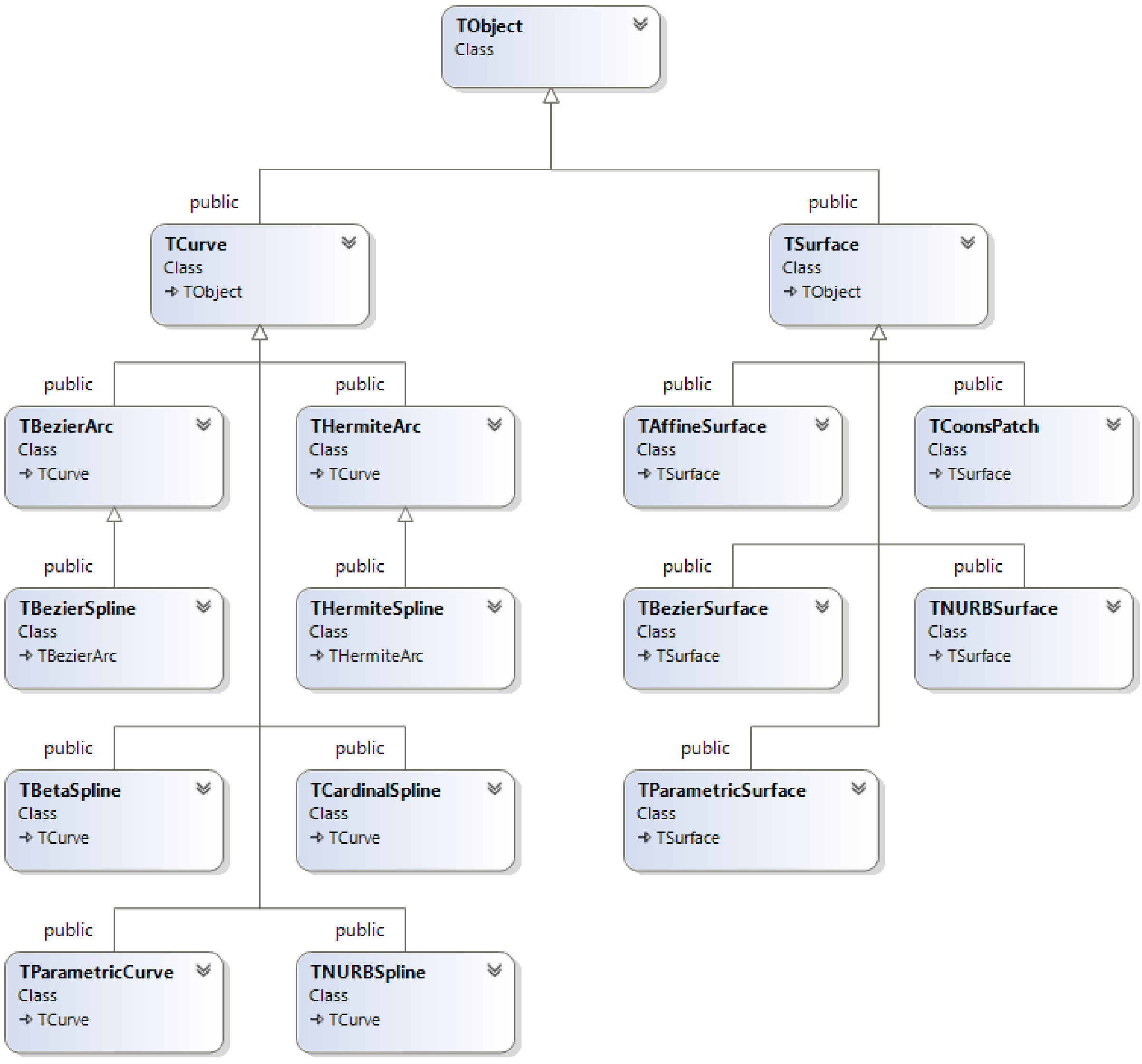


Figure 3-1 Class structure

Class structure for graphic objects has root in **TObject** class. (Figure 3-1) It has integer variable for error code and precision. It has only one method.

```
int GetError()
```

Returns integer code of an error.

3.2.1. Curve classes

TCurve class contains all variables common to mostly all arcs and splines. Namely sequence of control points, points on curve, point on curve corresponding to highlight parameter, spline type, number of enumerated control points - precision and highlight parameter.

Class methods include setting up control vertices, precision, highlight parameter and methods to obtain points on curve, control points and point on curve corresponding to highlight parameter. This list includes only functions, which are used during enumeration.

TCurve

`void SetControlPoints(vector<TVector> points)`

`points:`	sequence of control points

Sets control points for curve. If any control point has multiplicity greater than one, it creates new control point with same coordinates and it is added into sequence just behind processed control point.

`void SetPrecision(int precision)`

`precision:`	precision defines number of enumerated points

Sets precision and clears current curve points.

TParametricCurve

`void Init(vector<TVector> points, int precision, double parameter)`

`points:`	curve points
`precision:`	number of enumerated points
`parameter:`	highlight parameter

Initializes new points and other parameters for parametric curve. If all points have same coordinates detects an error.

`void ComputeCurve()`

Actual enumeration of curve points is done in *MainForm* using *muparser* library. Point on curve corresponding to highlight parameter is obtained for evaluated parametric value closest to highlight parameter.

TBezierArc

`void Init(vector<TVector> points, int precision, double parameter, int israte ional)`

`points:`	sequence of control points
`precision:`	number of enumerated points
`parameter:`	highlight parameter
`isrational:`	defines whether curve is integral or rational Bézier arc

Initializes new control points and other parameters for integral or rational Bézier arc. Tests number of points to catch an error if sequence does not have enough points.

```
void SetRationality(int isrational)
```

`isrational:`	defines whether curve is integral or rational Bézier arc

Sets type of curve according to input.

```
void DegreeElevation()
```

Elevates degree of integral or rational Bézier arc and computes new control points.

```
void DegreeReduction(int method)
```

`method:`	specifies method that is used to compute new control points

Reduces degree of integral or rational Bézier arc by chosen method and computes new control points.

```
double BernsteinPolynomial(double t, int n, int i)
```

`t:`	value, for which Bernstein polynomial is evaluated
`n:`	degree of Bernstein polynomial
`i:`	specifies i-th Bernstein polynomial to be evaluated

This function recursively enumerates specified polynomial. It is used to show blending functions.

```
double RationalPart(double t, int n)
```

`t:`	value, for which Bernstein polynomials are evaluated
`n:`	degree of Bernstein polynomials

This function recursively enumerates denominator of rational polynomial. It is used to show blending functions for rational Bézier arc.

```
void ComputeCurve()
```

Computes points on integral or rational Bézier arc using de Casteljau algorithm. Point on curve corresponding to highlight parameter is obtained for evaluated parametric value closest to highlight parameter. Sequence of points corresponding to steps in de Casteljau algorithm for highlight parameter is created.

THermiteArc

```
void Init(vector<TVector> points, int precision, double parameter, int type)
```

`points:`	sequence of control points
`precision:`	number of enumerated points
`parameter:`	highlight parameter
`type:`	defines whether arc is cubic or quintic

Initializes new control points and other parameters for Hermite arc. Tests number of points to catch an error if sequence does not have enough points depeding on type of an arc.

```
void CalculateHermitTangents()
```

Depending on a type of Hermite arc calculates either two tangents or two tangents and two second derivatives from control vertices.

```
double HermitePolynomial(double u, int n)
```

`u:`	value, for which cubic Hermite polynomial is evaluated
`n:`	specifies n-th cubic Hermite polynomial to be evaluated

This function enumerates specified cubic polynomial. It is used to show blending functions.

`double HermitePolynomialQuintic(double u, int n)`

`u:`	value, for which quintic Hermite polynomial is evaluated
`n:`	specifies n-th quintic Hermite polynomial to be evaluated

This function enumerates specified quintic polynomial. It is used to show blending functions.

