Notes on Bézier Shapes

Daniel W. Zaide

Scientific Computation Research Center, Rensselaer Polytechnic Institute

September 20, 2015

Table of Contents

Bézier shapes

P^{th} order Bézier shapes can be defined as follows:

$$\begin{aligned}
\text{Edge} \quad \mathbf{B}(u) &= \sum_{i+j=P} \frac{P!}{i!j!} u^i v^j \mathbf{C}_{ij}, \quad v = 1-u \\
\text{Triangle} \quad \mathbf{B}(u,v) &= \sum_{i+j+k=P} \frac{P!}{i!j!k!} u^i v^j w^k \mathbf{C}_{ijk}, \quad w = 1-u-v \\
\text{Tet} \quad \mathbf{B}(u,v,w) &= \sum_{i+j+k+l=P} \frac{P!}{i!j!k!l!} u^i v^j w^k t^l \mathbf{C}_{ijkl}, \quad t = 1-u-v-w
\end{aligned}$$

With all parameters as barycentric coordinates,

$$u, v, w, t \in [0, 1]$$

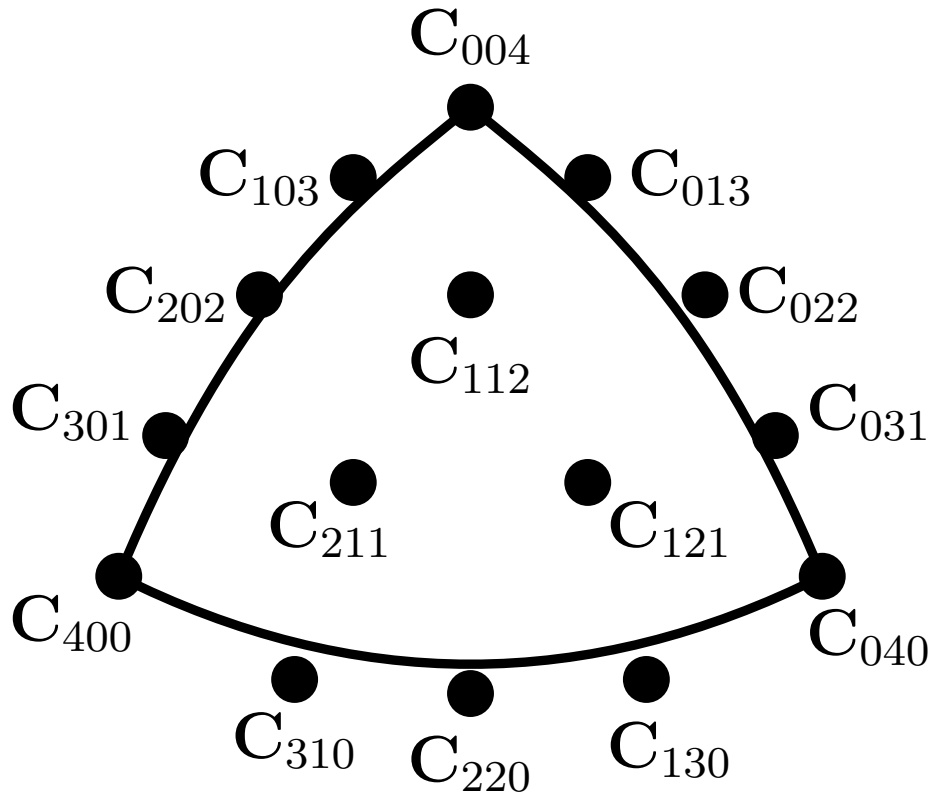
There is often the use of the binomial coefficient, here I will also use that notation, with

$$\frac{P!}{i!j!} = \binom{P}{i}, \qquad \frac{P!}{i!j!k!} = \binom{P}{i,j}, \qquad \frac{P!}{i!j!k!l!} = \binom{P}{i,j,k}$$

$$\frac{P!}{i!j!k!} = \binom{P}{i}\binom{P-i}{j}, \qquad \frac{P!}{i!j!k!l!} = \binom{P}{i}\binom{P-i}{j}\binom{P-i-j}{k}$$

Bézier Triangle

For example, the 4^{th} order Bézier Triangle is drawn here, to illustrate the control points and notation.



