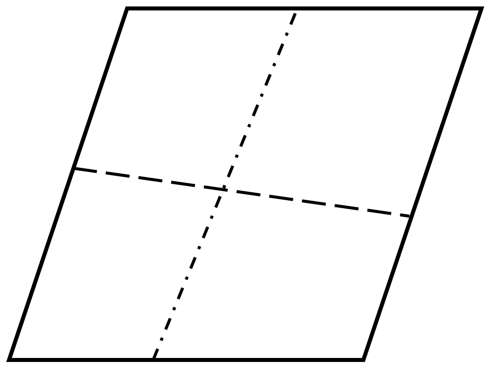
Interpolation and Deformations
A short cookbook



Linear Interpolation

$$\vec{\mathbf{p}}_2 = \begin{bmatrix} 40 & 30 & 20 \end{bmatrix}^T$$
$$\rho_2 = 20$$

$$\vec{\mathbf{p}}_3 = \begin{bmatrix} 20 & 20 & 20 \end{bmatrix}^T$$
$$\rho_3 = \mathbf{10}$$

$$\vec{\mathbf{p}}_1 = \begin{bmatrix} 10 & 15 & 20 \end{bmatrix}^T$$
$$\rho_1 = 5$$

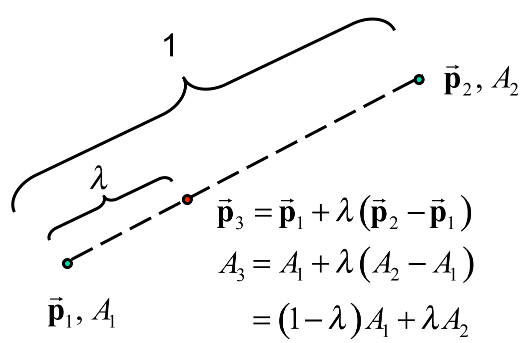
Linear Interpolation

$$\vec{\mathbf{p}}_2 = \begin{bmatrix} 40 & 30 & 20 \end{bmatrix}^T$$
$$\vec{\mathbf{q}}_2 = \vec{\mathbf{b}}$$

$$\vec{\mathbf{p}}_3 = \begin{bmatrix} 20 & 20 & 20 \end{bmatrix}^T$$
$$\vec{\mathbf{q}}_3 = \vec{\mathbf{a}} + \tfrac{1}{3}\left(\vec{\mathbf{b}} - \vec{\mathbf{a}}\right)$$

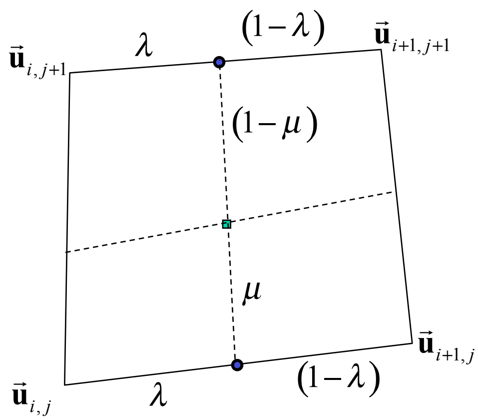
$$\vec{\mathbf{p}}_1 = \begin{bmatrix} 10 & 15 & 20 \end{bmatrix}^T$$
$$\vec{\mathbf{q}}_1 = \vec{\mathbf{a}}$$

Linear Interpolation

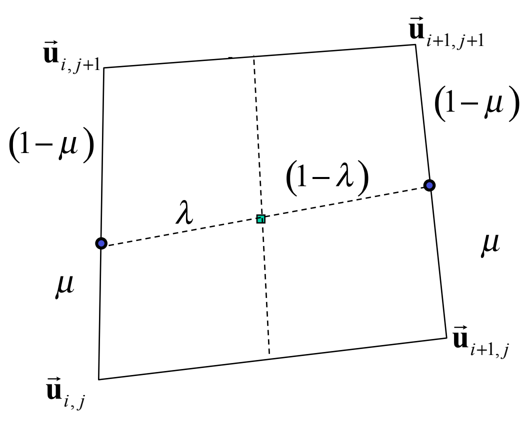


$$\vec{\mathbf{p}}_3 = \vec{\mathbf{p}}_1 + \lambda\left(\vec{\mathbf{p}}_2 - \vec{\mathbf{p}}_1\right)$$
$$A_3 = A_1 + \lambda\left(A_2 - A_1\right)$$
$$= \left(1-\lambda\right)A_1 + \lambda A_2$$
$$\lambda = \frac{\left(\vec{\mathbf{p}}_3 - \vec{\mathbf{p}}_1\right)\bullet\left(\vec{\mathbf{p}}_2 - \vec{\mathbf{p}}_1\right)}{\left(\vec{\mathbf{p}}_2 - \vec{\mathbf{p}}_1\right)\bullet\left(\vec{\mathbf{p}}_2 - \vec{\mathbf{p}}_1\right)}$$