`void ComputeCurve()`

Computes points on Hermite arc. Point on curve corresponding to highlight parameter is obtained for evaluated parametric value closest to highlight parameter.

THermiteSpline

```
void Init(vector<TVector> points, int precision, double parameter,
int splinetype, TVector tangents[2])
```

`points:`	sequence of control points
`precision:`	number of enumerated points for each segment
`parameter:`	highlight parameter
`splinetype:`	specifies end condition
`tangents[2]:`	two tangents for clamped spline

Initializes new control points and other parameters for Hermite spline. Tests number of points to catch an error if sequence does not have enough points.

`void SetTangents(TVector tangents[2])`

`tangents[2]:` two tangents for clamped spline

Sets tangents for clamped spline.

`void SetSplineType(int splinetype)`

`splinetype:` spline type defines end condition

Each end condition has assigned an integer number. Depending on its value, new control points satisfying chosen end condition are calculated. If spline type is incompatible, catches an error.

`double HermitePolynomial(double u, int n)`

`u:`	value, for which cubic Hermite polynomial is evaluated
`n:`	specifies $n - th$ cubic Hermite polynomial to be evaluated

This function enumerates specified cubic polynomial. It is used to show blending functions.

`void CalculateTangents()`

Calculates tangent in each control point.

`vector<TVector> SolveSystem(vector< vector<double> > A, vector<TVector> x)`

`A:`	system of linear equations
`x:`	vector consisting of constant terms

Finds solution for given system of linear equations using Gauss elimination.

`void ComputeCurve()`

Computes points on Hermite spline. Each segment is treated as Hermite arc and it is evaluated by **THermiteArc**'s function. Point on curve corresponding to highlight parameter is obtained for evaluated parametric value closest to highlight parameter.

TCardinalSpline

```
void Init(vector<TVector> points, int precision, double parameter, double sparameter, int splinetype, TVector tangents[2])
```

`points:`	sequence of control points
`precision:`	number of enumerated points for each segment
`parameter:`	highlight parameter
`sparameter:`	parameter s
`splinetype:`	specifies end condition
`tangents[2]:`	two tangents for clamped spline

Initializes new control points and other parameters for Cardinal spline. Tests number of points to catch an error if sequence does not have enough points.

```
void SetSParameter(double sparam)
```

`sparameter:`	parameter s

Sets value of parameter s.

```
void SetTangents(TVector tangents[2])
```

`tangents[2]:`	two tangents for clamped spline

Sets tangents for clamped spline.

```
void SetSplineType(int splinetype)
```

`splinetype:`	spline type defines end condition

Each end condition has assigned an integer number. Depending on its value, new control points satisfying chosen end condition are calculated. If spline type is incompatible, catches an error.

```
double CardinalPolynomial(double u, int n)
```

`u:`	value, for which cubic cardinal polynomial is evaluated
`n:`	specifies n-th cubic cardinal polynomial to be evaluated

This function enumerates specified polynomial. It is used to show blending functions.

```
void ComputeCurve()
```

Computes points on Cardinal spline. Point on curve corresponding to highlight parameter is obtained for evaluated parametric value closest to highlight parameter.

TBezierSpline

```
void Init(vector<TVector> points, int precision, double parameter, int typeisrational, bool clamped, TVector tangents[2])
```

`points:`	Sequence of control points
`precision:`	Number of enumerated points for each segment
`parameter:`	highlight parameter
`typeisrational:`	defines whether curve is integral or rational and interpolation or approximation Bézier spline

`clamped:` defines whether spline is clamped or not
`tangents[2]:` two tangents for clamped spline

Initializes new control points and other parameters for integral or rational and approximation or interpolation Bézier spline. Tests number of points to catch an error if sequence does not have enough points.

```
void SetRationality(int isrational)
```
`isrational:` defines whether curve is integral or rational Bézier spline

Sets type of spline according to input.

```
void SetTangents(TVector tangents[2])
```
`tangents[2]:` two tangents for clamped spline

Sets tangents for clamped spline.

```
void CreateNewControlPoints()
```

Creates control points for each segment depending on type of spline and whether it is clamped spline or not.

```
vector<TVector> SolveSystem(vector< vector<double> > A, vector<TVector> x)
```
`A:` system of linear equations
`x:` vector consisting of constant terms

Finds solution for given system of linear equations using Gauss elimination.

```
void ComputeCurve()
```

Computes points on integral or rational Bézier spline. Each segment is treated as integral or rational Bézier arc and it is evaluated by **TBezierArc**'s function. Point on curve corresponding to highlight parameter is obtained for evaluated parametric value closest to highlight parameter.

TBetaSpline

```
void Init(vector<TVector> points, int precision, double parameter, double
beta1, double beta2, int splinetype, TVector tangents[2])
```
`points:` sequence of control points
`precision:` number of enumerated points for each segment
`parameter:` highlight parameter
`beta1:` parameter β_1
`beta2:` parameter β_2
`splinetype:` specifies end condition
`tangents[2]:` two tangents for clamped spline

Initializes new control points and other parameters for Beta spline. Tests number of points to catch an error if sequence does not have enough points.

```
void SetBeta(double beta1, double beta2)
```
`beta1:` parameter β_1
`beta2:` parameter β_2

Sets values of parameters β_1 and β_2.

`void SetTangents(TVector tangents[2])`

`tangents[2]:` two tangents for clamped spline

Sets tangents for clamped spline.

`void SetSplineType(int splinetype)`

`splinetype:` spline type defines end condition

Each end condition has assigned an integer number. Depending on its value, new control points satisfying chosen end condition are calculated. If spline type is incompatible, catches an error.

`double BetaPolynomial(double u, int n)`

`u:`	value, for which cubic Hermite polynomial is evaluated
`n:`	specifies n-th cubic Hermite polynomial to be evaluated

This function enumerates specified polynomial. It is used to show blending functions.

`void ComputeCurve()`

Computes points on Beta spline. Point on curve corresponding to highlight parameter is obtained for evaluated parametric value closest to highlight parameter.

TNURBSpline

`void Init(vector<TVector> points, int precision, vector<double> knots, double parameter, int degree, int isrational, int knotvectortype)`

`points:`	sequence of control points
`precision:`	number of enumerated points for each segment
`knots:`	knot vector
`parameter:`	highlight parameter
`degree:`	degree of spline
`isrational:`	defines whether curve is integral or is rational B-spline
`knotvectortype:`	specifies type of knot vector

Initializes new control points and other parameters for integral or rational B-spline.

`void SetRationality(int isrational)`

`isrational:` defines whether curve is integral or rational B-spline

Sets type of curve according to input.

`void SetDegree(int degree)`

`degree:` degree of spline

Sets degree of spline. Tests number of points to catch an error if sequence does not have enough points.

`void SetKnotVectorType(int knotvectortype)`

`knotvectortype:` specifies type of knot vector

Sets type of knot vector.

`void SetKnotVector(vector<double> knots)`

`knots:` knot vector

Sets knot vector for B-spline. Depending on type of knot vector, either creates knot vector (in the case of uniform or open knot vector) or checks user defined knot vector for size, maximum multiplicity and non-decreasing property. Any problem is solved either by sorting, deleting or creating new knots. If enumeration interval is empty, catches an error.

`void KnotRemoval(double knotvalue)`
`knotvalue:` knot value, which is removed from knot vector

Removes knot from knot vector and computes new control vertices, both for integral or rational B-spline.

`void KnotInsertion(double knotvalue)`
`knotvalue:` knot value, which is inserted to knot vector

Inserts knot into knot vector using Boehm's algorithm and computes new control vertices, both for integral or rational B-spline.

`double NURBSPolynomial(double u, int p, int i)`
`u:` value, for which B-spline polynomial is evaluated
`p:` degree of B-spline polynomial
`i:` specifies i-th B-spline polynomial to be evaluated

This function recursively enumerates specified polynomial. It is used to show blending functions.

