Datorgrafik HT 2006

Curves and Surfaces

Splines, NURBS and such

By
Ulf Assarsson

Most of the material is originally made by Edward Angel and modified by Ulf Assarsson. Some is made by Magnus Bondesson

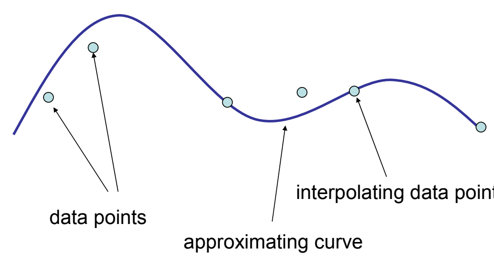
Introduction

- Read "KURV- OCH YTAPPROXIMATION MED POLYNOM" by Magnus Bondesson:
  - http://www.ce.chalmers.se/edu/year/2006/course/EDA360/DG_KURV2004.pdf
- See also course book, Angel "Interactive Computer Graphics – A Top-Down Approach Using OpenGL" chapter 11, pages 569-624
- Läs avsnitt 24, sid 34-38 i "Introduktion till OpenGL"-häftet på kurshemsidan.
- OH 114-138

Objectives

- Introduce types of curves and surfaces
  - –Explicit
  - –Implicit
  - –Parametric

Modeling with Curves



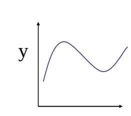
What Makes a Good Representation?

- There are many ways to represent curves and surfaces
- Want a representation that is
  - –Stable
  - –Smooth
  - –Easy to evaluate
  - –Must we interpolate or can we just come close to data?
  - –Do we need derivatives?

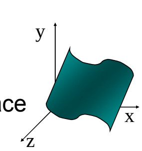
Explicit Representation

- Most familiar form of curve in 2D

  $y=f(x)$



- Cannot represent all curves
  - –Vertical lines
  - –Circles
- Extension to 3D
  - –$y=f(x)$, $z=g(x)$
  - –The form $z = f(x,y)$ defines a surface



Implicit Representation

- Two dimensional curve(s)

  $g(x,y)=0$
- Much more robust
  - –All lines $ax+by+c=0$
  - –Circles $x^2+y^2-r^2=0$
- Three dimensions $g(x,y,z)=0$ defines a surface
  - –Intersect two surface to get a curve

Parametric Curves

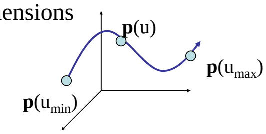
- Separate equation for each spatial variable

  $x=x(u)$

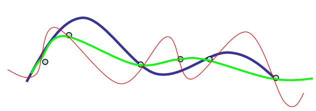
  $y=y(u)$

  $z=z(u)$

  $\mathbf{p}(u)=[x(u), y(u), z(u)]^T$
- For $u_{max} \geq u \geq u_{min}$ we trace out a curve in two or three dimensions



Selecting Functions



- Usually we can select “good” functions
  - not unique for a given spatial curve
  - Approximate or interpolate known data
  - Want functions which are easy to evaluate
  - Want functions which are easy to differentiate
    - Computation of normals
    - Connecting pieces (segments)
  - –Want functions which are smooth

Parametric Lines

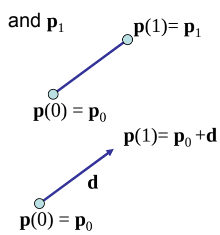
We can let u be over the interval (0,1)

Line connecting two points $\mathbf{p}_0$ and $\mathbf{p}_1$

$\mathbf{p}(u)=(1-u)\mathbf{p}_0+u\mathbf{p}_1$

Ray from $\mathbf{p}_0$ in the direction $\mathbf{d}$

$\mathbf{p}(u)=\mathbf{p}_0+u\mathbf{d}$



Parametric Surfaces

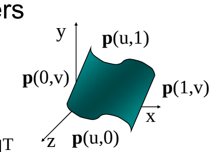
- Surfaces require 2 parameters

  $x=x(u,v)$

  $y=y(u,v)$

  $z=z(u,v)$

  $\mathbf{p}(u,v) = [x(u,v), y(u,v), z(u,v)]^T$
- Want same properties as curves:
  - –Smoothness
  - –Differentiability
  - –Ease of evaluation

