Interpolation and Deformations
A short cookbook
Linear Interpolation

\[ \vec{p}_1 = [10 \ 15 \ 20]^T \]
\[ \rho_1 = 5 \]
\[ \vec{p}_2 = [40 \ 30 \ 20]^T \]
\[ \rho_2 = 20 \]
\[ \vec{p}_3 = [20 \ 20 \ 20]^T \]
\[ \rho_3 = 10 \]
Linear Interpolation

\[ \mathbf{p}_2 = [40 \ 30 \ 20]^T \]
\[ \mathbf{q}_2 = \mathbf{b} \]

\[ \mathbf{p}_3 = [20 \ 20 \ 20]^T \]
\[ \mathbf{q}_3 = \mathbf{a} + \frac{1}{3}(\mathbf{b} - \mathbf{a}) \]

\[ \mathbf{p}_1 = [10 \ 15 \ 20]^T \]
\[ \mathbf{q}_1 = \mathbf{a} \]
Linear Interpolation

\[ \vec{p}_3 = \vec{p}_1 + \lambda (\vec{p}_2 - \vec{p}_1) \]

\[ A_3 = A_1 + \lambda (A_2 - A_1) = (1 - \lambda) A_1 + \lambda A_2 \]

\[ \lambda = \frac{(\vec{p}_3 - \vec{p}_1) \cdot (\vec{p}_2 - \vec{p}_1)}{(\vec{p}_2 - \vec{p}_1) \cdot (\vec{p}_2 - \vec{p}_1)} \]
Bilinear Interpolation

\[ \vec{u}_{i, j+1} \]
\[ \lambda \]
\[ (1-\lambda) \]
\[ \vec{u}_{i+1, j} \]
\[ \mu \]
\[ (1-\mu) \]
\[ \vec{u}_{i+1, j+1} \]
Bilinear Interpolation

\[ \tilde{\mathbf{u}}(\lambda, \mu) = \lambda \left( \mu \tilde{\mathbf{u}}_{i+1, j} + (1 - \mu) \tilde{\mathbf{u}}_{i+1, j+1} \right) + (1 - \lambda) \left( \mu \tilde{\mathbf{u}}_{i, j+1} + (1 - \mu) \tilde{\mathbf{u}}_{i, j} \right) \]

\[ = \tilde{\mathbf{u}}_{i, j} + \lambda \left( \tilde{\mathbf{u}}_{i+1, j} - \tilde{\mathbf{u}}_{i, j} \right) + \mu \left( \tilde{\mathbf{u}}_{i, j+1} - \tilde{\mathbf{u}}_{i, j} \right) + \lambda \mu \left( \tilde{\mathbf{u}}_{i+1, j+1} - \tilde{\mathbf{u}}_{i, j} \right) \]
Bilinear Interpolation

\[ \tilde{u}(\lambda, \mu) = \text{interpolate}\left(\{\lambda, \mu\}, \{\tilde{u}_{i,j}, \tilde{u}_{i+1,j}, \tilde{u}_{i+1,j+1}, \tilde{u}_{i,j+1}\}\right) \]

\[ A(\lambda, \mu) = \text{interpolate}\left(\{\lambda, \mu\}, \{A_{i,j}, A_{i+1,j}, A_{i+1,j+1}, A_{i,j+1}\}\right) \]
N-linear Interpolation

Let

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

be a set of interpolation parameters, and let

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

be a set of constants. Then we define:

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

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

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

NOTE: Sometimes in this situation we will use notation

$$A(\overline{\Lambda}_N) = A(\lambda_1, \ldots, \lambda_N)$$

$$= \text{NlinearInterpolate}(\overline{\Lambda}_N, \overline{A})$$
Barycentric Interpolation

\[ \vec{p}(\lambda) = (1 - \lambda)\vec{p}_2 + \lambda\vec{p}_1 = \lambda\vec{p}_1 + \mu\vec{p}_2 \]
\[ \lambda + \mu = 1 \]
Barycentric Interpolation

\[ \mathbf{p}(\lambda, \mu) = \mathbf{p}_3 + \lambda(\mathbf{p}_1 - \mathbf{p}_3) + \mu(\mathbf{p}_2 - \mathbf{p}_3) \]

\[ = \lambda \mathbf{p}_1 + \mu \mathbf{p}_2 + (1 - \lambda - \mu) \mathbf{p}_3 \]