Number of Control Points

For P^{th} order Bézier shapes contain the following number of control points

	Total	Internal
Curve	$P+1$	$P-1$
Triangle	$\frac{1}{2}(P+1)(P+2)$	$\frac{1}{2}(P-1)(P-2)$
Tet	$\frac{1}{6}(P+1)(P+2)(P+3)$	$\frac{1}{6}(P-1)(P-2)(P-3)$

For example,

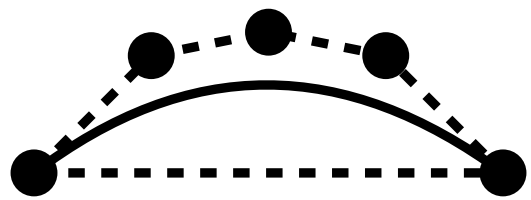
Order	Tri Total	Tri Internal	Tet Total	Tet Internal
4	15	3	35	1
5	21	6	56	4
6	28	10	84	10

Useful Properties

- The convex hull property. The max and min values of a Bézier polynomial evaluated within the domain are bounded by the max and min values of its corresponding control points.
- Variation diminishing. The curve has no more intersections with any plane than polygon of its control points, and converges to the curve.
- Affine invariance. Curves form an affine map, allowing for reparametrization.
- Degree elevation and subdivision. Curves can be increased to higher-orders without changing behavior of the curve.
- Derivatives of Bézier shapes are themselves Bézier shapes.
- Edges of Bézier triangles are Bézier curves and faces of Bézier tets are Bézier triangles.

Convex Hull Property

The Convex Hull property of Bézier shapes guarantees that the shape will lie entirely inside its convex hull. The curve below is inside the hull formed by its control points.

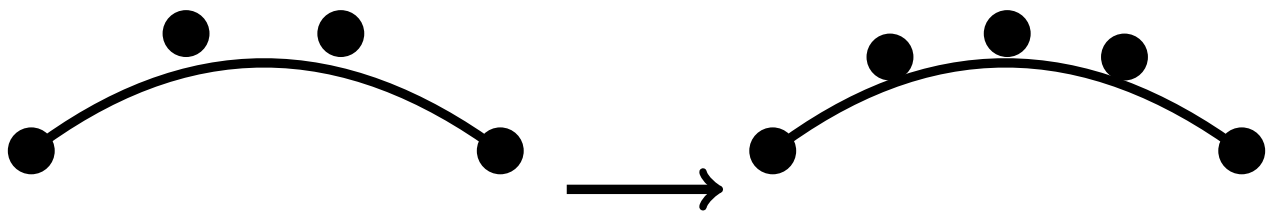


This property guarantees a bound on the shape based solely on its control points, and is important in the validity calculations.

Table of Contents

Elevation

Elevation is the property of Bézier shapes that describes increasing the order of the shape without changing it. Elevating a curve by one order is shown below



We note that as we elevate the curve, the control points converge to the shape itself.

Elevation - Curves

To elevate an order P curve by one order, we have

$$\mathbf{C}_i^{P+1} = \frac{i}{P+1}\mathbf{C}_{i-1}^P + \left(1 - \frac{i}{P+1}\right)\mathbf{C}_i^P, \quad 1 \le i \le P$$

Generalized to elevate by r orders, we get

$$\mathbf{C}_i^{P+r} = \sum_{j=\max(0,i-r)}^{\min(i,P)} \frac{\binom{P}{i}\binom{r}{i-j}}{\binom{P+r}{i}}\mathbf{C}_j^P, \quad 1 \le i < P+r$$

Elevation - Triangles

Elevating triangles is achieved similarly. To elevate by r orders, we have that

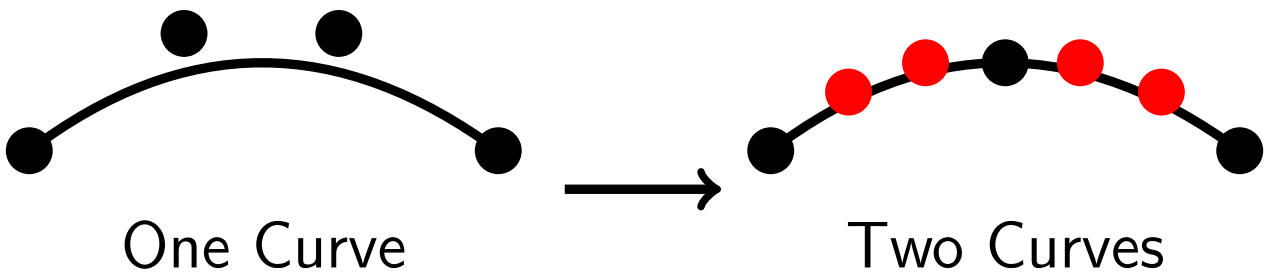
$$\mathbf{C}_{ij}^{P+r} = \sum_{k=\max(0,i-r)}^{\min(i,P)} \sum_{l=\max(0,i-k+j-r)}^{\min(j,P-k)} \frac{\binom{P}{k,l}\binom{r}{i-k,j-l}}{\binom{P+r}{i,j}} \mathbf{C}_{lk}^{P}$$