`double RationalPart(double u, int p)`
`u:` value, for which B-spline polynomials are evaluated
`p:` degree of B-spline polynomials

This function recursively enumerates denominator of rational polynomial. It is used to show blending functions for NURBS spline.

`void ComputeCurve()`

Computes points on integral or rational B-spline using de Boor algorithm. Point on curve corresponding to highlight parameter is obtained for evaluated parametric value closest to highlight parameter. Sequence of points corresponding to steps in de Boor algorithm for highlight parameter is created.

3.2.2. Surface classes

TPatch class contains all variables common to mostly all arcs and splines. Namely two dimensional array of control points and points on patch, point on patch corresponding to highlight parameter, precision and highlight parameter. This list includes only functions, which are used during enumeration.

TPatch

`void SetControlPoints(vector< vector<TVector> > points)`
`points:` two dimensional array of control points

Sets control net for patch.

`void SetPrecision(int precision)`
`precision:` precision defines both sizes of two dimensional array for enumerated points

Sets precision and clears current patch points.

TParametricSurface

```
void Init(vector< vector<TVector> > points, int precision, double parameter[2])
```

`points:`	two dimensional array of surface points
`precision:`	defines both sizes of two dimensional array for enumerated points
`parameter[2]:`	highlight parameter

Initializes new points and other parameters for parametric curve. If all points have same coordinates detects an error.

```
void ComputePatch()
```

Actual enumeration of surface points is done in *MainForm* using *muparser* library. Point on patch and isocurves corresponding to highlight parameter are obtained for evaluated parametric value closest to highlight parameter.

TAffineSurface

```
void Init(vector< vector<TVector> > points, TVector axis, int precision, double
parameter[2], double angle[2], int type)
```

`points:`	two sequences of control points
`precision:`	defines both sizes of two dimensional array for enumerated points
`parameter[2]:`	highlight parameter
`angle[2]:`	defines angle of rotation
`type:`	defines whether result is surface of revolution or extruded surface

Initializes new control points and other parameters for surface of revolution or extruded surface. Depending on type tests, if one or both curves are degenerated to a single point or if interval for rotation is empty or if curve lies in plane perpendicular to axis of rotation.

```
void ComputePatch()
```

Computes points on surface of revolution or extruded surface. In case of surface of revolution, spherical interpolation is used on quaternions to evaluate points on patch. Point on patch and isocurves corresponding to highlight parameter are obtained for evaluated parametric value closest to highlight parameter.

TCoonsPatch

```
void Init(vector< vector<TVector> > points, int precision, double parameter[2],
int type, int twisttype, int scale)
```

`points:`	four sequences of control points
`precision:`	defines both sizes of two dimensional array for enumerated points
`parameter[2]:`	highlight parameter
`type:`	defines whether patch is linear, bilinear, partially bicubic or bicubic Coons patch.
`twisttype:`	defines method for computing twist vectors
`scale:`	defines scale of created control net for bicubic Coons patch

Initializes new control points and other parameters for Coons patch. Detects if boundary curves are degenerated to a single point or if they fulfil C^0 compatibility. In the case of bicubic Coons patch sets twist vectors according to specified type.

`void SetTwistType(int twisttype)`

`twisttype:`	defines method for computing twist vectors

Sets type of twist vector.

`void SetControlPointsandTangents()`

If bicubic Coons patch is initialized, sets up control points for it scaled by scale modifier.

`void CalculateTangents()`

Calculates tangents and twist vectors, depending on their type, from control points.

`double HermitePolynomial(double u, int n)`

`u:`	value, for which cubic Hermite polynomial is evaluated
`n:`	specifies n-th cubic Hermite polynomial to be evaluated

This function enumerates specified cubic polynomial. It is used to show blending functions.

`void ComputePatch()`

Computes points on linear, bilinear, partially bicubic or bicubic Coons patch. Point on patch and isocurves corresponding to highlight parameter are obtained for evaluated parametric value closest to highlight parameter.

TBezierSurface

```
void Init(vector< vector<TVector> > points, int precision, double parameter[2],
int isrational)
```

`points:`	two dimensional array of control points
`precision:`	defines both sizes of two dimensional array for enumerated points
`parameter[2]:`	highlight parameter
`isrational:`	defines whether patch is integral or rational Bézier patch

Initializes new control net and other parameters for integral or rational Bézier patch.

`void SetRationality(int isrational)`

`isrational:`	defines whether patch is integral or rational B- patch

Sets type of patch according to input. Detects if input consists of less than 3 non-collinear vertices with non-zero weight or all points with non-zero weights are in the same row or column.

`void DegreeElevation(int type)`

`type:`	defines parametric direction for degree elevation

Elevates degree of integral or rational Bézier patch in given parametric direction and computes new control points.

`void DegreeReduction(int type, int method)`

`type:`	defines parametric direction for degree elevation
`method:`	specifies method that is used to compute new control points

Reduces degree of integral or rational Bézier patch in given parametric direction by chosen method and computes new control points.

```
double BernsteinPolynomial(double t, int n, int i)
```

`t:`	value, for which Bernstein polynomial is evaluated
`n:`	degree of Bernstein polynomial
`i:`	specifies i-th Bernstein polynomial to be evaluated

This function recursively enumerates specified polynomial. It is used to show blending functions.

```
double RationalPart(int type, int cr, double t, int n)
```

`type:`	specifies parametric direction
`cr:`	specifies column or row of control net
`t:`	value, for which Bernstein polynomials are evaluated
`n:`	degree of Bernstein polynomials

This function recursively enumerates denominator of rational polynomial. It is used to show blending functions for rational Bézier patch.

```
void ComputePatch()
```

Computes points on integral or rational Bézier patch by using bilinear interpolation. Point on patch and isocurves corresponding to highlight parameter are obtained for evaluated parametric value closest to highlight parameter.

TNURBSurface

```
void Init(vector< vector<TVector> > points, int precision, vector<double>
knotsu, vector<double> knotsv, double parameter[2], int degree[2],
int isratіonal, int knotvectortype[2])
```

`points:`	two dimensional array of control points
`precision:`	defines both sizes of two dimensional array for enumerated points
`knotsu:`	knot vector for parametric direction u
`knotsv:`	knot vector for parametric direction v
`parameter[2]:`	highlight parameter
`degree[2]:`	degree of patch in both parametric directions
`isrational:`	defines whether patch is integral or rational B-spline patch
`knotvectortype[2]:`	specifies types of knot vectors

Initializes new control net and other parameters for integral or rational B-spline patch.

```
void SetRationality(int isrational)
```

`isrational:`	defines whether patch is integral or rational B-spline

Sets type of patch according to input. Detects if input consists of less than 3 non-collinear vertices with non-zero weight or all points with non-zero weights are in the same row or column.

```
void SetDegree(int degree[2])
```

`degree[2]:`	degree of patch in both parametric directions

Set degree of spline. Tests number of points to catch an error if sequence does not have enough points.

```
void SetKnotVectorType(int knotvectortype[2])
```

`knotvectortype[2]:` specifies types of both knot vectors

Sets type of knot vector.

```
void SetKnotVectorType(int type, int knotvectortype)
```

`type:` specifies which knot vector is modified
`knotvectortype:` specifies type of knot vector

Sets type of knot vector for chosen knot vector.

```
void SetKnotVector(int type, vector<double> knots)
```

`type:` specifies which knot vector is modified
`knots:` knot vector

Sets knot vector for NURBS surface. Depending on type of knot vector, either creates knot vector (in the case of uniform or open knot vector) or checks user defined knot vector for size, maximum multiplicity and non-decreasing property. Any problem is solved either by sorting, deleting or creating new knots. If enumeration interval is empty, catches an error.

```
double NURBSPolynomial(int type, double u, int p, int i)
```