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

\[ A(\lambda, \mu, \nu) = \lambda A_1 + \mu A_2 + \nu A_3 \]
Barycentric Interpolation

\[
\begin{bmatrix}
\vec{p} \\
1
\end{bmatrix} =
\begin{bmatrix}
\vec{p}_1 & \vec{p}_2 & \vec{p}_3
\end{bmatrix}
\begin{bmatrix}
\lambda \\
\mu \\
\nu
\end{bmatrix}
\]
Barycentric Interpolation

Let
\[ \tilde{\Lambda} = \{\lambda_1, \ldots, \lambda_N\}, \text{ with } 0 \leq \lambda_k \leq 1 \text{ and } \sum_{k=1}^{N} \lambda_k = 1 \]
be a set of interpolation parameters, and let
\[ \tilde{\mathbf{A}} = \{A_1, \ldots, A_{2^N}\} \]
be a set of constants. Then we define:

\[ \text{BarycentricInterpolate}(\tilde{\Lambda}, \tilde{\mathbf{A}}) = \tilde{\Lambda} \cdot \tilde{\mathbf{A}} = \sum_{k=1}^{N} \lambda_k A_k \]

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

\[ y(v) \]

0 \quad 1

\text{v}
Fitting of interpolation curves

- The discussion below follows (in part)

1-D Interpolation

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

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

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 polynomials and the b-spline basis functions, which we will discuss in a moment.
1-D Interpolation

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

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

To do this, solve:

\[
\begin{bmatrix}
P_{N,0}(v_0) & \cdots & P_{N,N}(v_0) \\
\vdots & \ddots & \vdots \\
P_{N,0}(v_m) & \cdots & P_{N,N}(v_m)
\end{bmatrix}
\begin{bmatrix}
c_0 \\
\vdots \\
c_m
\end{bmatrix}
= \begin{bmatrix}
y_0 \\
\vdots \\
y_m
\end{bmatrix}
\]
Bezr and Bernstein Polynomials

\[
P(\mathbf{c}_0, \ldots, \mathbf{c}_N ; v) = \sum_{k=0}^{N} c_k \left( \begin{array}{c} N \\ k \end{array} \right) (1 - v)^{N-k} v^k
\]

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

where \( B_{N,k}(v) = \left( \begin{array}{c} N \\ k \end{array} \right) (1 - v)^{N-k} v^k \)

- Excellent numerical stability for \( 0 < v < 1 \)
- There exist good ways to convert to more conventional power basis
Barycentric Bezier Polynomials

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

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

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

- Excellent numerical stability for \( c<0<1 \)
- There exist good ways to convert to more conventional power basis
Beziers Curves

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

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

computed over the range $0 \leq v \leq 1$ generates a Bezier curve with control vertices $\bar{c}_j$. 
Beziers Curves: de Casteljau Algorithm

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

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

where

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

$$b_j^k = (1 - \nu)b_j^{k-1} + \nu b_{j+1}^{k-1}$$
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

\[ \text{for } j \leftarrow 0 \text{ step 1 until } j = N - k \text{ do} \]

\[ b_j \leftarrow (1 - v)b_j + vb_{j+1} \]

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 \( \mathbf{C} = \{c_0, \ldots, c_{L+D-1}\} \)

"knot points" \( \mathbf{u} = \{u_0, \ldots, u_{L+2D-2}\} \) with \( u_i \leq u_{i+1} \)