with $i + j = P + r$. In practice, we could elevate the edges separately from the internal points, however this is not currently done.

Table of Contents

Subdivision

Subdividing a Bézier shape splits it it into multiple shapes that together exactly represent the original shape. Curves become two curves, and triangles become three triangles. For a third order curve subdivided at $u = 1/2$, we get the diagram below



Again, the new control points all converge to the actual curve.

Subdivision - Curves

Subdivision is fairly straight forward, computed using de Casteljau's algorithm. We note that we can write any curve as a sum of two curves of one degree lower,

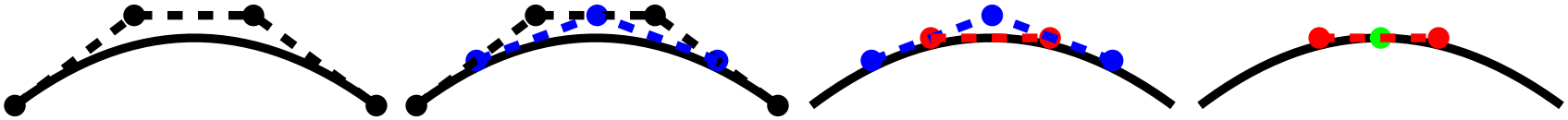
$$\begin{aligned}\mathbf{B}(u) &= \sum_{i=0}^{P} \binom{P}{i} u^i (1-u)^{P-i} \mathbf{C}_i \\ &= (1-u) \sum_{i=0}^{P-1} \binom{P-1}{i} u^i (1-u)^{P-1-i} \mathbf{C}_i + u \sum_{i=0}^{P-1} \binom{P-1}{i} u^i (1-u)^{P-1-i} \mathbf{C}_{i+1}\end{aligned}$$

This is the premise of de Casteljau's algorithm. Given a parameter u and a P^{th} order curve consisting of control points, $\mathbf{C}_i^0$, we first determine the next set of control points, $\mathbf{C}_i^1$ as

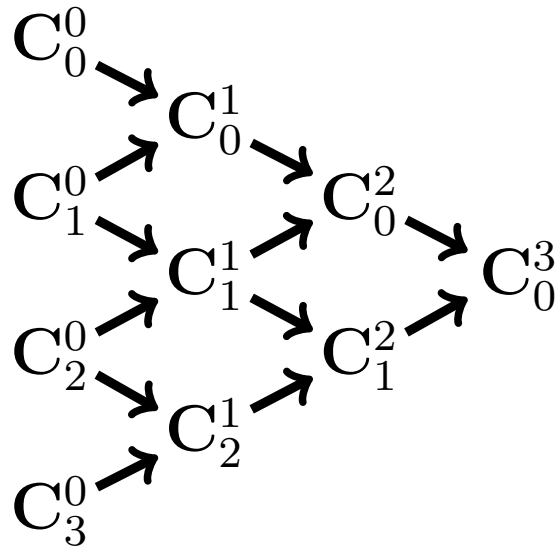
$$\mathbf{C}_i^1 = (1-u)\mathbf{C}_{i-1}^0 + u\mathbf{C}_i^0, \qquad 1 \leq i < P$$

Subdivision - Curves

This continues on, so for a third order curve



Original First Step Second Step Third Step



The two new curves are comprised of points along the upper and lower of the diagram, $\mathbf{C}_0^0, \mathbf{C}_0^1, \mathbf{C}_0^2, \mathbf{C}_0^3$, and $\mathbf{C}_3^0, \mathbf{C}_2^1, \mathbf{C}_1^2, \mathbf{C}_0^3$, where $\mathbf{C}_0^3 = \mathbf{B}(u)$, it lies exactly on the original curve, at u.