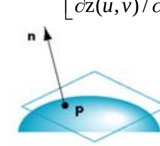


Normals

We can differentiate with respect to u and v to obtain the normal at any point $\mathbf{p}$

$$\frac{\partial \mathbf{p}(u,v)}{\partial u} = \begin{bmatrix} \partial x(u,v)/\partial u \\ \partial y(u,v)/\partial u \\ \partial z(u,v)/\partial u \end{bmatrix} \qquad \frac{\partial \mathbf{p}(u,v)}{\partial v} = \begin{bmatrix} \partial x(u,v)/\partial v \\ \partial y(u,v)/\partial v \\ \partial z(u,v)/\partial v \end{bmatrix}$$

$$\mathbf{n} = \frac{\partial \mathbf{p}(u,v)}{\partial u} \times \frac{\partial \mathbf{p}(u,v)}{\partial v}$$



Parametric Planes

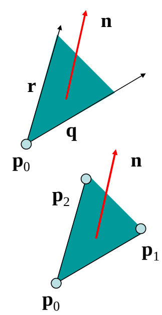
point-vector form

$\mathbf{p}(u,v)=\mathbf{p}_0+u\mathbf{q}+v\mathbf{r}$

$\mathbf{n} = \mathbf{q} \times \mathbf{r}$

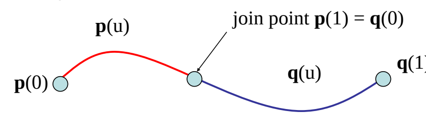
(three-point form

$\mathbf{q} = \mathbf{p}_1 - \mathbf{p}_0$
$\mathbf{r} = \mathbf{p}_2 - \mathbf{p}_0$)



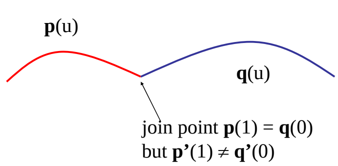
Curve Segments

- After normalizing u, each curve is written
  $\mathbf{p}(u)=[x(u), y(u), z(u)]^T, \quad 1 \geq u \geq 0$
- In classical numerical methods, we design a single global curve
- In computer graphics and CAD, it is better to design small connected curve *segments*



We choose Polynomials

- Easy to evaluate
- Continuous and differentiable everywhere
  - Must worry about continuity at join points including continuity of derivatives



Parametric Polynomial Curves

$$x(u)=\sum_{i=0}^{N} c_{xi} u^i \quad y(u)=\sum_{j=0}^{M} c_{yj} u^j \quad z(u)=\sum_{k=0}^{L} c_{zk} u^k$$

•Cubic polynomials gives N=M=L=3

•Noting that the curves for x, y and z are independent, we can define each independently in an identical manner

•We will use the form $p(u)=\sum_{k=0}^{L} c_k u^k$ where p can be any of x, y, z

Cubic Parametric Polynomials

- Cubic polynomials give balance between ease of evaluation and flexibility in design

$$p(u)=\sum_{k=0}^{3} c_k u^k$$

- Four coefficients to determine for each of x, y and z
- Seek four independent conditions for various values of u resulting in 4 equations in 4 unknowns for each of x, y and z
  - Conditions are a mixture of continuity requirements at the join points and conditions for fitting the data

Objectives

- Introduce the types of curves
  - Interpolating
    - Blending polynomials for interpolation of 4 control points (fit curve to 4 control points)
  - Hermite
    - fit curve to 2 control points + 2 derivatives (tangents)
  - Bezier
    - 2 interpolating control points + 2 intermediate points to define the tangents
  - B-spline
    - To get C^2 continuity
  - NURBS
    - Different weights of the control points
- Analyze them

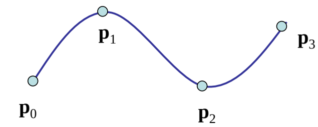
Matrix-Vector Form

$$p(u) = \sum_{k=0}^{3} c_k u^k$$

define $\mathbf{c} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$ $\quad \mathbf{u} = \begin{bmatrix} 1 \\ u \\ u^2 \\ u^3 \end{bmatrix}$

then $p(u) = \mathbf{u}^T \mathbf{c} = \mathbf{c}^T \mathbf{u}$

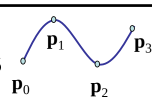
Interpolating Curve



Given four data (control) points $\mathbf{p}_0$, $\mathbf{p}_1$,$\mathbf{p}_2$, $\mathbf{p}_3$
determine cubic $\mathbf{p}$(u) which passes through them