`type:`	specifies knot vector on which B-spline polynomial is evaluated
`u:`	value, for which B-spline polynomial is evaluated
`p:`	degree of B-spline polynomial
`i:`	specifies i-th B-spline polynomial to be evaluated

This function enumerates specified polynomial. It is used to show blending functions.

```
double RationalPart(int type, int cr, double t, int n)
```

`type:`	specifies parametric direction
`cr:`	specifies column or row of control net
`t:`	value, for which rational B-spline polynomials are evaluated
`n:`	degree of B-spline polynomials

This function recursively enumerates denominator of rational polynomial. It is used to show blending functions for rational NURBS patch.

```
void KnotRemoval(int type, double knotvalue)
```

`type:`	specifies knot vector from which knot is removed
`knotvalue:`	knot value, which is removed from knot vector

Removes knot from specified knot vector and computes new control vertices, both for integral or rational B-spline patch.

```
void KnotInsertion(int type, double knotvalue)
```

`type:`	specifies knot vector to which knot is inserted
`knotvalue:`	knot value, which is inserted to knot vector

Inserts knot into knot vector using Boehm’s algorithm and computes new control vertices, both for integral or rational B-spline patch.

```
void ComputePatch()
```

Computes points on integral or rational B-spline patch using de Boor algorithm. Point on patch and isocurves corresponding to highlight parameter are obtained for evaluated parametric value closest to highlight parameter.

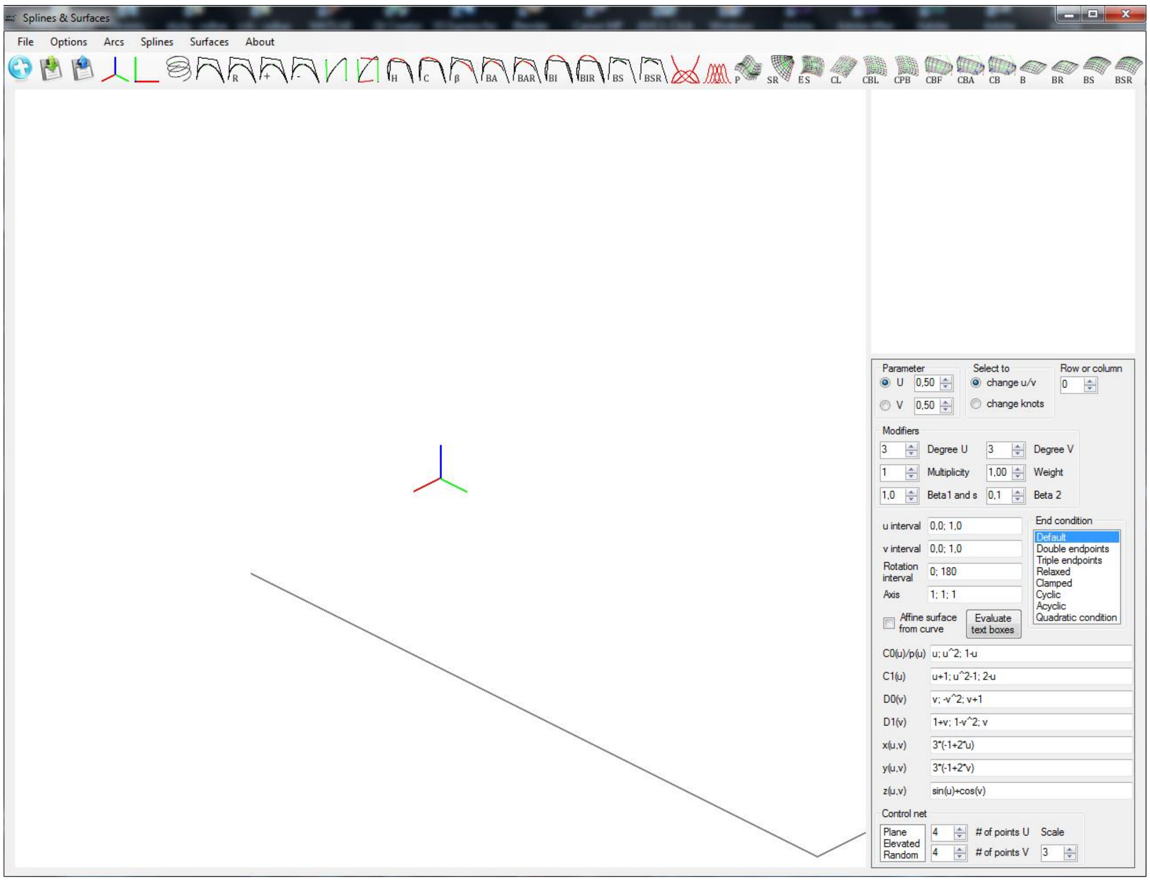
4. Chapter – User's guide

4.1. Installation

In order to run this application, you have to use at least Windows XP. You also need both Visual C++ Redistributable for Visual Studio 2012 Update 4 package and Microsoft .NET Framework 4 Client Profile installed. Both can be downloaded from [12] and [13] or you can find them on the included CD. You have to choose correct version for *x86* or *x64* architecture (32-bit and 64-bit).

File "muparserd32.dll" and/or "muparserd64.dll" have to be in same folder as executable file "Splines and surfaces.exe" in order to secure full functionality.

4.2. Graphic user interface



Parameter
(● U) 0,50
(○ V) 0,50

Select to
(● change u/v)
(○ change knots)

Row or column: 0

Modifiers
3 Degree U
3 Degree V
1 Multiplicity
1,00 Weight
1,0 Beta1 and s
0,1 Beta 2

u interval: 0,0; 1,0
v interval: 0,0; 1,0
Rotation interval: 0; 180
Axis: 1; 1; 1

☐ Affine surface from curve

Evaluate text boxes

End condition
[✓ Default]
Double endpoints
Triple endpoints
Relaxed
Clamped
Cyclic
Acyclic
Quadratic condition

C0(u)/p(u): u; u^2; 1-u
C1(u): u+1; u^2-1; 2-u
D0(v): v; -v^2; v+1
D1(v): 1+v; 1-v^2; v
x(u,v): 3*(-1+2*u)
y(u,v): 3*(-1+2*v)
z(u,v): sin(u)+cos(v)

Control net
Plane
Elevated
Random
4 # of points U
4 # of points V
Scale 3

Figure 4-1 Main window of Splines & Surfaces

When you launch *Splines & Surfaces* application, main window will show up (Figure 4-1). Menu strip is located in its top part. Tool strip is just under menu. The largest part of window is occupied by *3D window*. All graphic objects you create and modify will be displayed here. On the right side of *3D window* are *Auxiliary window* and *Control box* under it.

Inside menu under *File* tab, you can choose

- *New* create new object

- *Save* save currently modelled object
- *Load* load any previously saved object.

If you choose *Save* or *Load*, you will be prompted to choose folder and/or file name. Last option in this submenu is *Quit*. Under second tab *Options* you can find *Display options* and *Curve and patch options*. *Display options* are described below.