Bilinear Interpolation

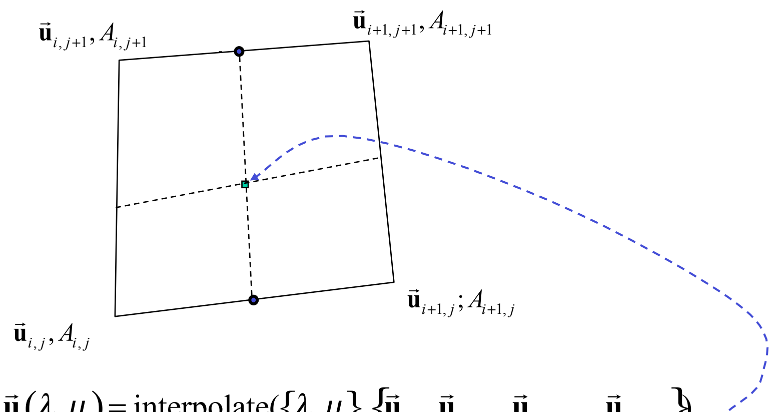


Bilinear Interpolation



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

Bilinear Interpolation



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

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

N-linear Interpolation

Let

$$\overline{\Lambda}_N = \{\lambda_1,\ldots,\lambda_N\}, \text{ with } 0 \le \lambda_k \le 1$$

be a set of interpolation parameters, and let

$$\overline{\mathbf{A}} = \{A_1,\ldots,A_{2^N}\}$$

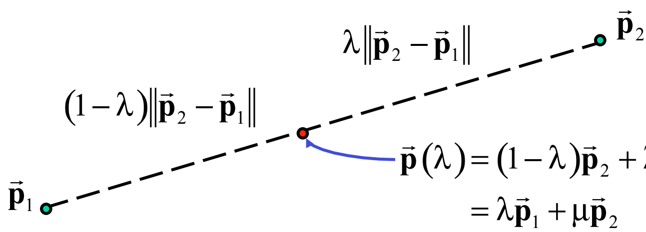
be a set of constants. Then we define:

$$\begin{aligned}
&\text{NlinearInterpolate}(\Lambda_N,\mathbf{A}) = \\
&\qquad (1-\lambda_N)\text{NlinearInterpolate}(\Lambda_{N-1},\{A_1,\ldots,A_{2^{N-1}}\}) \\
&\qquad + \lambda_N \ \text{NlinearInterpolate}(\Lambda_{N-1},\{A_{2^{N-1}+1},\ldots,A_{2^N}\})
\end{aligned}$$

NOTE: Sometimes in this situation we will use notation

$$\begin{aligned}
A(\overline{\Lambda}_N) &= A(\lambda_1,\ldots,\lambda_N) \\
&= \text{NlinearInterpolate}(\overline{\Lambda}_N,\overline{\mathbf{A}})
\end{aligned}$$

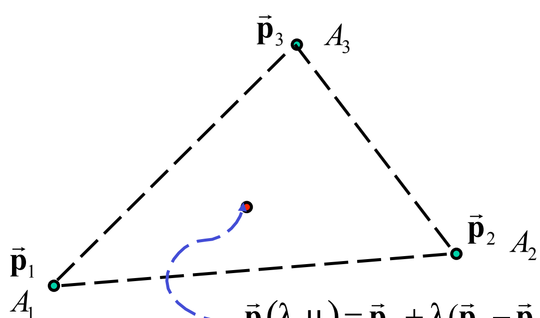
Barycentric Interpolation



$$\begin{aligned}\vec{\mathbf{p}}(\lambda) &= (1-\lambda)\vec{\mathbf{p}}_2+\lambda\vec{\mathbf{p}}_1\\ &= \lambda\vec{\mathbf{p}}_1+\mu\vec{\mathbf{p}}_2\end{aligned}$$

$$\lambda+\mu=1$$

Barycentric Interpolation

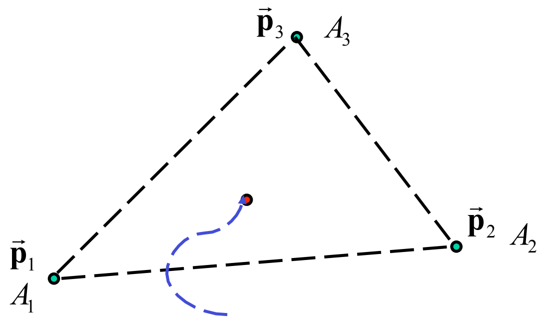


$$\begin{aligned}\vec{\mathbf{p}}(\lambda,\mu) &= \vec{\mathbf{p}}_3+\lambda(\vec{\mathbf{p}}_1-\vec{\mathbf{p}}_3)+\mu(\vec{\mathbf{p}}_2-\vec{\mathbf{p}}_3)\\ &= \lambda\vec{\mathbf{p}}_1+\mu\vec{\mathbf{p}}_2+(1-\lambda-\mu)\vec{\mathbf{p}}_3\end{aligned}$$