Must find $\mathbf{c}_0$,$\mathbf{c}_1$,$\mathbf{c}_2$, $\mathbf{c}_3$

Interpolation Equations



$p(u) = c_0 + c_1 u + c_2 u^2 + c_3 u^3$

apply the interpolating conditions at u=0, 1/3, 2/3, 1

$p_0=p(0)=c_0$
$p_1=p(1/3)=c_0+(1/3)c_1+(1/3)^2c_2+(1/3)^3c_3$
$p_2=p(2/3)=c_0+(2/3)c_1+(2/3)^2c_2+(2/3)^3c_3$
$p_3=p(1)=c_0+c_1+c_2+c_3$

or in matrix form with $\mathbf{p} = [p_0\ p_1\ p_2\ p_3]^T$

$\mathbf{p=Ac}$

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & \left(\frac{1}{3}\right) & \left(\frac{1}{3}\right)^2 & \left(\frac{1}{3}\right)^3 \\ 1 & \left(\frac{2}{3}\right) & \left(\frac{2}{3}\right)^2 & \left(\frac{2}{3}\right)^3 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

I.e., $\mathbf{c=A^{-1}p}$

Interpolation Matrix

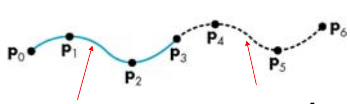
Solving for $\mathbf{c}$ we find the *interpolation matrix*

$$\mathbf{M}_I = \mathbf{A}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -5.5 & 9 & -4.5 & 1 \\ 9 & -22.5 & 18 & -4.5 \\ -4.5 & 13.5 & -13.5 & 4.5 \end{bmatrix}$$

$$\mathbf{c} = \mathbf{M}_I \mathbf{p}$$

Note that $\mathbf{M}_I$ does not depend on input data and can be used for each segment in x, y, and z

Interpolating Multiple Segments

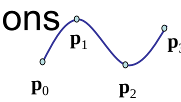


use $\mathbf{p} = [p_0\ p_1\ p_2\ p_3]^T$

use $\mathbf{p} = [p_3\ p_4\ p_5\ p_6]^T$

Get continuity at join points but not continuity of derivatives

Blending Functions



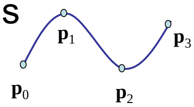
Rewriting the equation for p(u)

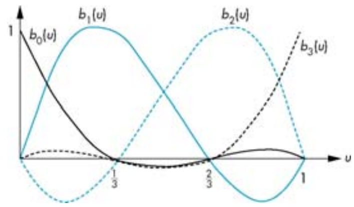
$$p(u) = \mathbf{u}^T\mathbf{c} = \mathbf{u}^T\mathbf{M}_I\mathbf{p} = \mathbf{b}(u)^T\mathbf{p}$$

where $b(u) = [b_0(u)\ b_1(u)\ b_2(u)\ b_3(u)]^T$ is an array of *blending polynomials* such that
$p(u) = b_0(u)p_0 + b_1(u)p_1 + b_2(u)p_2 + b_3(u)p_3$

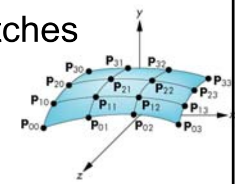
$b_0(u) = -4.5(u-1/3)(u-2/3)(u-1)$
$b_1(u) = 13.5u\ (u-2/3)(u-1)$
$b_2(u) = -13.5u\ (u-1/3)(u-1)$
$b_3(u) = 4.5u\ (u-1/3)(u-2/3)$

Blending Functions





Blending Patches



$$p(u,v) = \sum_{i=o}^{3} \sum_{j=0}^{3} c_{ij} u^i v^j$$

$$p(u,v) = \sum_{i=o}^{3} \sum_{j=0}^{3} b_i(u) b_j(v) p_{ij}$$

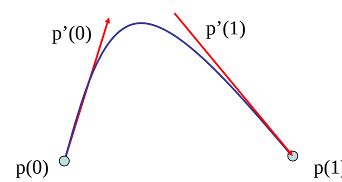
Each $b_i(u)b_j(v)$ is a blending patch

Shows that we can build and analyze surfaces from our knowledge of curves

Hermite Curves and Surfaces

- How can we get around the limitations of the interpolating form
  - –Lack of smoothness
  - –Discontinuous derivatives at join points
- We have four conditions (for cubics) that we can apply to each segment
  - –Use them other than for interpolation
  - –Need only come close to the data