Subdivision - Curves

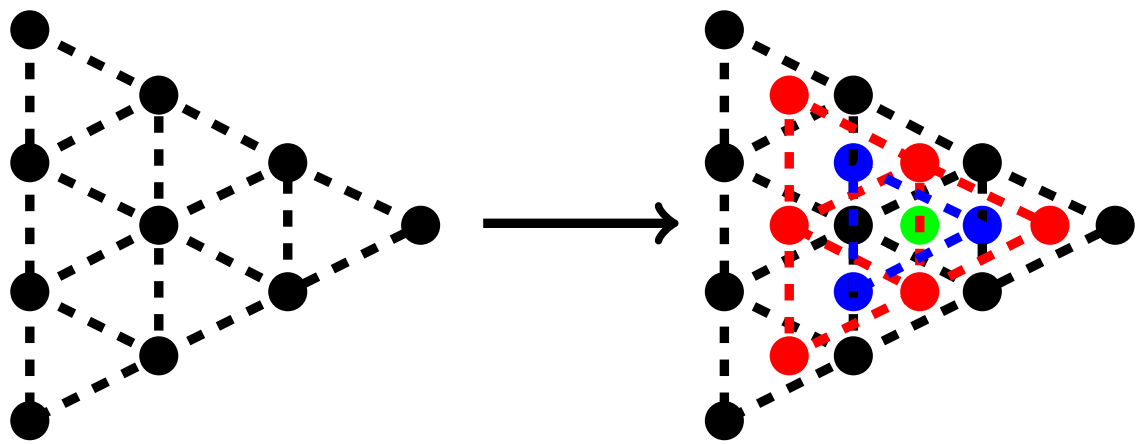
Algorithmically, with the curves as $\mathbf{D}, \mathbf{E}$ we have that

$\mathbf{D}_0 = \mathbf{C}_0^0$
$\mathbf{E}_P = \mathbf{C}_P^0$
for $i \in [0..P)$ **do**
  for $j \in [0..P-i)$ **do**
    $\mathbf{C}_j^{i+1} = (1-u)\mathbf{C}_j^i + u\mathbf{C}_{j+1}^i$
  end for
  $\mathbf{D}_{i+1} = \mathbf{C}_{i+1}^{i+1}$
  $\mathbf{E}_{P-i-1} = \mathbf{C}_{P-i-1}^{i+1}$
end for

and we need to compute all intermediate control points to determine the subdivision. We also note that we could evaluate the curves at any u using this method, and it is more stable, however the direct method is significantly faster.

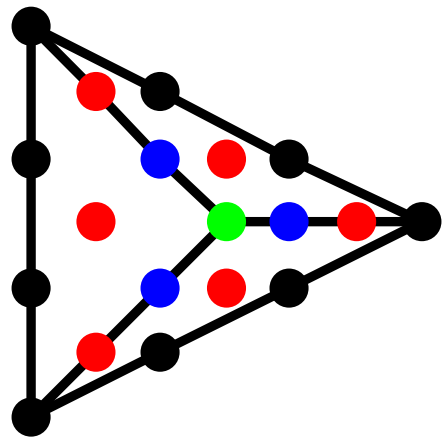
Subdivision - Triangles

Subdivision of triangles turns one triangle into three by splitting at some point (u, v, w). Consider the third order triangle, with 10 points. The steps of the subdivision algorithm are shown on the right by subdividing each subtriangle, shown in red, blue, and green.



Subdivision - Triangles

The three triangles, each with ten control points, are shown below.



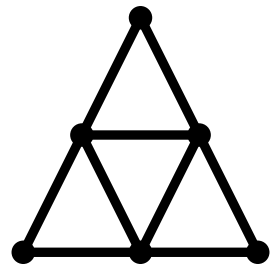
The control points are

$$\mathbf{C}_{i,j,k}^{n+1} = u\mathbf{C}_{i,j,k}^{n} + v\mathbf{C}_{i-1,j+1,k}^{n} + w\mathbf{C}_{i-1,j,k+1}^{n}$$

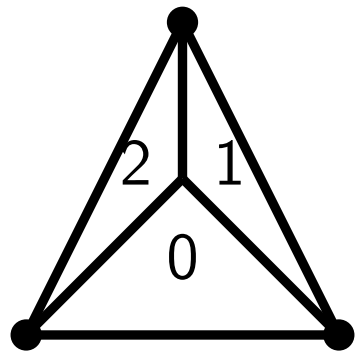
For each step of the algorithm.

