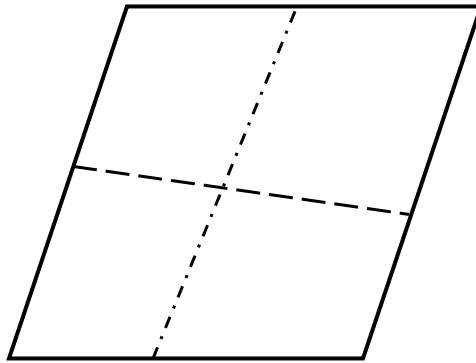


Interpolation and Deformations

A short cookbook



1 600.445 Fall 2000; Updated: October 7, 14
Copyright © R. H. Taylor

Engineering Research Center for Computer Integrated Surgical Systems and Technology



Linear Interpolation

$$\bullet \vec{p}_2 = [40 \ 30 \ 20]^T$$

$$\rho_2 = 20$$

$$\bullet \vec{p}_3 = [20 \ 20 \ 20]^T$$

$$\rho_3 = 10$$

$$\bullet \vec{p}_1 = [10 \ 15 \ 20]^T$$

$$\rho_1 = 5$$

2 600.445 Fall 2000; Updated: October 7, 14
Copyright © R. H. Taylor

Engineering Research Center for Computer Integrated Surgical Systems and Technology

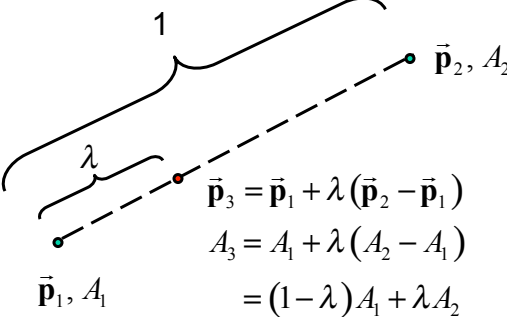


Linear Interpolation

$$\begin{aligned}
 & \bullet \vec{p}_2 = [40 \quad 30 \quad 20]^T \\
 & \quad \vec{q}_2 = \vec{b} \\
 & \bullet \vec{p}_3 = [20 \quad 20 \quad 20]^T \\
 & \quad \vec{q}_3 = \vec{a} + \frac{1}{3}(\vec{b} - \vec{a}) \\
 & \bullet \vec{p}_1 = [10 \quad 15 \quad 20]^T \\
 & \quad \vec{q}_1 = \vec{a}
 \end{aligned}$$



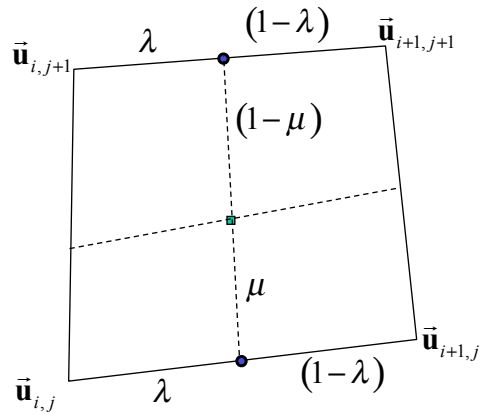
Linear Interpolation



$$\begin{aligned}
 \vec{p}_3 &= \vec{p}_1 + \lambda (\vec{p}_2 - \vec{p}_1) \\
 A_3 &= A_1 + \lambda (A_2 - A_1) \\
 &= (1 - \lambda) A_1 + \lambda A_2 \\
 \lambda &= \frac{(\vec{p}_3 - \vec{p}_1) \bullet (\vec{p}_2 - \vec{p}_1)}{(\vec{p}_2 - \vec{p}_1) \bullet (\vec{p}_2 - \vec{p}_1)}
 \end{aligned}$$



Bilinear Interpolation

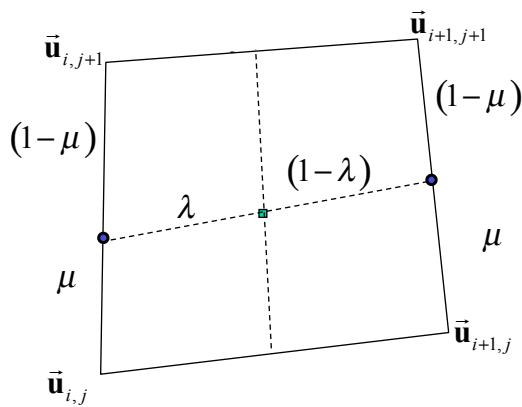


5 600.445 Fall 2000; Updated: October 7, 14
Copyright © R. H. Taylor

Engineering Research Center for Computer Integrated Surgical Systems and Technology