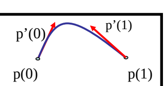
Hermite Form



Use two interpolating conditions and two derivative conditions per segment

Ensures continuity and first derivative continuity between segments

Equations



$p(u) = c_0+uc_1+u^2c_2+u^3c_3$

Interpolating conditions are the same at ends

$p(0) = p_0 = c_0$
$p(1) = p_3 = c_0+c_1+c_2+c_3$

Differentiating we find $p'(u) = c_1+2uc_2+3u^2c_3$

Evaluating at end points

$p'(0) = p'_0 = c_1$
$p'(1) = p'_3 = c_1+2c_2+3c_3$

$$\mathbf{q} = \begin{bmatrix} p_0 \\ p_3 \\ p'_0 \\ p'_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 \end{bmatrix} \mathbf{c}$$

Matrix Form

$$\mathbf{q} = \begin{bmatrix} p_0 \\ p_3 \\ p'_0 \\ p'_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 \end{bmatrix} \mathbf{c}$$

Solving, we find $\mathbf{c}=\mathbf{M}_H\mathbf{q}$ where $\mathbf{M}_H$ is the Hermite matrix

$$\mathbf{M}_H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{bmatrix}$$

Blending Polynomials

$p(u) = \mathbf{b}(u)^T\mathbf{q}$

$p(u) = u^T\mathbf{M}_H\mathbf{q}$

$$\mathbf{b}(u) = \begin{bmatrix} 2u^3-3u^2+1 \\ -2u^3+3u^2 \\ u^3-2u^2+u \\ u^3-u^2 \end{bmatrix} \qquad \mathbf{M}_H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{bmatrix}$$

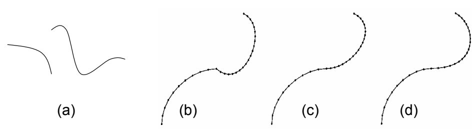
Although these functions are smooth, the Hermite form is not used directly in Computer Graphics and CAD because we usually have control points but not derivatives

However, the Hermite form is the basis of the Bezier form

Parametric and Geometric Continuity

- We can require the derivatives of x, y,and z to each be continuous at join points (*parametric continuity*)
- Alternately, we can only require that the tangents of the resulting curve be continuous (*geometry continuity*)
- The latter gives more flexibility as we have need satisfy only two conditions rather than three at each join point

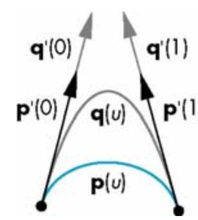
Continuity



- A) Non-continuous
- B) C^0-continuous
- C) G^1-continuous
- D) C^1-continuous
- (C^2-continuous)

Example

- Here the p and q have the same tangents at the ends of the segment but different derivatives
- Generate different Hermite curves
- This techniques is used in drawing applications



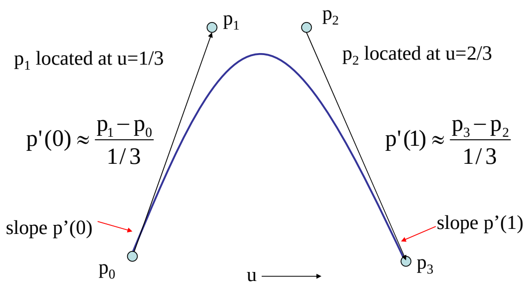
Higher Dimensional Approximations

- The techniques for both interpolating and Hermite curves can be used with higher dimensional parametric polynomials
- For interpolating form, the resulting matrix becomes increasingly more ill-conditioned and the resulting curves less smooth and more prone to numerical errors
- In both cases, there is more work in rendering the resulting polynomial curves and surfaces

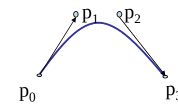
Bezier's Idea

- In graphics and CAD, we do not usually have derivative data
- Bezier suggested using the same 4 data points as with the cubic interpolating curve to approximate the derivatives in the Hermite form

Approximating Derivatives



Equations



$p(u) = c_0+uc_1+u^2c_2+u^3c_3$

Interpolating conditions are the same

$$p(0) = p_0 = c_0$$
$$p(1) = p_3 = c_0+c_1+c_2+c_3$$

Approximating derivative conditions

$$p'(0) = 3(p_1- p_0) = c_0$$
$$p'(1) = 3(p_3- p_2) = c_1+2c_2+3c_3$$

Solve four linear equations for $\mathbf{c}=\mathbf{M}_B\mathbf{p}$

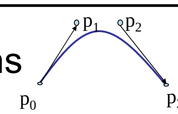
Bezier Matrix

$$\mathbf{M}_B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}$$

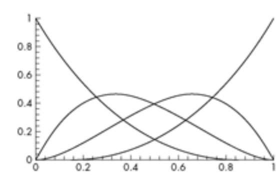
$$p(u) = \mathbf{u}^T\mathbf{M}_B\mathbf{p} = \mathbf{b}(u)^T\mathbf{p}$$

blending functions

Blending Functions



$$\mathbf{b}(u) = \begin{bmatrix} (1-u)^3 \\ 3u(1-u)^2 \\ 2u^2(1-u) \\ u^3 \end{bmatrix}$$



