

CSI422/502 Lecture 34 Curved models..

Images.

HB Sect. 8-4, 8-5: Surfaces defined by (1) an equation $F(x, y, z) = 0$ or
(2) parameterized formulas $x(u, v)$, $y(u, v)$, $z(u, v)$.

Torus:

$$\left(\frac{\sqrt{x^2 + y^2} - R}{r} \right)^2 + \left(\frac{z}{r} \right)^2 = 1$$

$$x(\phi, \theta) = (R + r \cos \phi) \cos \theta$$

$$y(\phi, \theta) = (R + r \cos \phi) \sin \theta$$

$$z(\phi, \theta) = r \sin \phi$$

See the funny Superquadratics in 8-5.

HB Sect. 8-6: Some approximations by Meshes:

For project 4, you may use `glutSolidTorus` for copied-torus-code credit.

(Grad. students must derive torus code on paper for full credit there.)

HB Sect. 8-7 **Bloppy Objects**: Case of $F(x, y, z) = 0$ defining the surface.

Examples:

Gaussian bumps:

$$f(x, y, z) = \sum_{\text{Centers } k=1}^n b_k e^{-a_k \left((x-x_k)^2 + (y-y_k)^2 + (z-z_k)^2 \right)} - T = 0$$

Meta-ball (in Blender?) uses quadratic densities..

Splines Today..

HB Sect. 8-19 **Sweep Reps**: Observe Blender's **Extrude** and **Spin**.

HB Sect. 8-23 **Fractal Models** (really cool!)

HB Sect. 8-24 **Shape Grammars** (evaluated recursively)

HB Sects. 8-25, 8-26 **Particle systems** and other **Physically Based**

Models (springs, membranes, cloth.)

Splines.

Extension of Interpolation: Connect a sequence of given points by a curve with enough smoothness.

Approximation: Given a sequence of **control points**, make a smooth enough curve that is close enough to them.

Spline curves are defined by **polynomial vector-valued** functions like

$$\mathbf{P}(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix} = \begin{bmatrix} a_x u^3 + b_x u^2 + c_x u + d_x \\ a_y u^3 + b_y u^2 + c_y u + d_y \\ a_z u^3 + b_z u^2 + c_z u + d_z \end{bmatrix}$$

because (1) these polys are easy to figure out from the given interpolation or control points and (2) polys are easy to evaluate for plotting/rasterizing. (“NURBS” uses ratios of polynomials)

Splines in action:

1. **Spline surfaces:** Blender features (1) Mesh subdivision (activate via Mesh panel) and (2) Bezier surfaces (create via Add operation in Object mode).

2. **Splines for interpolated animation between key frames:** See HB Chap. 13.

Blender: (1) Create an object. (2) Select it. (3) Activate **Insert key** (i) (4) Increase frame number by 20 or so. (5) Transform the object. (6) Repeat steps (3)-(5) as you like.

Now display the IPO window. Left and Right Arrow presses in this window will preview the animation.

You can edit the spline curves in the IPO window.

You can edit the **tangent line** where two curve sections meet.

An n -th degree polynomial function has $n + 1$ coefficients

$$f(u) = f_3u^3 + f_2u^2 + f_1u + f_0 \text{ Here is a cubic or degree 3 poly.}$$

The polynomial is uniquely determined by its values at $n + 1$ distinct values of u .

(A bad way) to interpolate any $n + 1$ points by an n -degree polynomial:

Suppose the given points are $(u_0, y_0), (u_1, y_1), \dots, (u_n, y_n)$. (Lagrange's Cleverness:)

$$P_k(u) = \frac{(u - u_0)(u - u_1) \cdots (u - u_n)}{(u_k - u_0)(u_k - u_1) \cdots (u_k - u_n)}$$

where the numerator **OMITS** $(u - u_k)$ and denominator **OMITS** $(u_k - u_k)$.

$$P_k(u_k) = 1 \text{ (WHY??) and } P_k(u_j) = 0 \text{ for } j \neq k \text{ (WHY???)}$$

so the points are interpolated by

$$f(u) = y_0P_0(u) + y_1P_1(u) + \cdots + y_nP_n(u). \text{ WHY???)}$$

The results are bad because

1. Lagrange polynomials fluctuate widely between the interpolated points.
2. Calculating high degree polynomials takes time and might be unstable.