Bilinear Interpolation

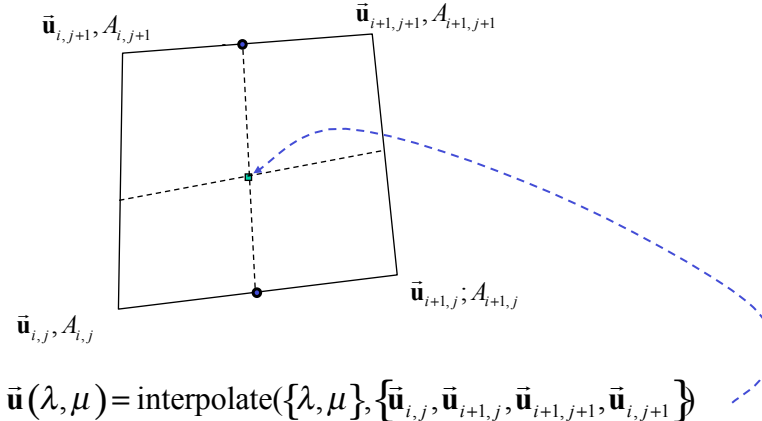


$$\begin{aligned}\bar{u}(\lambda, \mu) &= \lambda (\mu \bar{u}_{i+1,j+1} + (1-\mu) \bar{u}_{i+1,j}) + (1-\lambda) (\mu \bar{u}_{i,j+1} + (1-\mu) \bar{u}_{i,j}) \\ &= \bar{u}_{i,j} + \lambda (\bar{u}_{i+1,j} - \bar{u}_{i,j}) + \mu (\bar{u}_{i,j+1} - \bar{u}_{i,j}) + \lambda \mu (\bar{u}_{i+1,j+1} - \bar{u}_{i,j})\end{aligned}$$

6



Bilinear Interpolation



$$\bar{\mathbf{u}}(\lambda, \mu) = \text{interpolate}(\{\lambda, \mu\}, \{\bar{\mathbf{u}}_{i,j}, \bar{\mathbf{u}}_{i+1,j}, \bar{\mathbf{u}}_{i+1,j+1}, \bar{\mathbf{u}}_{i,j+1}\})$$

$$A(\lambda, \mu) = \text{interpolate}(\{\lambda, \mu\}, \{A_{i,j}, A_{i+1,j}, A_{i+1,j+1}, A_{i,j+1}\})$$

7 600.445 Fall 2000; Updated: October 7, 14
Copyright © R. H. Taylor

Engineering Research Center for Computer Integrated Surgical Systems and Technology



N-linear Interpolation

Let

$$\bar{\Lambda}_N = \{\lambda_1, \dots, \lambda_N\}, \text{ with } 0 \leq \lambda_k \leq 1$$

be a set of interpolation parameters, and let

$$\bar{\mathbf{A}} = \{A_1, \dots, A_{2^N}\}$$

be a set of constants. Then we define:

$$\text{NlinearInterpolate}(\Lambda_N, \mathbf{A}) =$$

$$(1 - \lambda_N) \text{NlinearInterpolate}(\Lambda_{N-1}, \{A_1, \dots, A_{2^{N-1}}\}) \\ + \lambda_N \text{NlinearInterpolate}(\Lambda_{N-1}, \{A_{2^{N-1}+1}, \dots, A_{2^N}\})$$

NOTE: Sometimes in this situation we will use notation

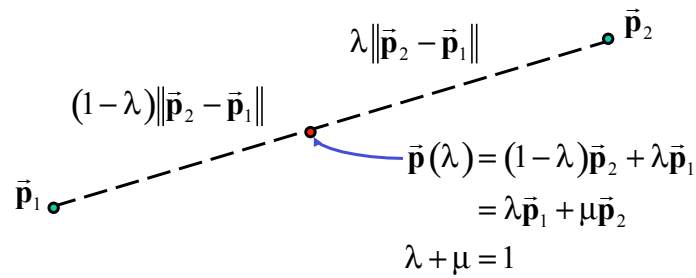
$$A(\bar{\Lambda}_N) = A(\lambda_1, \dots, \lambda_N) \\ = \text{NlinearInterpolate}(\bar{\Lambda}_N, \bar{\mathbf{A}})$$

8 600.445 Fall 2000; Updated: October 7, 14
Copyright © R. H. Taylor

Engineering Research Center for Computer Integrated Surgical Systems and Technology