$$\vec{\mathbf{p}}(\lambda,\mu,\nu)=\lambda\vec{\mathbf{p}}_1+\mu\vec{\mathbf{p}}_2+\nu\vec{\mathbf{p}}_3 \quad \text{where } \lambda+\mu+\nu=1$$

$$A(\lambda,\mu,\nu)=\lambda A_1+\mu A_2+\nu A_3$$

Barycentric Interpolation



$$\begin{bmatrix} \vec{\mathbf{p}} \\ 1 \end{bmatrix} = \begin{bmatrix} \vec{\mathbf{p}}_1 & \vec{\mathbf{p}}_2 & \vec{\mathbf{p}}_3 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \lambda \\ \mu \\ \nu \end{bmatrix}$$

Barycentric Interpolation

Let

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

be a set of interpolation parameters, and let

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

be a set of constants. Then we define:

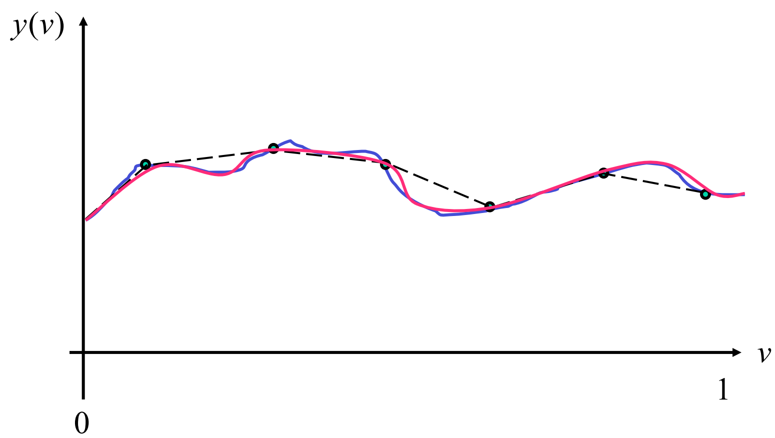
$$\text{BarycentricInterpolate}(\vec{\Lambda}, \vec{\mathbf{A}}) = \vec{\Lambda} \bullet \vec{\mathbf{A}} = \sum_{k=1}^{N} \lambda_k A_k$$

NOTE: Sometimes in this situation we will use notation

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

NOTE: This is a special case of barycentric Bezier polynomial interpolations (here, 1st degree)

Interpolation of functions



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.

1-D Interpolation

Given set of known values $\{y_0(v_0),...,y_m(v_m)\}$,

find an approximating polynomial $y \cong P(c_0,...,c_N;v)$

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

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 plynomials and the b-spline basis functions, which we will discuss in a moment

1-D Interpolation

Given set of known values $\{y_0(v_0),...,y_m(v_m)\}$,

find an approximating polynomial $y \cong P(c_0,...,c_N;v)$

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

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

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

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

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

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

Barycentric Bezier Polynomials

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

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

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

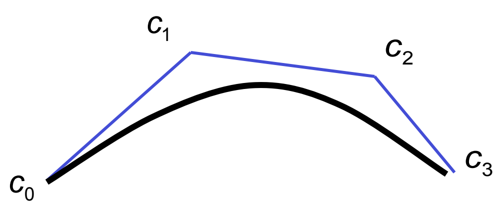
- Excellent numerical stability for c<0<1
- There exist good ways to convert to more conventional power basis

Bezier Curves

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

$$P\left(\overrightarrow{c_0},\ldots,\overrightarrow{c_N};v\right)=\sum_{k=0}^{N}\overrightarrow{c_k}B_{N,k}(v)$$

computed over the range $0 \le v \le 1$ generates a Bezier curve with control vertices $\overrightarrow{c_j}$.