Note that all zeros are at 0 and 1 which forces the functions to be smoother over (0,1)

Smoother because the curve stays inside the convex hull, and therefore does not have room to fluctuate so much.

Bernstein Polynomials

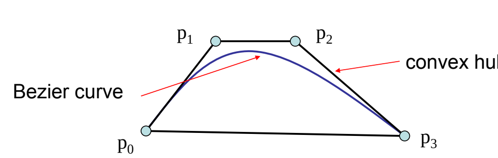
- The blending functions are a special case of the Bernstein polynomials

$$b_{kd}(u) = \frac{d!}{k!(d-k)!} u^k (1-u)^{d-k}$$

- These polynomials give the blending polynomials for any degree Bezier form
  - All zeros at 0 and 1
  - For any degree they all sum to 1
  - They are all between 0 and 1 inside [0,1]

Convex Hull Property

- All weights within [0,1] ensures that all Bezier curves lie in the convex hull of their control points
- Hence, even though we do not interpolate all the data, we cannot be too far away

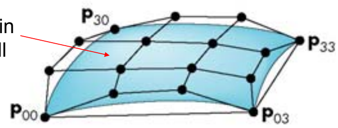


Bezier Patches

Using same data array $\mathbf{P}=[p_{ij}]$ as with interpolating form

$$p(u,v)=\sum_{i=0}^{3}\sum_{j=0}^{3} b_i(u)\, b_j(v)\, p_{ij} = u^T \mathbf{M}_B \mathbf{P} \mathbf{M}_B^T v$$

Patch lies in convex hull



Analysis

- Although the Bezier form is much better than the interpolating form, we have the derivatives are not continuous at join points
- What shall we do to solve this?

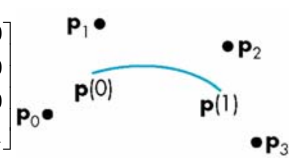
B-Splines

- Basis splines: use the data at $\mathbf{p}=[p_{i-2}\ p_{i-1}\ p_i\ p_{i+1}]^T$ to define curve only between p_{i-1} and p_i
- Allows us to apply more continuity conditions to each segment
- For cubics, we can have continuity of function, first and second derivatives at join points

Cubic B-spline

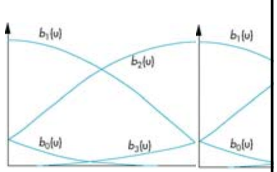
$p(u) = \mathbf{u}^T\mathbf{M}_S\mathbf{p} = \mathbf{b}(u)^T\mathbf{p}$

$$\mathbf{M}_S = \begin{bmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}$$

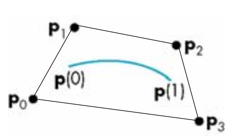


Blending Functions

$$\mathbf{b}(u) = \frac{1}{6}\begin{bmatrix} (1-u)^3 \\ 4-6u^2+3u^3 \\ 1+3u+3u^2-3u^2 \\ u^3 \end{bmatrix}$$



convex hull property



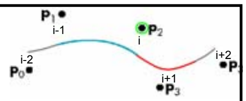
Splines and Basis

- If we examine the cubic B-spline from the perspective of each control (data) point, each interior point contributes (through the blending functions) to four segments
- We can rewrite p(u) in terms of the data points as

$$p(u)=\sum B_i(u)\, p_i$$

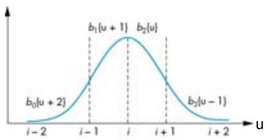
defining the basis functions $\{B_i(u)\}$