So, we use **piece-wise polynomial functions**

For example, **Bezier cubic spline curves**.

A n -th degree polynomial function F is (obviously!) uniquely specified by the $n + 1$ coefficients a_k of u^k

$$F(u) = a_n u^n + a_{n-1} u_{n-1} + \cdots + a_1 u + a_0$$

It is also uniquely specified by the $n + 1$ coefficients y_k of the $n + 1$

Lagrangian basis (or blending) functions L_k

$$F(u) = y_0 L_0(u) + y_1 L_1(u) + \cdots + y_n L_n(u)$$

where

$$L_k(u) = \frac{(u - u_0)(u - u_1) \cdots (u - \widetilde{u_k}) \cdots (u - u_n) [(u - u_k) \text{ is omitted}]}{(u_k - u_0)(u_k - u_1) \cdots (u_k - \widetilde{u_k}) \cdots (u_k - u_n) [(u_k - u_k) \text{ is omitted}]}$$

so

$$L_k(u_k) = 1 \quad \text{and} \quad L_k(u_i) = 0 \text{ with } i \neq k$$

Therefore, the coefficients in Lagrange's expression are the function values!

$$y_k = F(u_k) \text{ for } k = 0, 1, \dots, n$$

Lagrange interpolating polynomials have degree n for $n + 1$ points (high!)

They (1) vary wildly between interpolated points and (2) are computationally unstable when expanded out.

A **spline** is a piecewise polynomial function that satisfies specified **continuity conditions** at the boundary points where two adjacent pieces meet.

In practice, the user specifies a sequence (or a mesh for surfaces in 3D) of **control points**.

Each polynomial section is determined by *some* ($n + 1$ for n -degree) of the control points.

Adjacent sections

- touch along their borders (zero-order continuity), and may
- smoothly connect (first and higher continuity)

BECAUSE adjacent they share some control points!

Some old splines... (not fully determined by control points)

Natural cubic: n sections each a cubic polynomial function, interpolate the $n + 1$ points where they meet at u_k , and there, their first derivatives are equal (1st order parametric continuity) and their second derivatives are equal (2nd order parametric continuity). 4 more boundary conditions determine the whole spline.

Disadvantage: User loses local control of curve shape.

Hermite cubic:

For each section, the user specifies the two endpoints.

For each boundary point, the user specifies the common first derivative at that boundary point.

Easy way to express the k -th section:

$$\mathbf{P}(u) = \mathbf{p}_k H_0(u) + \mathbf{p}_{k+1} H_1(u) + \mathbf{Dp}_k H_2(u) + \mathbf{Dp}_{k+1} H_2(u)$$

$$H_0(u) = 2u^3 - 3u^2 + 1$$

$$H_1(u) = -2u^3 + 3u^2$$

$$H_2(u) = u^3 - 2u^2 + u$$

$$H_3(u) = u^3 - u^2$$

What's cool here...

For H_0 and H_1 :

- Derivatives at both endpoints ($u = 0, 1$) are 0.
- $H_0(0) = 1, H_0(1) = 0; H_1(0) = 0, H_1(1) = 1.$

For H_2 and H_3 :

- Values at both endpoints are 0.
- $H_2'(0) = 1, H_2'(1) = 0; H_3'(0) = 0, H_3'(1) = 1.$

So the expression works..

Catmull-Rom (or Overhauser or cardinal) spline (I think this is the default option for Blender surface “subdivision”.)

Hermite method applied with the *derivatives calculated from adjacent control points and a tension parameter*

So, each section is determined by 4 adjacent control points.