Barycentric Interpolation

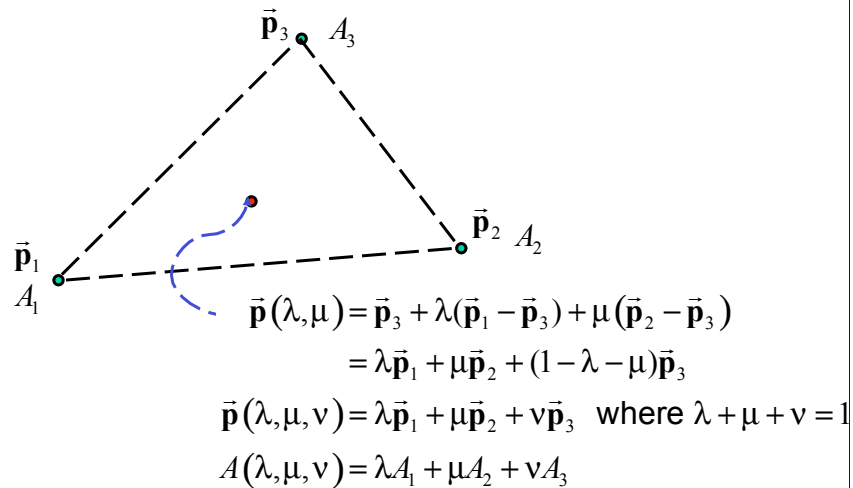


9 600.445 Fall 2000; Updated: October 7, 14
Copyright © R. H. Taylor

Engineering Research Center for Computer Integrated Surgical Systems and Technology



Barycentric Interpolation

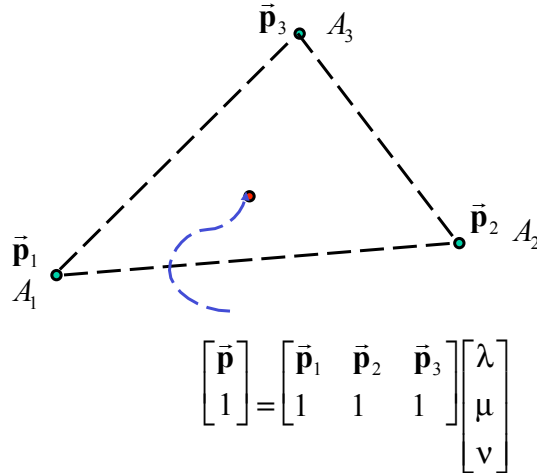


10 600.445 Fall 2000; Updated: October 7, 14
Copyright © R. H. Taylor

Engineering Research Center for Computer Integrated Surgical Systems and Technology



Barycentric Interpolation



11 600.445 Fall 2000; Updated: October 7, 14
Copyright © R. H. Taylor

Engineering Research Center for Computer Integrated Surgical Systems and Technology



Barycentric Interpolation

Let

$$\vec{\Lambda} = \{\lambda_1, \dots, \lambda_N\}, \text{ with } 0 \leq \lambda_k \leq 1 \text{ and } \sum_{k=1}^N \lambda_k = 1$$

be a set of interpolation parameters, and let

$$\vec{A} = \{A_1, \dots, A_{2^N}\}$$

be a set of constants. Then we define:

$$\text{BarycentricInterpolate}(\vec{\Lambda}, \vec{A}) = \vec{\Lambda} \cdot \vec{A} = \sum_{k=1}^N \lambda_k A_k$$

NOTE: Sometimes in this situation we will use notation

$$\mathbf{A}(\Lambda_N) = \mathbf{A}(\lambda_1, \dots, \lambda_N) = \text{BarycentricInterpolate}(\Lambda_N, \mathbf{A})$$

NOTE: This is a special case of barycentric Bezier

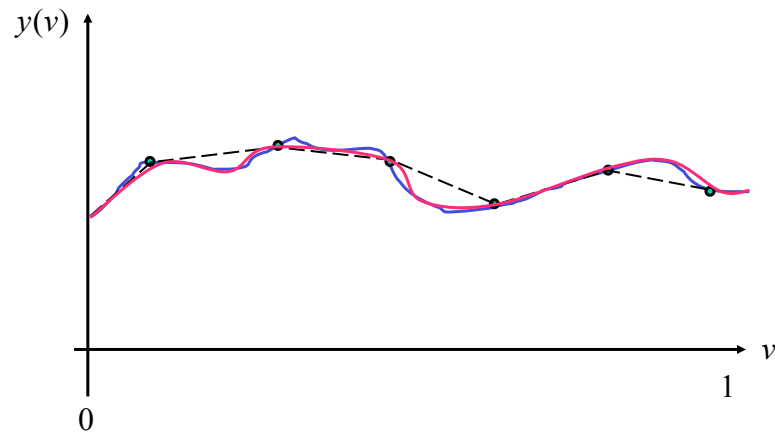
polynomial interpolations (here, 1st degree)

12 600.445 Fall 2000; Updated: October 7, 14
Copyright © R. H. Taylor

Engineering Research Center for Computer Integrated Surgical Systems and Technology



Interpolation of functions



13 600.445 Fall 2000; Updated: October 7, 14
Copyright © R. H. Taylor

Engineering Research Center for Computer Integrated Surgical Systems and Technology



Fitting of interpolation curves

- The discussion below follows (in part)

G. Farin, Curves and surfaces for computer-aided geometric design, a practical guide, Academic Press, Boston, 1990, chapter 10 and pp 281-284.