Curve and patch options include functions:

- *Degree reduction for Bézier curve or patch*
  Reduces degree of Bézier curve or patch with preselected method and recomputes current object. In case of patch, degree reduction will happen in selected parametric direction. If you hover mouse over this option, 3 more options will show up. These define method for reducing degree. You can choose one of them method by clicking on it.
    - *Method i/n*
    - *Method 0 : 1*
    - *Method sum*
- *Degree elevation for Bézier curve or patch*
  Elevates degree of Bézier curve or patch with preselected method and recomputes current object.
- *Reset weights of control points*
  Sets weights for all control points to 1.
- *Reset multiplicity of control points*
  Sets multiplicity for all control points to 1.
- *Set open knot vector*
  Sets knot vector type for selected parametric direction to open.
- *Set uniform knot vector*
  Sets knot vector type for selected parametric direction to uniform.
- *Set spline type*
  Sets spline type to one of following:
    - *Default – no end condition*
    - *Double endpoints*
    - *Triple endpoints*
    - *Relaxed spline*
    - *Clamped spline*
    - *Cyclic spline*
    - *Acyclic spline*
    - *Quadratic condition*

Under *Arcs* tab you can choose:

- *Parametric curve*
- *Bézier arc*
    - *Integral*
    - *Rational*
- *Hermite arc*
    - *Cubic*
    - *Quintic*

Under *Splines* tab you can choose:

- *Hermite spline*
- *Cardinal spline*
- *Bézier spline*
    - *Integral approximation Bézier spline*
    - *Rational approximation Bézier spline*
    - *Integral interpolation Bézier spline*
    - *Rational interpolation Bézier spline*
- *Beta spline*
- *B-spline:*
    - *Integral*
    - *Rational - NURBS*

Under *Surfaces* tab you can choose:

- *Parametric surface*
- *Affine surface*
    - *Surface of revolution*
    - *Extruded surface*
- *Coons patch:*
    - *Linear*
    - *Bilinear*
    - *Partially bicubic*
    - *Bicubic*
        - *Fergusson patch*
        - *Adini twist*
        - *Custom twist*
- *Bézier patch:*
    - *Integral*
    - *Rational*
- *B-spline patch:*
    - *Integral*
    - *Rational - NURBS*

Under *About* you can find information about author and version of software.

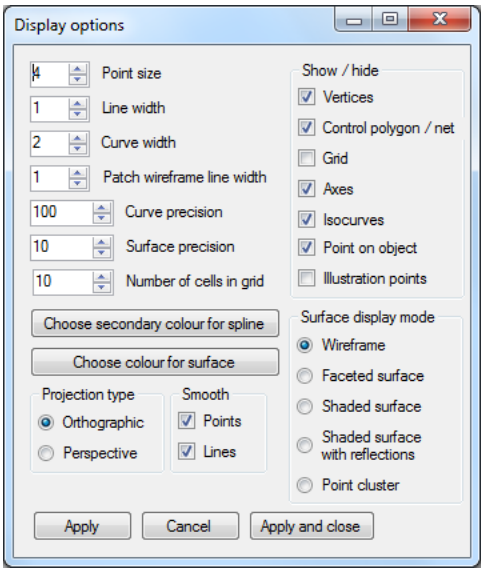
Just below Menu strip you see toolbar. It contains icons representing functions:

- *New*
- *Save*
- *Load*
- *Reset view*
- *Top view*
- *Degree elevation for Bézier curve or patch*
- *Degree reduction for Bézier curve or patch*
- *Set open knot vector*
- *Set uniform knot vector*

- Other buttons include all types of arcs, curves and surfaces. If you hover mouse cursor over any of them, tool tip shows up with information about object type.

On the right side of window, you can see large white square. This is *Auxiliary window*. Here, blending functions are displayed for most objects and also knot vector if it is used. In that case you can press right mouse button to show *Knot vector options* window. We describe it in detail later. *Control box* is located under this white square (described below).

4.3. Display options


Display options

4	Point size
1	Line width
2	Curve width
1	Patch wireframe line width
100	Curve precision
10	Surface precision
10	Number of cells in grid

Show / hide
- ✓ Vertices
- ✓ Control polygon / net
- ☐ Grid
- ✓ Axes
- ✓ Isocurves
- ✓ Point on object
- ☐ Illustration points

Projection type: [✓ Orthographic] Perspective

Smooth: ✓ Points, ✓ Lines

Surface display mode: [✓ Wireframe] Faceted surface, Shaded surface, Shaded surface with reflections, Point cluster

Figure 4-2 Display options

Display options window (Figure 4-2) covers all options which can modify how objects you create will look. You can change size of points, width of lines, curve and lines for wireframe. Next two numeric boxes control how precise will be object enumerated. It defines number of points that will form the object. Setting this value too high may result in lag, especially for parametric curve and surfaces. Last numeric box enables you to change size of grid in a plane $z = 0$.

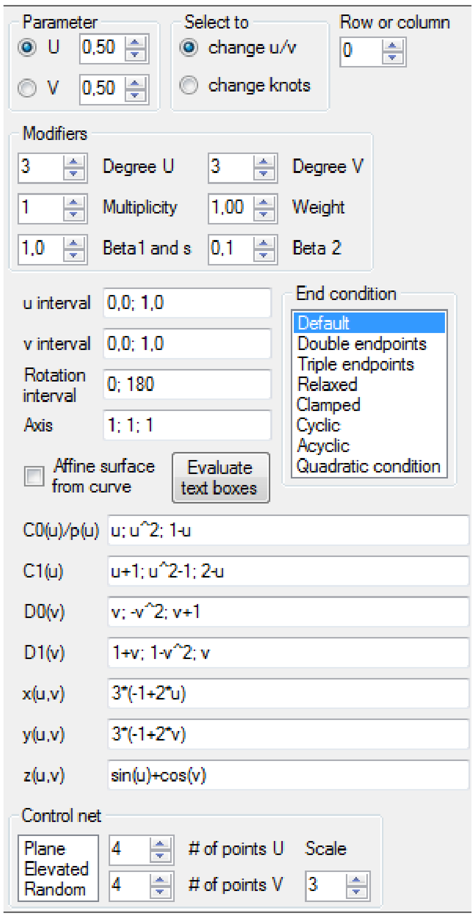
You can choose secondary colour for spline. Each odd segments will be coloured with this colour, so you can clearly see end points of each segment. Surface colour defines its base colour, which can be modified by shading.

Application offers you choice between orthographic and perspective projection. You can change it at any time. If you want display points as squares and not circles, deselect point smoothing option. Line smoothing applies antialiasing to lines and makes them little bit wider.

Show / hide section lets you select which objects will be displayed. *Vertices* and *Control polygon / net* applies for all curves but only for tensor product surfaces. *Grid*, *Axes*, *Isocurves* and *Point on object* are self-explanatory. *Illustration points* are points, which will show you process of curve enumeration. This applies in the case of Bézier arc, Bézier spline, B-spline and rational curves of these types. For (rational) Bézier approximation spline, control points for each segment and for interpolation spline points required to create C^2 continuous spline are displayed. If you have Hermite spline modelled then it will show you tangents at control points. If you hide control polygon, tangents are still displayed as points and lines illustrating algorithms are still visible.

In *Surface display mode* you can choose, how the surface will look. Default option is *Wireframe*, which is formed by lines connecting enumerated points on surface. *Faceted surface* displays whole patch with single colour. *Shaded surface* and *Shaded surface with reflections* incorporate lights into the scene. Last option *Point cluster* renders only points on surface.

4.4. Control box



Parameter: [✓ U] 0,50; V 0,50
Select to: [✓ change u/v]; change knots
Row or column: 0

Modifiers
3 Degree U; 3 Degree V
1 Multiplicity; 1,00 Weight
1,0 Beta1 and s; 0,1 Beta 2

u interval: 0,0; 1,0
v interval: 0,0; 1,0
Rotation interval: 0; 180
Axis: 1; 1; 1
Affine surface from curve
Evaluate text boxes

End condition: [✓ Default]; Double endpoints; Triple endpoints; Relaxed; Clamped; Cyclic; Acyclic; Quadratic condition

C0(u)/p(u): u; u^2; 1-u
C1(u): u+1; u^2-1; 2-u
D0(v): v; -v^2; v+1
D1(v): 1+v; 1-v^2; v
x(u,v): 3*(-1+2*u)
y(u,v): 3*(-1+2*v)
z(u,v): sin(u)+cos(v)

Control net: Plane; Elevated; Random
4 # of points U
4 # of points V
Scale 3

Figure 4-3 Control box

Control box (Figure 4-3) offers functionality to modify objects and each sub-section has tool tips ready to display. Under *Parameter* you can choose parameter, for which blending functions will be displayed in *Auxiliary window*. You can change value of highlight parameter by setting it inside numeric box or if *change u/v* is selected, by left clicking inside *Auxiliary window*. If *change knots* is selected and knots are displayed inside *Auxiliary window* (blue hollow rectangles), you can select one of them by pressing left mouse button, if mouse cursor is close enough. Selected knot will then become blue filled rectangle. By next left mouse click inside this window, you can define new value for selected knot and it will become hollow rectangle again (unselected).