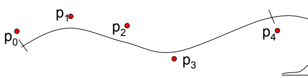
Basis Functions

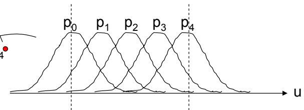


In terms of the blending polynomials

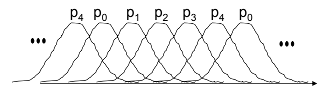
$$B_i(u) = \begin{cases} 0 & u < i-2 \\ b_0(u+2) & i-2 \le u < i-1 \\ b_1(u+1) & i-1 \le u < i \\ b_2(u) & i \le u < i+1 \\ b_3(u-1) & i+1 \le u < i+2 \\ 0 & u \ge i+2 \end{cases}$$

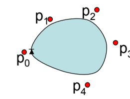






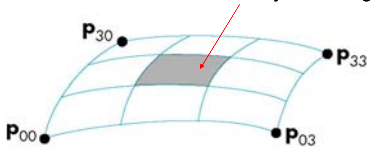
Ett till exempel





B-Spline Patches

$$p(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} b_i(u)\, b_j(v)\, p_{ij} = u^T \mathbf{M}_S \mathbf{P} \mathbf{M}_S^T v$$



Generalizing Splines

- We can extend to splines of any degree
- Data and conditions to not have to be given at equally spaced values (the *knots*)
  - –Nonuniform and uniform splines
  - –Can have repeated knots
    - Can force spline to interpolate points
- (Cox-deBoor recursion gives method of evaluation (also known as deCasteljau-recursion, see page 597 for details))

NURBS

- Nonuniform Rational B-Spline curves and surfaces add a fourth variable w to x,y,z
  - –Can interpret as weight to give more importance to some control data
  - –Can also interpret as moving to homogeneous coordinate
- Requires a perspective division
  - –NURBS act correctly for perspective viewing
- Quadrics are a special case of NURBS

NURBS

B-Splines: (sammanfattning för jämförelse)

Givet: n+1 punkter $\mathbf{P}_0, \mathbf{P}_1, \ldots, \mathbf{P}_n$.

Man kan skriva:

$$\boldsymbol{P}_i(t) = B_{i-1}(t)\boldsymbol{P}_{i-1} + B_i(t)\boldsymbol{P}_i + B_{i+1}(t)\boldsymbol{P}_{i+1} + B_{i+2}(t)\boldsymbol{P}_{i+2}$$

där

$$B(t) = \begin{cases} \frac{1}{6}(2-|t|)^3 & 1 \le |t| \le 2 \\ \frac{1}{6}[1+3(1-|t|)+3(1-|t|)^2-3(1-|t|)^3] & |t| \le 1 \\ 0 & 2 \le |t| \end{cases}$$

NURBS

NURBS: Basfunktionen $B_i(t)$, $0 \le i \le n$, bestämd av **skarvföljden** (eng. knot vector) $[t_i-2, t_i-1, t_i, t_i+1, t_i+2]$. M a o "centrerad" kring t_i och 0 utanför intervallet $[t_i-2, t_i+2]$.

För B-Splines hade vi: $p(u) = \sum B_i(u)\, p_i$

För NURBS kan vi även ange vikter för varje punkt. Motsvarande NURBS-approximation blir:

$$P_i(t) = \frac{\sum_{j=i-1}^{i+2} w_j B_j(t) P_j}{\sum_{j=i-1}^{i+2} w_j B_j(t)}$$

Man dividerar med summan av vikterna för att genomsnittsvikten hos punkterna skall bli 1. Annars inför man nämligen en förskjutning av kurvan - vilket iofs kan vara "kul" men oftast inte önskvärt.

NURBS

- Concider a control point in 3 dimensions:
  $\mathbf{p}_i = [x_i, y_i, z_i]$
- The weighted homogeneous-coordinate is:

$$\mathbf{q}_i = w_i \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix}$$

- The idea is to use the weights w_i to increase or decrease the importance of a particular control point

NURBS

- The first three components of the resulting spline are simply the B-spline representation of the weighted points:

$$\mathbf{q}(u) = \begin{bmatrix} x(u) \\ y(u) \\ z(u) \end{bmatrix} = \sum_{i=0}^{n} B_{i,d}(u) w_i \mathbf{p}_i$$

- The w-component is the scalar B-spline polynomial derived from the set of weights:

$$w(u) = \sum_{i=0}^{n} B_{i,d}(u) w_i$$

Man dividerar med summan av vikterna för att genomsnittsvikten hos punkterna skall bli 1. Annars inför man nämligen en förskjutning av kurvan - vilket iofs kan vara "kul" men oftast inte önskvärt.