Subdivision - Triangle

In practice, subdivision at the triangle center $(1/3, 1/3, 1/3)$ is not ideal, since repeated subdivision does not converge (the longest edge of a triangle does not change). The classic option,

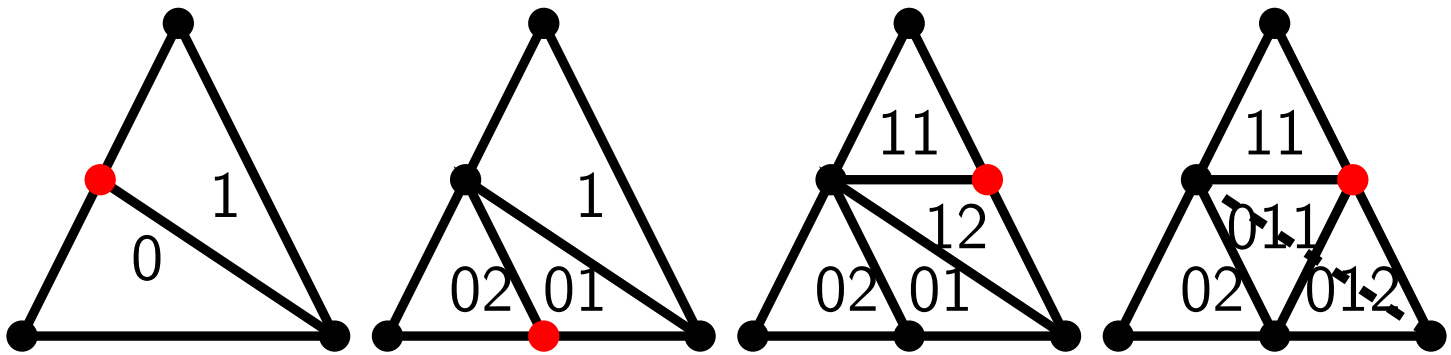


Can be obtained by four calls to the subdivision algorithm. First, denoting the three triangles in the split by



Subdivision - Triangles

1. Split at (0.5,0.5,0).
2. Split 0 at (0,0.5,0.5).
3. Split 1 at (0,0.5,0.5).
4. Split 01 at (-1,1,1)



The issue with this split is that it requires a non convex call (-1,1,1), but in practice, this has been shown to work.

Table of Contents

Derivatives of a Curve

The derivatives of a Bézier curve as also a Bézier curve, writing the formula as a function of just u,

$$\mathbf{B}(u) = \sum_{i=0}^{P} \binom{P}{i} u^i (1-u)^{P-i} \mathbf{C}_i$$

The derivative is an order $P-1$ Bézier, of the form

$$\frac{\mathrm{d}\mathbf{B}(u)}{\mathrm{d}u} = P \sum_{i=0}^{P-1} \binom{P-1}{i} u^i (1-u)^{P-1-i} (\mathbf{C}_{i+1} - \mathbf{C}_i)$$

Implementation wise, it is easier to evaluate it based on just the control points themselves, as

$$\frac{\mathrm{d}\mathbf{B}(u)}{\mathrm{d}u} = -P(1-u)^{P-1}\mathbf{C}_0 + \sum_{i=1}^{P-1} \binom{P}{i} (i - Pu) u^{i-1} (1-u)^{P-i-1} \mathbf{C}_i + Pu^{P-1}\mathbf{C}_P$$

The derivatives of a triangle and tet are written out similarly, as sums of derivatives of the weights of each control point.

Table of Contents

Blended Triangles

The blended triangle is a transfinite interpolation of triangle edges, of the form

$$\mathbf{x}(\xi) = \sum_{i=0}^{3} ||\xi_{e_i}|| w_{e_i}(\xi_{e_i}) \mathbf{x}(\xi_{e_i}) - \sum_{i=0}^{3} \xi_i \mathbf{x}_{v_i}$$

We can use this on all triangles not on a boundary, triangular blending of shape functions can be used. This occurs for all 2D triangles and 3D interior triangles with an edge or more on the boundary.