Numeric box *Row or column* specifies which row or column of blending functions will be displayed. In u and v parametric direction it shows selected row and column, respectively. This option applies only for rational Bézier and B-spline surfaces.

Modifiers group include numeric boxes for degree in both parametric directions, multiplicity, weight and β_1, s and β_2 parameters. Values of multiplicity and weight are used in procedure of modifying control point.

Inside text boxes you can define curves, which will be used to construct surfaces. Text boxes labelled *u interval, v interval* and *Rotation interval* define intervals for u, v and rotation, respectively. Content of these text boxes have to be two numbers separated by ";" and you can use "," as decimal separator. You can define these numbers as value of a function, for example $\sin(5 * \pi)$. *Axis* text box defines axis of rotation for surface of revolution. Four curves (first of them serves also for parametric curve) can be defined via text boxes labelled *C0(u)/p(u), C1(u), D0(v)* and *D1(v)*. These five text boxes therefore need three numbers or expressions, respectively. Next three text boxes $x(u, v), y(u, v)$ and $z(u, v)$ are used to define parametric surface. Here only one expression per text box is needed. In all text boxes use only the parameter denoted in parenthesis for each respective expression. You can use any combination of addition, subtraction, multiplication, division of numbers or functions, all of which have predefined keyword. They are listed in table of keywords on the page 78.

If you want to save (and later load) your object created from text boxes' definition, you have to make sure that all expressions do not have spaces in them, but are separated with at least one space. For example (good)

- *u+1; u^2-2; sin(u)+1,5* for text boxes *C0(u)/p(u), C1(u), D0(v)* and *D1(v)*
- *u*2+1* for text boxes $x(u, v), y(u, v)$ and $z(u, v)$
- *0,1; 1,3* for intervals text boxes

as opposed to (wrong)

- *u +1;u^2- 2;sin(u) +1* for text boxes *C0(u)/p(u), C1(u), D0(v)* and *D1(v)*
- *u * 3,8 + 1.5* for text boxes $x(u, v), y(u, v)$ and $z(u, v)$
- *0,0;1,3* for intervals text boxes

which will not cause an error during evaluating but loading object will fail completely. You can use constants π and e, which have keywords *pi* and e, respectively .

Evaluate text boxes button serves to refresh currently modelled object. If you check *Affine surface from curve* checkbox, then affine surface will be constructed from curve, you have previously modelled.

You can choose *End condition* from list. In the case it is not supported by current spline, you will see error message. In the case of *Clamped* application's input mode will change to awaiting coordinates for tangent vectors. You have to define 2 tangent vectors. Process of defining these tangents is totally same as defining points.

Control net group defines parameters for initialisation of control net for tensor product surfaces and for bicubic Coons patch. You can change number of points via two numeric boxes (only for tensor product surfaces) and scale affects overall size of control net. List box contains options:

- *Plane:* control points have z coordinate 0
- *Elevated:* control points are located on an arc
- *Random:* control points have random z coordinate

4.5. Controls

When the application starts, it is awaiting coordinates of points. If you move mouse cursor, lines indicating actual cursor coordinates will show up. You can press left mouse button to set x and y coordinates of control point. Now application awaits z coordinate. You can either press left mouse button, which will set z coordinate to cursor position or press right mouse button to set z coordinate to 0. Application awaits again x and y coordinates for next control point. You can add as many points as you need. If you are finished with defining control points, you press right mouse button to cancel mode of defining new control points. In this mode coordinates of cursor are not displayed.

If you want to change position of any control point, you move mouse cursor above it. It will become selected if the cursor is close enough and application will let you know by highlighting that point with red colour and drawing lines to better illustrate its coordinates. Now press left mouse button. Again, define new x and y coordinates by left clicking. If you press right mouse button, x and y coordinates will not change. After pressing one or the other, you have to set z coordinate. You can set it to mouse cursor position by pressing left mouse button or to 0 by pressing right mouse button. It is essentially the same procedure as defining new point.

If you are not in point defining mode and no control point is selected, you can press left mouse button to start rotating *3D window*. Moving mouse cursor now rotates viewport. You can accept new view by pressing left mouse button or right mouse button. Zooming in and out is done via turning the mouse wheel at any time. If you press arrows on keyboard at any time with mouse cursor inside *3D window*, view port will shift in that direction.

If you are not in point defining mode, you can press right mouse button to show *Context menu*. (Figure 4-4)

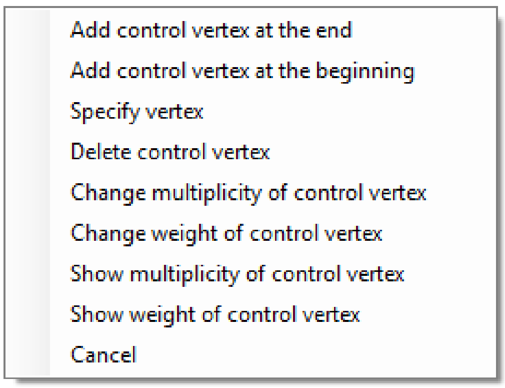
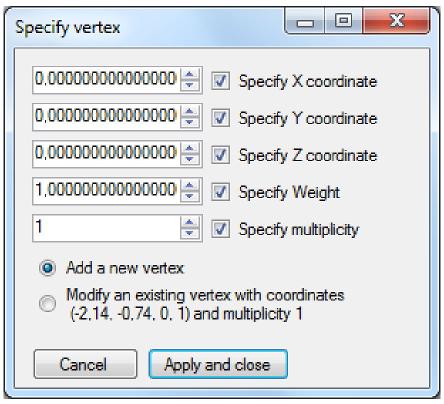

Add control vertex at the end
Add control vertex at the beginning
Specify vertex
Delete control vertex
Change multiplicity of control vertex
Change weight of control vertex
Show multiplicity of control vertex
Show weight of control vertex
Cancel

Figure 4-4 Context menu

It offers you choices of:

- *Add control vertex at the end*
  Enables you to add more control points to end of control points' sequence.
- *Add control vertices at the beginning*
  Enables you to add more control points to beginning of control points' sequence.
- *Specify vertex*
  Show form for specifying vertex. It is described in detail below.
- *Delete control vertex*
  If you have selected control point, it will delete it form sequence of control points.
- *Change multiplicity of control vertex*
  Sets multiplicity of selected control vertex to value specified in numeric box labelled *Multiplicity* located inside *Control box*
- *Change weight of control vertex*
  Sets weight of selected control vertex to value specified in numeric box labelled *Weight* located inside *Control box*
- *Show multiplicity of control vertex*
  Displays message window with value of multiplicity for selected vertex.
- *Show weight of control vertex*
- Displays message window with value of weight for selected vertex.
- *Cancel*
  Closes this context menu.

Choosing *Specify vertex* option will open new window. (Figure 4-5) If offers you two or only one choice, according to whether you have or have not selected control point (or tangent) before.


Specify vertex

0,000000000000000	✓ Specify X coordinate
0,000000000000000	✓ Specify Y coordinate
0,000000000000000	✓ Specify Z coordinate
1,000000000000000	✓ Specify Weight
1	✓ Specify multiplicity

● Add a new vertex
○ Modify an existing vertex with coordinates (-2.14, -0.74, 0, 1) and multiplicity 1

Cancel | Apply and close

Figure 4-5 Specify vertex

In the first case, you can only add new vertex to the end of control points' sequence. In the second case you can also modify selected vertex. Its actual coordinates are displayed too so you can see them at once. Check boxes allows you to change only some attributes of selected point. They do not have effect on attributes of new point.