14 600.445 Fall 2000; Updated: October 7, 14
Copyright © R. H. Taylor

Engineering Research Center for Computer Integrated Surgical Systems and Technology



1-D Interpolation

Given set of known values $\{y_0(v_0), \dots, y_m(v_m)\}$,
find an approximating polynomial $y \cong P(c_0, \dots, c_N; v)$

$$P(c_0, \dots, c_N; v) = \sum_{k=0}^N c_k P_{N,k}(v)$$

Note that many forms of polynomial may be used
for the $P_{N,k}(v)$. One common (not very good) choice
is the power basis:

$$P_{N,k}(v) = v^k$$

Better choices are the Bernstein polynomials and the
b-spline basis functions, which we will discuss in
a moment



1-D Interpolation

Given set of known values $\{y_0(v_0), \dots, y_m(v_m)\}$,
find an approximating polynomial $y \cong P(c_0, \dots, c_N; v)$

$$P(c_0, \dots, c_N; v) = \sum_{k=0}^N c_k P_{N,k}(v)$$

To do this, solve:

$$\begin{bmatrix} P_{N,0}(v_0) & \cdots & P_{N,N}(v_0) \\ \vdots & \ddots & \vdots \\ P_{N,0}(v_m) & \cdots & P_{N,N}(v_m) \end{bmatrix} \begin{bmatrix} c_0 \\ \vdots \\ c_N \end{bmatrix} \cong \begin{bmatrix} y_0 \\ \vdots \\ y_m \end{bmatrix}$$



Bezier and Bernstein Polynomials

$$P(c_0, \dots, c_N; v) = \sum_{k=0}^N c_k \binom{N}{k} (1-v)^{N-k} v^k$$

$$= \sum_{k=0}^N c_k B_{N,k}(v)$$

where $B_{N,k}(v) = \binom{N}{k} (1-v)^{N-k} v^k$

- Excellent numerical stability for $0 < v < 1$
- There exist good ways to convert to more conventional power basis

17 600.445 Fall 2000; Updated: October 7, 14
Copyright © R. H. Taylor

Engineering Research Center for Computer Integrated Surgical Systems and Technology



Barycentric Bezier Polynomials

$$P(c_0, \dots, c_N; u, v) = \sum_{k=0}^N c_k \binom{N}{k} u^{N-k} v^k$$

$$= \sum_{k=0}^N c_k B_{N,k}(u, v)$$

where $B_{N,k}(u, v) = \binom{N}{k} u^{N-k} v^k \quad u+v=1$

- Excellent numerical stability for $0 < u, v < 1$
- There exist good ways to convert to more conventional power basis

18 600.445 Fall 2000; Updated: October 7, 14
Copyright © R. H. Taylor

Engineering Research Center for Computer Integrated Surgical Systems and Technology

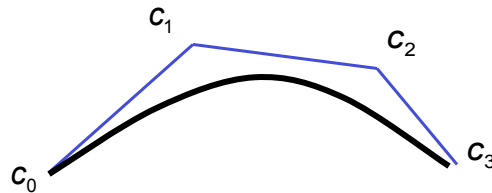


Bezier Curves

Suppose that the coefficients \vec{c}_j are multi-dimensional vectors (e.g., 2D or 3D points). Then the polynomial

$$P(\vec{c}_0, \dots, \vec{c}_N; v) = \sum_{k=0}^N \vec{c}_k B_{N,k}(v)$$

computed over the range $0 \leq v \leq 1$ generates a Bezier curve with control vertices \vec{c}_j .



19 600.445 Fall 2000; Updated: October 7, 14
Copyright © R. H. Taylor

Engineering Research Center for Computer Integrated Surgical Systems and Technology



Bezier Curves: de Casteljau Algorithm

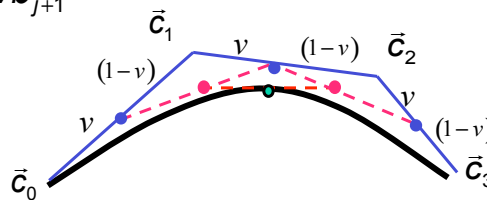
Given coefficients \vec{c}_j , Bezier curves can be generated recursively by repeated linear interpolation:

$$P(\vec{c}_0, \dots, \vec{c}_N; v) = b_0^N$$

where

$$b_j^0 = \vec{c}_j$$

$$b_j^k = (1-v)b_j^{k-1} + vb_{j+1}^{k-1}$$



20 600.445 Fall 2000; Updated: October 7, 14
Copyright © R. H. Taylor

Engineering Research Center for Computer Integrated Surgical Systems and Technology