Each section can be expressed by sum of control point coordinates weighted by blending functions:

Section k : $\mathbf{P}(u) = \mathbf{p}_{k-1}CAR_0(u) + \mathbf{p}_kCAR_1(u) + \mathbf{p}_{k+1}CAR_2(u) + \mathbf{p}_{k+2}CAR_3(u)$

Section k interpolates between \mathbf{p}_k and \mathbf{p}_{k+1} .

With n sections, $k = 1$ to $k = n$, $n + 1$ section endpoints $\mathbf{p}_1, \dots, \mathbf{p}_{n+1}$ are interpolated but two more, \mathbf{p}_0 and \mathbf{p}_{n+2} are needed to set the tangents at the whole spline’s endpoints.

Bezier

Binomial theorem (for $n = 3$)

$$\begin{aligned}(A + B)^3 &= \binom{3}{0} A^0 B^3 + \binom{3}{1} A^1 B^2 + \binom{3}{2} A^2 B^1 + \binom{3}{3} A^3 B^0 \\(A + B)^3 &= (A + B)(A + B)(A + B) \\&= 1BBB + 3ABB + 3AAB + 1AAA\end{aligned}$$

Now, (clever!) substitute $A \leftarrow u$ and $B \leftarrow (1 - u)$:

$$(A + B) = ((u) + (1 - u)) = (u + 1 - u) = 1$$

So

$$\begin{aligned}1 &= 1BBB + 3ABB + 3AAB + 1AAA \\1 &= 1(1 - u)^3 + 3u(1 - u)^2 + 3u^2(1 - u) + 1u^3 \\1 &= BEZ_{0,3}(u) + BEZ_{1,3}(u) + BEZ_{2,3}(u) + BEZ_{3,3}(u)\end{aligned}$$

where the Bezier blending functions, aka Bernstein polynomials are given

by the formulas:

$$BEZ_{k,n} = \binom{n}{k} u^k (1-u)^{n-k}. \text{ I illustrated } n = 3$$

An n degree **Bezier spline section** is specified by $n + 1$ control points

$\mathbf{p}_k = (x_k, y_k, z_k)$, $k = 0$ to n by

$$x(u) = \sum_{k=0}^n x_k BEZ_{k,n}(u)$$

$$y(u) = \sum_{k=0}^n y_k BEZ_{k,n}(u)$$

$$z(u) = \sum_{k=0}^n z_k BEZ_{k,n}(u)$$

more succinctly,

$$\mathbf{P}(u) = \sum_{k=0}^n \mathbf{p}_k BEZ_{k,n}(u)$$

What's cool:

$$\mathbf{P}(0) = \mathbf{p}_0$$

$$\mathbf{P}(1) = \mathbf{p}_n$$

$$\mathbf{P}'(0) = n(\mathbf{p}_1 - \mathbf{p}_0)$$

$$\mathbf{P}'(1) = n(\mathbf{p}_n - \mathbf{p}_{n-1})$$

and: For every intermediate point on the curve $P(u)$ for $0 \leq u \leq 1$, $P(u)$ is in the convex hull of the $n + 1$ control points P_k because

$$\text{Each } BEZ_{k,n} = \binom{n}{k} u^k (1-u)^{n-k} = \text{positive}(\text{positive})(\text{positive})$$

and

$$\sum_{k=0}^n \binom{n}{k} u^k (1-u)^{n-k} = (u+1-u)^n = (1)^n = 1$$

Surfaces??? Parametrized by 2 parameters; each section surface is the image of the $[0, 1] \times [0, 1]$ square.

The following surface is determined by the $(m + 1) \times (n + 1)$ control points $\mathbf{P}_{j,k}$:

$$\mathbf{P}(u, v) = \sum_{j=0}^m \sum_{k=0}^n \mathbf{P}_{j,k} B E Z_{j,m}(v) B E Z_{k,m}(u)$$

Different polynomial degrees can be used for the 2 different directions along the surface.

Similar expressions apply for writing sections in term of control points when other (non-Bezier) blending functions are used.

Other applications of the basis idea: **Lossy Data (image) compression**; see **Discrete Cosine transform HB p. 776-7**