4.6. Modelling objects

Objects are divided to two main groups, splines and surfaces. For splines, except one curve type, you have define control polygon by yourself. In the case of surfaces, application first creates the object and then you can modify it.

For arcs and spline, you simply create sequence of control points and then choose type of curve you want. You can modify it by changing position, weight and multiplicity of control vertices or tangents (in some cases even vectors of second derivative), modifying knot vector, changing end condition or using numeric boxes in control box if current spline is modifiable by them. If you select *Clamped*, application awaits coordinates for two tangent vectors. You define them exactly as control points. Changes will occur immediately.

One exception is parametric curve. This is defined by text box inside *Control box* and this is the only way to modify its shape (other than changing its enumerating interval). When you modify this expression, either choose *Parametric curve* from menu, click on its icon in toolbar or click on *Evaluate text boxes* button.

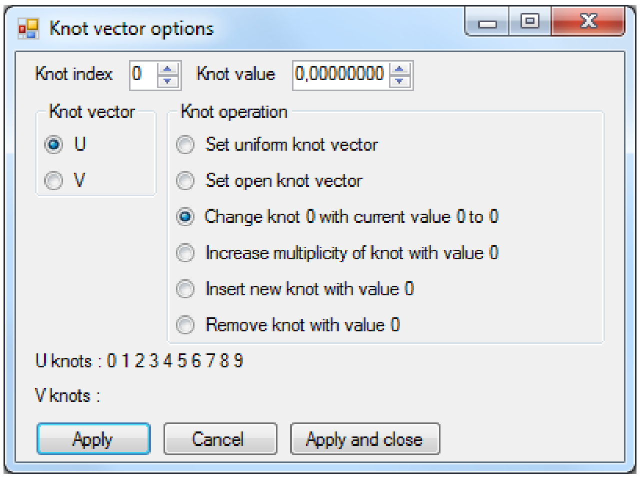
For surfaces, there are generally two ways to model them. For Bézier, B-spline and Bicubic Coons patch, control net is created by application. Scale of this control net is controlled by *Size* numeric box inside *Control net* in *Control box. # of points U* and*# of points V* do not affect bicubic Coons patch. You can modify it by changing position or weights, modifying knot vector and choosing different types of twist vectors.

If you model rational Bézier or B-spline surface, you can select which row or column of blending functions you want to see in *Auxialiary window*. You can select it in numeric box *Row or column.*

Other group consists of affine surfaces, Coons patches other than bicubic and parametric surface. All of them are defined by multiple text boxes inside *Control box.* Here you can change predefined expressions to modify shape of surface. In order to change take effect, you either choose desired type of surface in menu, click on its icon in toolbar or if you have any one of these surfaces already created, click on *Evaluate text boxes* button.

Affine surfaces offers you option to create surface of revolution or extruded surface either from curve, also called profile curve, defined in text box or the one you have modelled. First, you have to create a sequence of control points and select type of curve. Then you have to choose either type of affine surface and then select *Affine surface from curve* option. You can uncheck it at any time. To modify curve, which is now used to create affine surface, click on any one type of curve. This will bring you to curve modelling mode, where you can modify curve's shape. When you are satisfied, you can again select *Affine surface from curve* option.

If you will change precision with this option active, application will show message saying that no curve is defined. All you have to do is to select type of curve you want to be profile curve, then select type of affine surface and then finally select *Affine surface from curve* option.


Knot vector options

Knot index 0 Knot value 0,00000000

Knot vector
- [✓ U]
- V

Knot operation
- Set uniform knot vector
- Set open knot vector
- [✓ Change knot 0 with current value 0 to 0]
- Increase multiplicity of knot with value 0
- Insert new knot with value 0
- Remove knot with value 0

U knots : 0 1 2 3 4 5 6 7 8 9

V knots :

Figure 4-6 Knot vector options

Application will show you this window once you have B-spline curve or B-spline patch modelled and you press right mouse button while mouse cursor is in *Auxiliary window*. (Figure 4-6) In the case of patches, you can choose knot vector for respective parametric direction. Two numeric boxes change index and value of knot. If you change their value, labels for operations will change itself so you can clearly see which knot with its current value will be modified to new value, its multiplicity will be increased or which knot value will be inserted or removed from knot vector. Other two operations serves purpose of setting knot vector type to open or uniform. If you have clicked on *Apply* button, move mouse cursor away from it and new knot vector will be written down.

Conclusion

There are many software applications for visualisation of splines and surfaces. We can divide them in to two groups – web applets and other simple programs and other one is professional CAD tools.

Web applets are usually designed to visualise only one type of curve or spline and each one of them may have different design and controls. This is not the case in *Splines & Surfaces*.
Professional software for CAD on the other hand usually comes with well devised controls and design. One downside to this is that they are not often free to use and second that these applications are designed for work and not for educational purpose.

We have not find any other software that would offer such wide variety types of splines and surfaces. For sure there are better applications than *Splines & Surfaces*, in terms of rendering, creating large surfaces composed form large number of patches, mainly professional ones, but we think that it is suitable for the job.

Modification of source code can be also used as template for students. They would not have to design and develop entire application, which takes much more time than writing code only for example initialisation of B-spline and de Boor's algorithm.

Possible extensions in the future may include new classes of both splines and surfaces, for example Bézier triangle patches or DMS splines, visualisation of bi-parametric blending functions as surfaces, creating more than one object, for example tensor and triangle Bézier patch as well as improved rendering options. Including new class into implementation should not be very difficult.

Literature

[1] KUDLIČKOVÁ, S.: Representation of geometric objects, study materials. [online] 3.5.2016 Available at < https://flurry.dg.fmph.uniba.sk/webog/sk/kudlickova-vyucba/311-reprezentacie-geometrickych-objektov.html >

[2] ROGERS, F. D.: An introduction to NURBS With Historical Perspective. San Francisco : Morgan Kaufmann, 2001. 324 p. ISBN 1-55860-669-6

[3] FARIN, G.: Curves and surfaces CAGD A practical guide. Fifth edition. San Francisco : Morgan Kaufmann, 2001. 520 p. ISBN 1-55860-737-4

[4] SZIRMAY-KALOS, L. et al.: Theory of Three Dimensional Computer Graphics. [online] 3.5.2016 Available at < http://www.fsz.bme.hu/~szirmay/ani.pdf >

[5] HOSCHEK J. – LASSER, D.: Fundamentals of Compute Aided Geometric Design. Wellesley, Massachusetts : A K Peters, 1993. 727 p. ISBN 1-56881-007-5

[6] MORGAN, D.: Degree reduction of Bézier curves. [online] 3.5.2016 Available at < http://yamnuska.ca/geek/degreeReductionBezier.pdf >

[7] SEDERBERG, W. T.: Computer aided geometric design. [online] 3.5.2016 Available at < http://tom.cs.byu.edu/~557/text/cagd.pdf >

[8] ECK, M. – HADENFELD, J.: Knot Removal for B-Spline Curves. [online] 2.5.2016 Avaible at < http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.32.177&rep=rep1&type=pdf >

[9] MARSH, D.: Applied Geometry for Computer Graphics and CAD. Great Britain : Springer, 1999. 288 p. ISBN 1852330805

[10] CHALMOVIANSKY, P.: Modelling curves and surfaces, study materials

[11] AGARWAL, R. B.: Computer Aided Design in Mechanical Engineering. [online] 3.5.2016 Available at < http://www.engr.sjsu.edu/ragarwal/ME165/ME165_Lecture_Notes_files/ >

[12] MICROSOFT. Visual C++ Redistributable for Visual Studio 2012 Update 4 [online] 3.5.2016 < https://www.microsoft.com/en-us/download/details.aspx?id=30679 >