Blended Triangles

For a given entity on the mesh, we can write the shape as

$$\mathbf{x}(\xi) = \sum_{i=0}^{N} w_i(\xi)\mathbf{x}_i$$

for an entity with N total nodes, where $\xi = (u, v, w)$. For example, for Bézier shapes, $w_i(\xi) = B_{\mathbf{i}}(\xi)$ To implement the blending in our framework, we have

$$\mathbf{x}_{face}(\xi) = \sum_{i=0}^{3} ||\xi_{e_i}|| \sum_{j=0}^{N_{edge}} w_{e_i,j}(\xi_{e_i}) \cdot \mathbf{x}_{e_i,j} - \sum_{i=0}^{3} \xi_i \mathbf{x}_{v_i}$$

Blended Triangles

The edge parameters are

$$\begin{aligned}
\xi_{e_0} &= \frac{\xi_1}{\xi_0 + \xi_1}, \quad ||\xi_{e_0}|| = \xi_0 + \xi_1 \\
\xi_{e_1} &= \frac{\xi_2}{\xi_1 + \xi_2}, \quad ||\xi_{e_1}|| = \xi_1 + \xi_2 \\
\xi_{e_2} &= \frac{\xi_0}{\xi_2 + \xi_0}, \quad ||\xi_{e_2}|| = \xi_2 + \xi_0
\end{aligned}$$

Based on the canonical ordering of edges and faces. When any of $||\xi_{e_i}|| = 0$, $\frac{\xi_1}{\xi_0+\xi_1} = \frac{1}{2}$, the midpoint of the edge. This is irrelevant for the actual shape, but important in computing derivatives.