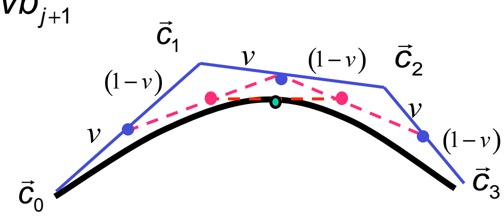
Bezier Curves: de Casteljau Algorithm

Given coefficients $\overrightarrow{c_j}$, Bezier curves can be generated recursively by repeated linear interpolation:

$$P\left(\overrightarrow{c_0},\ldots,\overrightarrow{c_N};v\right)=b_0^N$$

where

$$b_j^0 = \overrightarrow{c_j}$$
$$b_j^k = (1-v)b_j^{k-1} + v b_{j+1}^{k-1}$$



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 $\bar{\mathbf{C}} = \{\vec{c}_0, \cdots, \vec{c}_{L+D-1}\}$

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

D = "degree" of desired B-spline

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

Then

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

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/Classes/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

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

where

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

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

B-Spline Polynomials

For a B-spline polynomial

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

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

deBoor Algorithm

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

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

Step 2: Determine multplicity r such that

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

Step 3: Set $\vec{\mathbf{d}}_j^0 = c_j$ for $i-D+1 \le j \le i+1$

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

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

deBoor Algorithm: Example D=3, r=0

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

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

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

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

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

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

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

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

Uniform B-Spline Polynomials

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

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

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

Some advantages of B-splines

- Efficient
- Numerically stable
- Smooth
- <u>Local</u>

2D Interpolation (tensor form)

Consider the 2D polynomial

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

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

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

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

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 $\left[A_0(u),\cdots,A_m(u)\right]$ and right hand side basis functions $\left[B_0(v),\cdots,B_n(v)\right]$, the goal is to find the matrix $\mathbf{C}$ of coefficients $\mathbf{c}_{ij}$.

To do this, solve the least squares problem

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

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 $u_s \in \{u_0, \cdots, u_{N_u}\}$ and $v_s \in \{v_0, \cdots, v_{N_v}\}$. For each value $v_j \in \{v_0, \cdots, v_{N_v}\}$ solve the N_s row least squares problem

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

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

$$\left[\begin{array}{c|c|c|c} \mathbf{X}_{00} & \mathbf{X}_{01} & \cdots & \mathbf{X}_{0m} \\ \mathbf{X}_{10} & \mathbf{X}_{11} & \cdots & \mathbf{X}_{1m} \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & & \vdots \\ \mathbf{X}_{N_v 0} & \mathbf{X}_{N_v 1} & \cdots & \mathbf{X}_{N_v m} \end{array}\right] \approx \begin{bmatrix} B_0(v_0) & B_1(v_0) & \cdots & B_n(v_0) \\ B_0(v_1) & B_1(v_1) & \cdots & B_n(v_1) \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & & \vdots \\ B_0(v_{N_v}) & B_1(v_{N_v}) & \cdots & B_n(v_{N_v}) \end{bmatrix} \bullet \left[\begin{array}{c|c|c|c} \mathbf{c}_{00} & \mathbf{c}_{10} & \cdots & \mathbf{c}_{m0} \\ \mathbf{c}_{01} & \mathbf{c}_{11} & \cdots & \mathbf{c}_{m1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{c}_{0n} & \mathbf{c}_{1n} & \cdots & \mathbf{c}_{mn} \end{array}\right]$$

for the vectors $\left[\mathbf{c}_{j0}, \cdots, \mathbf{c}_{jn}\right]$. 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 **y**'s and the **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

$$F_{i_1\cdots i_N}(\vec{\mathbf{u}}) = A^1{}_{i_1}(u_1)\cdots A^N{}_{i_N}(u_N)$$

Then solve the least squares problem

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

N-dimensional interpolation

- The methods described earlier generalize naturally to N dimensions.

$$P(\vec{\mathbf{u}}) = P(u_1,\cdots,u_N) = \sum_{i_1=0}^{m_1}\cdots\sum_{i_N=0}^{m_N} c_{i_1\cdots i_n} A^1{}_{i_1}(u_1)\cdots A^N{}_{i_N}(u_N)$$

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

(good choices are Bernstein polynomials or

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

$\mathbf{y}_\mathbf{s} = \mathbf{y}(\vec{\mathbf{u}}_s)$ for $s = 0,\ldots,N_s$

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

Example: 3D Calibration of Distortion

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

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

$\vec{\mathbf{q}}_i$ = 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 $0 \le u \le 1$, so we need to determine a "bounding box" to scale out $\vec{\mathbf{q}}$ values. I.e., we pick upper and lower limits $\vec{\mathbf{q}}^{min}$ and $\vec{\mathbf{q}}^{max}$ and compute $\vec{\mathbf{u}}_s = ScaleToBox(\vec{\mathbf{q}}_s, \vec{\mathbf{q}}^{min}, \vec{\mathbf{q}}^{max})$ where

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

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

Example: 3D Calibration of Distortion

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

Example: 3D Calibration of Distortion

The correction function will then look like this:

$$\vec{\mathbf{p}} = CorrectDistortion(\vec{\mathbf{q}})$$

$$\{ \quad \vec{\mathbf{u}} = ScaleToBox(\vec{\mathbf{q}}, \vec{\mathbf{q}}^{\min}, \vec{\mathbf{q}}^{\max})$$

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

$$\}$$