NURBS

- The w-component may not be equal to 1.
- Thus we must do a perpsective division to get the three-dimensional points:

$$\mathbf{p}(u) = \frac{1}{w(u)} \mathbf{q}(u) = \frac{\sum_{i=0}^{n} B_{i,d} w_i \mathbf{p}(i)}{\sum_{i=0}^{n} B_{i,d} w_i}$$

- Each component of **p**(u) is now a rational function in *u*, and because we have not restricted the knots (the knots does not have to be uniformly distributed), we have derived a **nonuniform rational B-spline (NURBS)** curve

NURBS

- If we apply an affine transformation to a B-spline curve or surface, we get the same function as the B-spline derived from the transformed control points.
- Because perspective transformations are not affine, most splines will not be handled correctly in perspective viewing.
- However, the perspective division embedded in the NURBS ensures that NURBS curves are handled correctly in perspective views.
- Quadrics can be shown to be a special case of quadratic NURBS curve; thus, we can use a single modeling method, NURBS curves, for the most widely used curves and surfaces

NURBS and Subdivision Surfaces

Det finns två huvudmetoder för konstruktion av kurvor och ytor:

- **Splines**: Olika former av **splines**, framför allt B-splines och NURBS. Utgående från ett antal punkter sätts ett uttryck för kurvan eller ytan upp på parameterform. Kurvan ritas utifrån denna parameterframställning. Upplevs av de flesta som rätt matematiskt och krångligt. Stöds av OpenGL. Behandlas i det separata pappret *Kurv- och ytapproximation*, samt i OpenGL-häftet, avsnitt 24.
- **Uppdelningsmetoder** (eng. subdivision). Ytligt sett mera praktiskt. Utgående från ett antal punkter inför man successivt nya punkter och modifierar samtidigt de gamla. Därefter ritas kurvan genom ett polygontåg genom punkterna. Man får automatiskt sina objekt i flera upplösningar. Att analysera metoderna matematiskt är däremot inte lätt. Inget direkt stöd i OpenGL. Liksom alla kommersiella modelleringsprogram har *Blender* och *Art Of Illusion* verktyg (om än begränsade) för uppdelning. Användningsområdet är ytor. Blev populära i slutet av 1990-talet.

I båda fallen skiljer man på **interpolerande** kurvor/ytor och **approximerande** kurvor/ytor. Vi ägnar oss mest åt den senare typen. Kurvan/ytan går då inte säkert igenom de olika styrpunkter som man utgår ifrån.

Subdivision surface

Rendering Curves and Surfaces

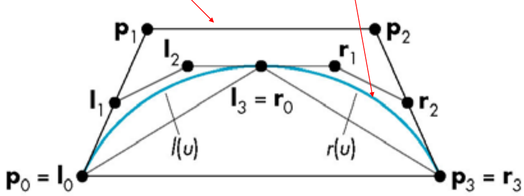
deCasteljau[1] Recursion

- We can use the convex hull property of Bezier curves to obtain an efficient recursive method that does not require any function evaluations
  - –Uses only the values at the control points
- Based on the idea that "any polynomial and any part of a polynomial is a Bezier polynomial for properly chosen control data"

[1] Paul de Casteljau och Pierre Bezier var bilingenjörer. Den förre vid Peugot och den andre vid Renault. Båda jobbade med Bezier-kurvor utan att känna till varandras arbete.

Splitting a Cubic Bezier

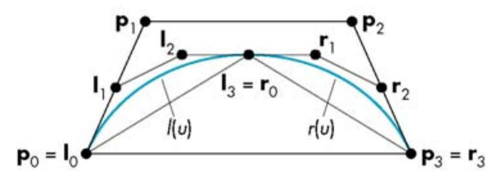
p_0, p_1, p_2, p_3 determine a cubic Bezier polynomial and its convex hull



Consider left half $l(u)$ and right half $r(u)$

$l(u)$ and $r(u)$

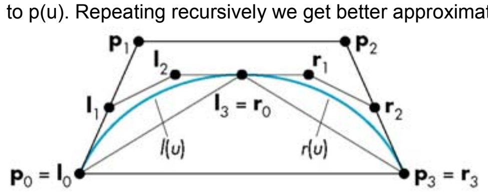
Since $l(u)$ and $r(u)$ are Bezier curves, we should be able to find two sets of control points $\{l_0, l_1, l_2, l_3\}$ and $\{r_0, r_1, r_2, r_3\}$ that determine them



Convex Hulls

$\{l_0, l_1, l_2, l_3\}$ and $\{r_0, r_1, r_2, r_3\}$ each have a convex hull that that is closer to p(u) than the convex hull of $\{p_0, p_1, p_2, p_3\}$ This is known as the *variation diminishing property*.