Iterative Form of deCasteljau Algorithm

Step 1: $b_j \leftarrow c_j$ for $0 \leq j \leq N$

Step 2: for $k \leftarrow 1$ step 1 until $k = N$ do
 for $j \leftarrow 0$ step 1 until $j = N - k$ do
 $b_j \leftarrow (1-v)b_j + vb_{j+1}$

Step 3: return b_0



Advantages of Bezier Curves

- Numerically very robust
- Many nice mathematical properties
- Smooth
- “Global” (may be viewed as a disadvantage)



B-splines

Given

coefficient values $\bar{\mathbf{C}} = \{\bar{c}_0, \dots, \bar{c}_{L+D-1}\}$

"knot points" $\bar{\mathbf{u}} = \{u_0, \dots, u_{L+2D-2}\}$ with $u_i \leq u_{i+1}$

D = "degree" of desired B-spline

Can define an interpolated curve $P(\bar{\mathbf{C}}, \bar{\mathbf{u}}; u)$ on $u_{D-1} \leq u < u_{L+D-1}$

Then

$$P(\bar{\mathbf{C}}; u) = \sum_{j=0}^{L+D-1} \bar{c}_j N_j^D(u)$$

where $N_j^D(u)$ are B-spline basis polynomials (discussed later)



B-Spline Polynomials

Some useful references include

- <http://en.wikipedia.org/wiki/B-spline>
- <http://vision.ucsd.edu/~kbranson/research/bsplines/bsplines.pdf>
- <http://scholar.lib.vt.edu/theses/available/etd-100699-171723/>
- https://www.cs.drexel.edu/~david/Courses/CS430/Lectures/L-09_BSplines_NURBS.pdf
- http://www.stat.columbia.edu/~ruf/ruf_bspline.pdf



B-spline polynomials & B-spline basis functions

Given $\bar{\mathbf{C}}, \bar{\mathbf{u}}, D$ as before

$$P(\bar{\mathbf{C}}, \bar{\mathbf{u}}; u) = \sum_{j=0}^{L+D-1} \bar{c}_j N_j^D(u)$$

where

$$N_j^0(u) = \begin{cases} 1 & u_{j-1} \leq u \leq u_j \\ 0 & \text{Otherwise} \end{cases}$$

$$N_j^k(u) = \frac{u - u_{j-1}}{u_{j+k-1} - u_{j-1}} N_j^{k-1}(u) + \frac{u_{j+k} - u}{u_{j+k} - u_j} N_{j+1}^{k-1}(u) \text{ for } k > 0$$



B-Spline Polynomials

For a B-spline polynomial

$$P(\bar{\mathbf{C}}, \bar{\mathbf{u}}; t) = \sum_{j=0}^{L+D-1} \bar{c}_j N_j^D(\bar{\mathbf{u}}, t)$$

the basis functions $N_j^D(\bar{\mathbf{u}}, t)$ are a function of the degree of the polynomial and the vector $\bar{\mathbf{u}} = [u_0, \dots, u_n]$ of "knot points". The polynomial is "uniform" if the distance between knot points is evenly spaced and "non-uniform" otherwise.



deBoor Algorithm

Given \mathbf{u} , \mathbf{c} , D as before, can evaluate $P(\mathbf{c}, \mathbf{u}; u)$

recursively as follows:

Step 1: Determine index i such that $u_i \leq u < u_{i+1}$

Step 2: Determine multiplicity r such that

$$u_{i-r} = u_{i-r+1} = \dots = u_i$$

Step 3: Set $\bar{\mathbf{d}}_j^0 = \mathbf{c}_j$ for $i - D + 1 \leq j \leq i + 1$

Step 4: Compute $P(\mathbf{c}, \mathbf{u}; u) = d_{i+1}^{D-r}$ recursively, where

$$\bar{\mathbf{d}}_j^k = \frac{u_{j+D-k} - u}{u_{j+D-k} - u_{j-1}} \bar{\mathbf{d}}_{j-1}^{k-1} + \frac{u - u_{j-1}}{u_{j+D-k} - u_{j-1}} \bar{\mathbf{d}}_j^{k-1} = \frac{\alpha_j^k \bar{\mathbf{d}}_{j-1}^{k-1}}{\gamma_j^k} + \frac{\beta_j^k \bar{\mathbf{d}}_j^{k-1}}{\gamma_j^k}$$