[13] MICROSOFT. .NET Framework 4 Client Profile (Standalone Installer) [online] 3.5.2016 < https://www.microsoft.com/en-us/download/details.aspx?id=24872 >

[14] WOOTON, B. Creating an OpenGL view on a Windows Form. [online] 3.5.2016 < www.codeproject.com/Articles/16051/Creating-an-OpenGL-view-on-a-Windows-Form >

[15] MUPARSER. [online] 3.5.2016 < http://beltoforion.de/article.php?a=muparser >

[16] ICONMOON. Web application Icons by IconMoon. [online] 3.5.2016 < http://www.iconarchive.com/show/web-application-icons-by-iconmoon.html >

Attachments

Table 1 Use cases

INPUT	SYSTEM STATE	RESPONSE
Left mouse button pressed	Awaiting x and y coordinates for sequence of control points	Set x and y coordinate as intersection with plane $z = 0$, change state to await z coordinate for sequence
Left mouse button pressed	Awaiting z coordinate for sequence of control points	Set z coordinate as intersection with plane $y = 0$, change state to await x and y coordinate for sequence
Right mouse button pressed	Awaiting z coordinate for sequence of control points	Set z coordinate to 0, change state to await x and y coordinate for sequence
Right mouse button pressed	Awaiting x and y coordinates for sequence of control points	Finish constructing sequence of control points, change state to awaiting input
Right mouse button pressed	Awaiting input	Show context menu
Mouse move	Awaiting input	Detect if cursor is around any control point, if it is, select point, if not, deselect point
Left mouse button pressed	Awaiting input, point selected	Change state to await x and y coordinate for selected point
Left mouse button pressed	Awaiting x and y coordinate for selected point	Set x and y coordinate as intersection with plane $z = 0$, change state to await z coordinate for selected point
Right mouse button pressed	Awaiting x and y coordinate for selected point	Do not change x and y coordinate, change state to await z coordinate for selected point

Left mouse button pressed	Awaiting z coordinate for selected point	Set z coordinate as intersection with plane $y = 0$, change state to await input
Right mouse button pressed	Awaiting z coordinate for selected point	Set z coordinate to 0, change state to await input
Left mouse button pressed	Awaiting input	Change state to rotate view
Mouse move	Rotating view	Rotate view
Left mouse button pressed	Rotating view	Stop rotating view
Mouse wheel scroll	Any state	Zoom in or out
Arrow pressed	Any state	Translate view in given direction

Table 2 Keywords

KEYWORD	# OF ARGUMENTS	FUNCTION
`sin`	1	sine function
`cos`	1	cosine function
`tan`	1	tangens function
`asin`	1	arcus sine function
`acos`	1	arcus cosine function
`atan`	1	arcus tangens function
`sinh`	1	hyperbolic sine function
`cosh`	1	hyperbolic cosine
`tanh`	1	hyperbolic tangens function
`asinh`	1	hyperbolic arcus sine function
`acosh`	1	hyperbolic arcus tangens function
`atanh`	1	hyperbolic arcur tangens function
`log2`	1	logarithm to the base 2
`log10`	1	logarithm to the base 10
`log`	1	logarithm to the base 10
`ln`	1	logarithm to base e (2.71828...)
`exp`	1	e raised to the power of x
`sqrt`	1	square root of a value
`sign`	1	sign function -1 if x<0; 1 if x>0
`rint`	1	round to nearest integer
`abs`	1	absolute value
`min`	var.	min of all arguments
`max`	var.	max of all arguments
`sum`	var.	sum of all arguments
`avg`	var.	mean value of all arguments

List 1 Error messages

User tries to save current graphic object, but no one is created.
User tries to use degree elevation or reduction for another object than Bézier arc or patch.
User tries to use degree reduction, but resulting object would be only a point or a curve for Bézier arc and patch, respectively.
User tries to create affine patch form curve, but not curve is defined.
Text box for defining interval for u parameter contains unrecognisable keyword.
Interval for u parameter consists of less or more than two separate numbers.
Interval for u parameter consists of two numbers with same value.
Text box for defining interval for v parameter contains unrecognisable keyword.
Interval for v parameter consists of less or more than two separate numbers.
Interval for v parameter consists of two numbers with same value.
Text box for defining rotation's interval contains unrecognisable keyword.
Interval for rotation consists of less or more than two separate number.
Interval for rotation consists of two numbers with same value.
Text box for defining of rotation's axis contains unrecognisable keyword.
Text box for defining of rotation's axis does not consist of three separate numbers.
At least one text box for defining curves contains unrecognisable keyword.
At least one of text boxes for defining boundary curves does not consist of three separate expressions.
Text box for defining x coordinate for parametric surface contains unrecognisable keyword.
Text box for defining x coordinate for parametric surface contains more than one expression.
Text box for defining y coordinate for parametric surface contains unrecognisable keyword.
Text box for defining y coordinate for parametric surface contains more than one expression.
Text box for defining z coordinate for parametric surface contains unrecognisable keyword.
Text box for defining z coordinate for parametric surface contains more than one expression.
Text box for defining parametric curve contains unrecognisable keyword.
Text box for defining parametric curve contains does not consist of three separate expressions.
User tries to change knot vector (by changing knot value, increasing knot multiplicity or inserting new knot), but operation would result in invalid knot vector.
User tries to increase multiplicity of knot, but knot with specified value do not exist in knot vector.
User tries to insert knot with value, but knot with that value already has maximum allowed multiplicity.
User tries to insert knot, but is it not a valid value.
User tries to remove knot with value, but no knot in knot vector has same value.
User tries to specify new vertex for patch via specify point form.
User tries to specify new vertex with zero weight via specify point form.
User tries to modify first or last control point weight to zero via specify point form.
User tries to specify point for patch via specify point form, but no point is selected.
User tries to add new control point either at the end or begging via context menu while patch is being modelled.
User tries to delete tangent (for clamped splines).
User tries to delete control point, but no one is selected.
User tries to delete control point, tangent or second derivative for Hermite arc.
User tries to delete control point of patch.
User tries to set weight of control point via context menu, but no point is selected.

User tries to set weight of first or last control point to zero via context menu.
User tries to set weight of patch control point via context menu, but no control point is selected.
User tries to set multiplicity of control point via context menu, but no point is selected.
User tries to set multiplicity of patch control point via context menu.
User tries to show multiplicity of control point via context menu, but no point is selected.
User tries to show weight of control point or patch control point via context menu, but no point is selected.
User tries to set spline’s end condition to clamped, but the curve he is working with does not support this condition.
User clicks on “Coons patch – bicubic” without modelling this type of patch before.
With *show illustration points* option active, user models Hermite spline with at least one vertex with multiplicity greater than 1.
User tries to display rational blending functions for rational Bézier or B-spline surface, but neither one is modelled.
User tries to save an object but there is no object

List 2 Information messages

Message asking user, if he wants to proceed with degree reduction of Bézier arc, when it will change shape of curve.
Message asking user, if he wants to proceed with degree reduction of Bézier patch, when it will change shape of patch.
User has defined more control vertices than number needed for cubic or quintic Hermite arc and excess vertices will be deleted.
User selected other end condition than default or clamped for Bézier spline.
User tries to modify v knot vector via knot vector form, but he is working with curves.
User tries to modify multiplicity of patch control point via specify point form.
User tries to modify multiplicity of tangents via specify point form.
User tries to modify weight of tangents via specify point form.
Message displaying multiplicity of control point.
Message that says multiplicity of all patch control points is one.
Message displaying weight of control point.
Message displaying weight of patch control point.
Message saying that tangents’ weight equals one.
Message saying that tangents’ multiplicity equals one.
User tries to select radio button "v" parameter for secondary window, but he is not modelling patch.
User tries to select radio button “change knots” for secondary window, but he is not working with spline or patch that uses knot vector.
User tries to select *Affine surface from curve* but he is not working with affine surfaces.