The polyline from l_0 to l_3 $(= r_0)$ to r_3 is an approximation to p(u). Repeating recursively we get better approximations.



Equations

Start with Bezier equations $p(u)=\mathbf{u}^T M_B \mathbf{p}$

$l(u)$ must interpolate $p(0)$ and $p(1/2)$

$l(0) = l_0 = p_0$
$l(1) = l_3 = p(1/2) = 1/8(p_0 +3 p_1 +3 p_2 + p_3)$

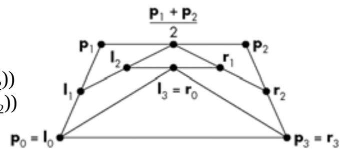
Matching slopes, taking into account that $l(u)$ and $r(u)$ only go over half the distance as $p(u)$

$l'(0) = 3(l_1 - l_0) = p'(0) = 3/2(p_1 - p_0)$
$l'(1) = 3(l_3 – l_2) = p'(1/2) = 3/8(- p_0 - p_1+ p_2 + p_3)$

Symmetric equations hold for $r(u)$

Efficient Form

$l_0 = p_0$
$r_3 = p_3$
$l_1 = ½(p_0 + p_1)$
$r_1 = ½(p_2 + p_3)$
$l_2 = ½(l_1 + ½(p_1 + p_2))$
$r_1 = ½(r_2 + ½(p_1 + p_2))$
$l_3 = r_0 = ½(l_2 + r_1)$



Requires only shifts and adds!

Every Curve is a Bezier Curve

- We can render a given polynomial using the recursive method if we find control points for its representation as a Bezier curve
- Suppose that p(u) is given as an interpolating curve with control points **q**

$$p(u)=\mathbf{u}^T\mathbf{M}_I\mathbf{q}$$

- There exist Bezier control points **p** such that

$$p(u)=\mathbf{u}^T\mathbf{M}_B\mathbf{p}$$

- Equating and solving, we find $\mathbf{p}=\mathbf{M}_B^{-1}\mathbf{M}_I$

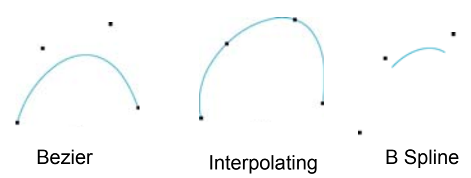
Matrices

Interpolating to Bezier $$\mathbf{M}_B^{-1}\mathbf{M}_I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -\frac{5}{6} & 3 & -\frac{3}{2} & \frac{1}{3} \\ \frac{1}{3} & -\frac{3}{2} & 3 & -\frac{5}{6} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

B-Spline to Bezier $$\mathbf{M}_B^{-1}\mathbf{M}_S = \begin{bmatrix} 1 & 4 & 1 & 0 \\ 0 & 4 & 2 & 0 \\ 0 & 2 & 4 & 0 \\ 0 & 1 & 4 & 1 \end{bmatrix}$$

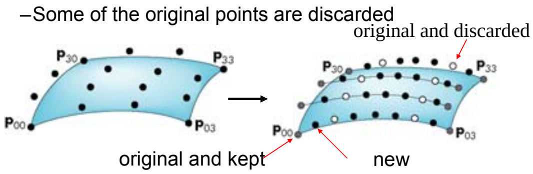
Example

These three curves were all generated from the same original data using Bezier recursion by converting all control point data to Bezier control points

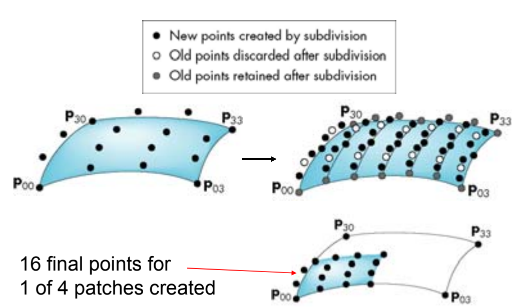


Surfaces

- Can apply the recursive method to surfaces if we recall that for a Bezier patch curves of constant u (or v) are Bezier curves in u (or v)
- First subdivide in u
  - Process creates new points
  - Some of the original points are discarded



Second Subdivision



THE END

- OBS!!!
- INGEN FÖRELÄSNING NU PÅ FREDAG 6:e oktober.