deBoor Algorithm: Example $D=3, r=0$

$$\bar{\mathbf{d}}_j^k = \frac{u_{j+D-k} - u}{u_{j+D-k} - u_{j-1}} \bar{\mathbf{d}}_{j-1}^{k-1} + \frac{u - u_{j-1}}{u_{j+D-k} - u_{j-1}} \bar{\mathbf{d}}_j^{k-1} = \frac{\alpha_j^k \bar{\mathbf{d}}_{j-1}^{k-1}}{\gamma_j^k} + \frac{\beta_j^k \bar{\mathbf{d}}_j^{k-1}}{\gamma_j^k}$$

$$\bar{\mathbf{d}}_{i+1}^{3-0} = \bar{\mathbf{p}}(\{\dots, \bar{\mathbf{c}}_i, \dots\}, 3; u)$$

$$= \left(\alpha_{i+1}^3 \bar{\mathbf{d}}_i^2 + \beta_{i+1}^3 \bar{\mathbf{d}}_{i+1}^2 \right) / \gamma_{i+1}^3 = \left((u_{i+1} - u) \bar{\mathbf{d}}_i^2 + (u - u_i) \bar{\mathbf{d}}_{i+1}^2 \right) / (u_{i+1} - u_i)$$

$$\bar{\mathbf{d}}_{i+1}^2 = \left(\alpha_{i+1}^2 \bar{\mathbf{d}}_i^1 + \beta_{i+1}^2 \bar{\mathbf{d}}_{i+1}^1 \right) / \gamma_{i+1}^2 = \left((u_{i+2} - u) \bar{\mathbf{d}}_i^1 + (u - u_i) \bar{\mathbf{d}}_{i+1}^1 \right) / (u_{i+2} - u_i)$$

$$\bar{\mathbf{d}}_i^2 = \left(\alpha_i^2 \bar{\mathbf{d}}_{i-1}^1 + \beta_i^2 \bar{\mathbf{d}}_i^1 \right) / \gamma_i^2 = \left((u_{i+1} - u) \bar{\mathbf{d}}_{i-1}^1 + (u - u_{i-1}) \bar{\mathbf{d}}_i^1 \right) / (u_{i+1} - u_{i-1})$$

$$\bar{\mathbf{d}}_{i+1}^1 = \left(\alpha_{i+1}^1 \bar{\mathbf{d}}_i^0 + \beta_{i+1}^1 \bar{\mathbf{d}}_{i+1}^0 \right) / \gamma_{i+1}^1 = \left((u_{i+3} - u) \bar{\mathbf{d}}_i^0 + (u - u_i) \bar{\mathbf{d}}_{i+1}^0 \right) / (u_{i+3} - u_i)$$

$$\bar{\mathbf{d}}_i^1 = \left(\alpha_i^1 \bar{\mathbf{d}}_{i-1}^0 + \beta_i^1 \bar{\mathbf{d}}_i^0 \right) / \gamma_i^1 = \left((u_{i+2} - u) \bar{\mathbf{d}}_{i-1}^0 + (u - u_{i-1}) \bar{\mathbf{d}}_i^0 \right) / (u_{i+2} - u_{i-1})$$

$$\bar{\mathbf{d}}_{i-1}^1 = \left(\alpha_{i-1}^1 \bar{\mathbf{d}}_{i-2}^0 + \beta_{i-1}^1 \bar{\mathbf{d}}_{i-1}^0 \right) / \gamma_{i-1}^1 = \left((u_{i+1} - u) \bar{\mathbf{d}}_{i-2}^0 + (u - u_{i-2}) \bar{\mathbf{d}}_{i-1}^0 \right) / (u_{i+1} - u_{i-2})$$



Uniform B-Spline Polynomials

Third degree uniform B-spline $P(\bar{\mathbf{C}}, \bar{\mathbf{u}}; t) = \sum_j \bar{c}_j N_j^2(\bar{\mathbf{u}}, t)$ with $t_j = j$

$$N_j^3(\bar{\mathbf{u}}, t) = \begin{cases} \frac{1}{6}(t-j)^2 & \text{if } j \leq t < j+1 \\ \frac{1}{6}[-3(t-j-1)^3 + 3(t-j-1)^2 + 3(t-j-1) + 1] & \text{if } j+1 \leq t < j+2 \\ \frac{1}{6}[3(t-j-1)^3 - 6(t-j-1)^2 + 4] & \text{if } j+2 \leq t < j+3 \\ \frac{1}{6}[1-(t-j-1)]^3 & \text{if } j+3 \leq t < j+4 \\ 0 & \text{otherwise} \end{cases}$$

<http://vision.ucsd.edu/~kbranson/research/bsplines/bsplines.pdf>

34 600.445 Fall 2000; Updated: October 7, 14
Copyright © R. H. Taylor

Engineering Research Center for Computer Integrated Surgical Systems and Technology



Some advantages of B-splines

- Efficient
- Numerically stable
- Smooth
- Local

35 600.445 Fall 2000; Updated: October 7, 14
Copyright © R. H. Taylor

Engineering Research Center for Computer Integrated Surgical Systems and Technology