\( D = "\text{degree}" \) of desired B-spline

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

Then

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

where \( N_j^D(u) \) are B-spline basis polynomials (discussed later)
deBoor Algorithm

Given \( u, c, D \) as before, can evaluate \( P(c,u;u) \) recursively as follows:

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

Step 2: Determine multiplicity \( r \) such that

\[
  u_{i-r} = u_{i-r+1} = \ldots = u_i
\]

Step 3: Determine \( d_j^0 = c_j \) for \( i-D+1 \leq j \leq i+1 \)

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

\[
  d_j^k = \frac{u_{j+D-k} - u}{u_{j+D-k} - u_{j-1}} d_{j-1}^{k-1} + \frac{u - u_{j-1}}{u_{j+D-k} - u_{j-1}} d_j^{k-1}
\]
B-spline basis functions

Given \( \overline{C}, \overline{u}, D \) as before

\[
P(\overline{C}, \overline{u}; u) = \sum_{j=0}^{L+D-1} \tilde{c}_j N_j^D(u)
\]

where

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

\[
N_j^k(u) = \frac{u - u_{j-1}}{u_{j+k-1} - u_{j-1}} N_j^{k-1}(u) + \frac{u_{j+k} - u}{u_{j+k} - u_j} N_{j+1}^{k-1}(u) \quad \text{for } k > 0
\]
Some advantages of B-splines

• Efficient
• Numerically stable
• Smooth
• Local
2D Interpolation

Consider the 2D polynomial

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

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

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

\[ y_s = y(u_s, v_s) \text{ for } s = 0, \ldots, N_s \]

We want to find an approximating polynomial P.
2D Interpolation: Finding the best fit

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

To do this, solve the least squares problem

\[
\begin{bmatrix}
y_s(u_s, v_s) \\
\vdots
\end{bmatrix} \approx \begin{bmatrix} A_0(u_s)B_0(v_s) & A_0(u_s)B_1(v_s) & \cdots & A_i(u_s)B_j(v_s) & \cdots & A_m(u_s)B_n(v_s) \end{bmatrix} \cdot \begin{bmatrix} c_{00} \\
c_{01} \\
\vdots \\
c_{ij} \\
\vdots \\
c_{mn} \end{bmatrix}
\]
2D Interpolation: Finding the best fit

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

\[
\begin{bmatrix}
\vdots \\
y_s(u_s, v_j)
\end{bmatrix} \approx \begin{bmatrix}
A_0(u_s) & \ldots & A_m(u_s)
\end{bmatrix} \begin{bmatrix}
x_{j0} \\
\vdots \\
x_{jm}
\end{bmatrix}
\]

for the unknown \(m\)-vector \(x_j\). Then solve \(m\) \(n\)-variable least squares problems

\[
\begin{bmatrix}
x_{00} & x_{01} & \cdots & x_{0m} \\
x_{10} & x_{11} & \cdots & x_{1m} \\
\vdots & \vdots & \ddots & \vdots \\
x_{N_u,0} & x_{N_u,1} & \cdots & x_{N_u,m}
\end{bmatrix} \approx \begin{bmatrix}
B_0(v_0) & B_1(v_0) & \cdots & B_n(v_0) \\
B_0(v_1) & B_1(v_1) & \cdots & B_n(v_1) \\
\vdots & \vdots & \ddots & \vdots \\
B_0(v_{N_v}) & B_1(v_{N_v}) & \cdots & B_n(v_{N_v})
\end{bmatrix} \begin{bmatrix}
c_{00} & c_{10} & \cdots & c_{m0} \\
c_{01} & c_{11} & \cdots & c_{m1} \\
\vdots & \vdots & \ddots & \vdots \\
c_{0n} & c_{1n} & \cdots & c_{mn}
\end{bmatrix}
\]

for the vectors \([c_{j0}, \ldots, c_{jn}]\). Note that this latter step requires only 1 SVD or similar matrix computation.
2D Interpolation: Finding the best fit

- 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.