2D Interpolation (tensor form)

Consider the 2D polynomial

$$P(u, v) = \sum_{i=0}^m \sum_{j=0}^n c_{ij} A_i(u) B_j(v)$$

$$= [A_0(u), \dots, A_m(u)] \begin{bmatrix} c_{00} & \cdots & c_{0n} \\ \vdots & \ddots & \vdots \\ c_{m0} & \cdots & c_{mn} \end{bmatrix} \begin{bmatrix} B_0(v) \\ \vdots \\ B_n(v) \end{bmatrix}$$

where $A_i(u)$ and $B_j(v)$ can be arbitrary functions (good choices Bernstein polynomials or B-Spline basis functions. Suppose that we have samples

$$\mathbf{y}_s = \mathbf{y}(u_s, v_s) \text{ for } s = 0, \dots, N_s$$

We want to find an approximating polynomial P.



2D Interpolation: Finding the best fit

Given a set of sample values $\mathbf{y}_s(u_s, v_s)$ corresponding to 2D coordinates (u_s, v_s) , left hand side basis functions $[A_0(u), \dots, A_m(u)]$ and right hand side basis functions $[B_0(v), \dots, B_n(v)]$, the goal is to find the matrix \mathbf{C} of coefficients \mathbf{c}_{ij} .

To do this, solve the least squares problem

$$\begin{bmatrix} \vdots \\ \mathbf{y}_s(u_s, v_s) \\ \vdots \end{bmatrix} \approx \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots \\ A_0(u_s)B_0(v_s) & A_0(u_s)B_1(v_s) & \cdots & A_i(u_s)B_j(v_s) & \cdots & A_m(u_s)B_n(v_s) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \bullet \begin{bmatrix} \mathbf{c}_{00} \\ \mathbf{c}_{01} \\ \vdots \\ \mathbf{c}_{ij} \\ \vdots \\ \mathbf{c}_{mn} \end{bmatrix}$$



2D Interpolation: Sampling on a regular grid

A common special case arises when the (u_s, v_s) form a regular grid. In this case we have $u_s \in \{u_0, \dots, u_{N_u}\}$ and $v_s \in \{v_0, \dots, v_{N_v}\}$. For each value $v_j \in \{v_0, \dots, v_{N_v}\}$ solve the N_u row least squares problem

$$\begin{bmatrix} \vdots \\ \mathbf{y}_s(u_s, v_j) \\ \vdots \end{bmatrix} \approx \begin{bmatrix} \vdots & \dots & \vdots \\ A_0(u_s) & \dots & A_m(u_s) \\ \vdots & \dots & \vdots \end{bmatrix} \cdot \begin{bmatrix} \mathbf{X}_{j0} \\ \vdots \\ \mathbf{X}_{jm} \end{bmatrix}$$

for the unknown m-vector \mathbf{X}_j . Then solve m n-variable least squares problems

$$\begin{bmatrix} \mathbf{X}_{00} & \mathbf{X}_{01} & \dots & \mathbf{X}_{0m} \\ \mathbf{X}_{10} & \mathbf{X}_{11} & \dots & \mathbf{X}_{1m} \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{X}_{N_v0} & \mathbf{X}_{N_v1} & \dots & \mathbf{X}_{N_vm} \end{bmatrix} \approx \begin{bmatrix} B_0(v_0) & B_1(v_0) & \dots & B_n(v_0) \\ B_0(v_1) & B_1(v_1) & \dots & B_n(v_1) \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ B_0(v_{N_v}) & B_1(v_{N_v}) & \dots & B_n(v_{N_v}) \end{bmatrix} \cdot \begin{bmatrix} \mathbf{c}_{00} & \mathbf{c}_{10} & \dots & \mathbf{c}_{m0} \\ \mathbf{c}_{01} & \mathbf{c}_{11} & \dots & \mathbf{c}_{m1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{c}_{0n} & \mathbf{c}_{1n} & \dots & \mathbf{c}_{mn} \end{bmatrix}$$

for the vectors $[\mathbf{c}_{j0}, \dots, \mathbf{c}_{jn}]$. Note that this latter step requires only 1 SVD or similar matrix computation.



2D Interpolation: Sampling on a regular grid

- There are a number of caveats to the “grid” method on the previous slide. (E.g., you need enough data for each of the least squares problems). But where applicable the method can save computation time since it replaces a number of m and n variable least squares problems for one big m x n problem
- Note that there is a similar trick that you can play by grouping all the common u_i elements together.
- Note that the \mathbf{y} 's and the \mathbf{c} 's do not have to be scalar numbers. They can be Vectors, Matrices, or other objects that have appropriate algebraic properties



N-dimensional interpolation

Define

$$F_{i_1 \dots i_N}(\bar{\mathbf{u}}) = A_{i_1}^1(u_1) \dots A_{i_N}^N(u_N)$$

Then solve the least squares problem

$$\begin{bmatrix} F_{00 \dots 0}(\bar{\mathbf{u}}_s) & F_{10 \dots 0}(\bar{\mathbf{u}}_s) & \dots & F_{m_1 \dots m_n}(\bar{\mathbf{u}}_s) \\ \vdots & \vdots & & \vdots \end{bmatrix} \begin{bmatrix} c_{00 \dots 0} \\ c_{10 \dots 0} \\ \vdots \\ c_{m_1 \dots m_n} \end{bmatrix} \cong \begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix} \bar{\mathbf{y}}_s$$



N-dimensional interpolation

- The methods described earlier generalize naturally to N dimensions.

$$P(\bar{\mathbf{u}}) = P(u_1, \dots, u_N) = \sum_{i_1=0}^{m_1} \dots \sum_{i_N=0}^{m_N} c_{i_1 \dots i_N} A_{i_1}^1(u_1) \dots A_{i_N}^N(u_N)$$

where $A_j^K(u)$ can be arbitrary functions

(good choices are Bernstein polynomials or

B-Spline basis functions). Suppose that we have samples

$$\mathbf{y}_s = \mathbf{y}(\bar{\mathbf{u}}_s) \text{ for } s = 0, \dots, N_s$$

We want to find coefficients of $c_{i_1 \dots i_n}$ approximating polynomial P.



Example: 3D Calibration of Distortion

Suppose we want to compute a distortion correction function for a distorted 3D navigational sensor. Let

$\bar{\mathbf{p}}_i$ = known 3D "ground truth"

$\bar{\mathbf{q}}_i$ = Values returned by navigational sensor

Here we will construct a "tensor form" interpolation polynomial using 5th degree Bernstein polynomials

$$F_{ijk}(u_x, u_y, u_z) = B_{5,i}(u_x)B_{5,j}(u_y)B_{5,k}(u_z)$$

We need to do the following:

1. Bernstein polynomials are really designed to work well in the range $0 \leq u \leq 1$, so we need to determine a "bounding box" to scale our $\bar{\mathbf{q}}_i$ values. I.e., we pick upper and lower limits $\bar{\mathbf{q}}^{\min}$ and $\bar{\mathbf{q}}^{\max}$ and compute $\bar{\mathbf{u}}_s = \text{ScaleToBox}(\bar{\mathbf{q}}_s, \bar{\mathbf{q}}^{\min}, \bar{\mathbf{q}}^{\max})$ where

$$\text{ScaleToBox}(x, x^{\min}, x^{\max}) = \frac{x - x^{\min}}{x^{\max} - x^{\min}}$$

2. Now, we set up and solve the least squares problem:

42 600.445 Fall 2000; Updated: October 7, 14
Copyright © R. H. Taylor

Engineering Research Center for Computer Integrated Surgical Systems and Technology



Example: 3D Calibration of Distortion

u, aka q in prev slide
actual EM measurements

solve for this

ground truth
c expected

$$\begin{bmatrix} F_{000}(\bar{\mathbf{u}}_s) & \cdots & F_{555}(\bar{\mathbf{u}}_s) \end{bmatrix} \begin{bmatrix} c_{000}^x & c_{000}^y & c_{000}^z \\ \vdots & \vdots & \vdots \\ c_{555}^x & c_{555}^y & c_{555}^z \end{bmatrix} \equiv \begin{bmatrix} p_s^x & p_s^y & p_s^z \\ \vdots & \vdots & \vdots \end{bmatrix}$$

solve this least squared problem

43 600.445 Fall 2000; Updated: October 7, 14
Copyright © R. H. Taylor

Engineering Research Center for Computer Integrated Surgical Systems and Technology



Example: 3D Calibration of Distortion

The correction function will then look like this:

\mathbf{q} is actual measured points from EM system

returns \mathbf{p} : corrected point

This is a function definition: $\bar{\mathbf{p}} = \text{CorrectDistortion}(\bar{\mathbf{q}})$

Implementation: $\{ \bar{\mathbf{u}} = \text{ScaleToBox}(\bar{\mathbf{q}}, \bar{\mathbf{q}}^{\min}, \bar{\mathbf{q}}^{\max})$

$$\text{return } \sum_{i=0}^5 \sum_{j=0}^5 \sum_{k=0}^5 \bar{\mathbf{c}}_{i,j,k} B_{5,i}(u_x) B_{5,j}(u_y) B_{5,k}(u_z)$$

$\}$

Note, if \mathbf{F} from before is stacked as one giant matrix,
it can all be done in a single matrix multiplication.

The summation is not required in that case.