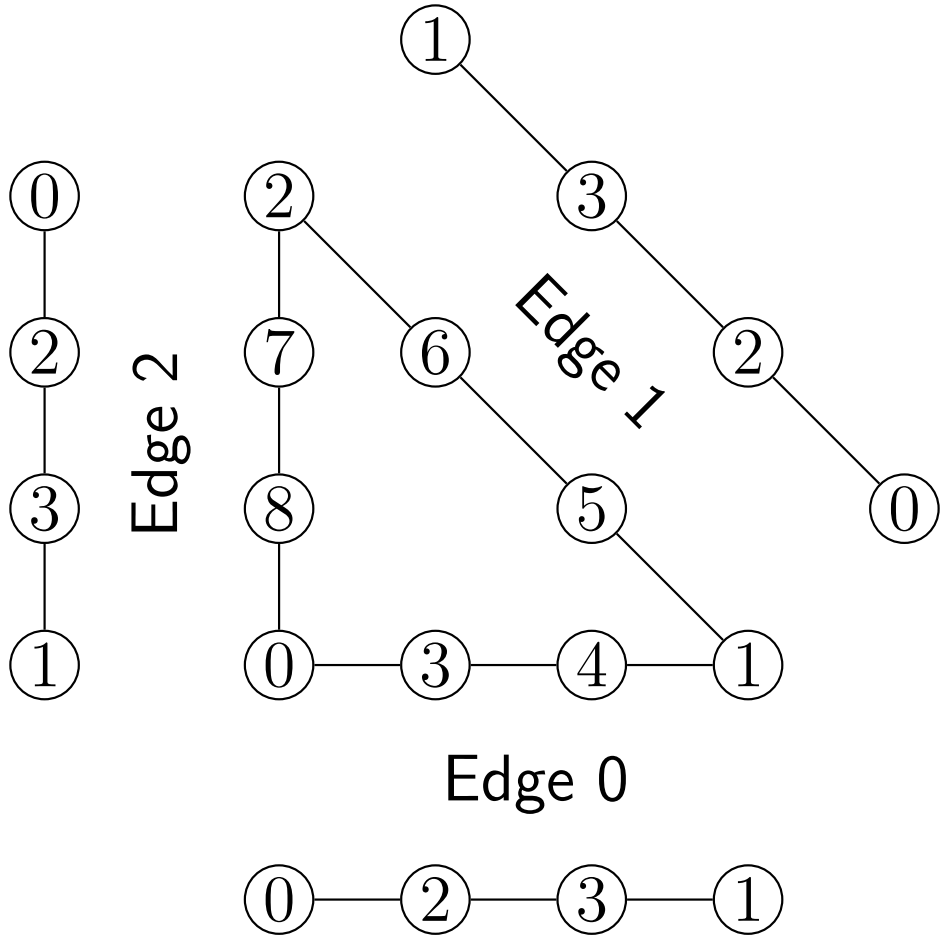
Blended Triangles

The goal is then to find face weights such that

$$\mathbf{x}_{face}(\xi) = \sum_{i=0}^{N_{face}} w_i(\xi) \cdot \mathbf{x}_i = \sum_{i=0}^{3} ||\xi_{e_i}|| \sum_{j=0}^{N_{edge}} w_{e_i,j}(\xi_{e_i}) \cdot \mathbf{x}_{e_i,j} - \sum_{i=0}^{3} \xi_i \mathbf{x}_{v_i}$$

This is a bookkeeping exercise, since every $x_{e_i,j}$ corresponds to a point on the face, in x_i.

Blended Triangles



Blended Triangles

The face has 9 nodes associated with it, ordered as shown. The weights are then

$$
\begin{aligned}
w_0 &= ||\xi_{e_0}||w_{e_0,0}(\xi_{e_0}) + ||\xi_{e_2}||w_{e_2,1}(\xi_{e_2}) - \xi_0 \\
w_1 &= ||\xi_{e_0}||w_{e_0,1}(\xi_{e_0}) + ||\xi_{e_1}||w_{e_1,0}(\xi_{e_1}) - \xi_1 \\
w_2 &= ||\xi_{e_2}||w_{e_2,0}(\xi_{e_2}) + ||\xi_{e_1}||w_{e_1,1}(\xi_{e_1}) - \xi_2 \\
w_3 &= ||\xi_{e_0}||w_{e_0,2}(\xi_{e_0}) \\
w_4 &= ||\xi_{e_0}||w_{e_0,3}(\xi_{e_0}) \\
w_5 &= ||\xi_{e_1}||w_{e_1,2}(\xi_{e_1}) \\
w_6 &= ||\xi_{e_1}||w_{e_1,3}(\xi_{e_1}) \\
w_7 &= ||\xi_{e_2}||w_{e_2,2}(\xi_{e_2}) \\
w_8 &= ||\xi_{e_2}||w_{e_2,3}(\xi_{e_2})
\end{aligned}
$$

Where, for example, $w_{e_0,0}(\xi_{e_0}) = B_{21}(\xi_{e_0})$.

Blended Triangles

The local gradients are then

$$\frac{\partial \mathbf{x}(\xi)}{\partial \xi_k} = \sum_{i=0}^{N} \frac{\partial w_i(\xi)}{\partial \xi_k} \cdot \mathbf{x}_i$$

$$\frac{\partial \mathbf{x}(\xi)}{\partial \xi_k} = \frac{\partial}{\partial \xi_k} \left[\sum_{i=0}^{3} ||\xi_{e_i}|| \sum_{j=0}^{N_{edge}} w_{e_i,j}(\xi_{e_i}) \cdot \mathbf{x}_{e_i,j} - \sum_{i=0}^{3} \xi_i \mathbf{x}_{v_i} \right]$$

$$\sum_{i=0}^{3} \frac{\partial}{\partial \xi_k} \left[||\xi_{e_i}|| \sum_{j=0}^{N_{edge}} w_{e_i,j}(\xi_{e_i}) \right] \cdot \mathbf{x}_{e_i,j} - \sum_{i=0}^{3} \frac{\partial \xi_i}{\partial \xi_k} \mathbf{x}_{v_i}$$

Blended Triangles

For the vertex terms, we have

$$\frac{\partial \xi_i}{\partial \xi_k} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & -1 & 0 \end{bmatrix}$$

and for the edge terms, the chain rule leads to

$$\begin{aligned} \frac{\partial}{\partial \xi_k}\left[||\xi_{e_i}|| \sum_{j=0}^{N_{edge}} w_{e_i,j}(\xi_{e_i}) \right] &= \frac{\partial ||\xi_{e_i}||}{\partial \xi_k} \sum_{j=0}^{N_{edge}} w_{e_i,j}(\xi_{e_i}) \\ &+ ||\xi_{e_i}|| \sum_{j=0}^{N_{edge}} \frac{\partial w_{e_i,j}(\xi_{e_i})}{\partial \xi_k} \end{aligned}$$

where the first term is straight forward, and the second term,

$$\begin{aligned} ||\xi_{e_i}|| \frac{\partial w_{e_i,j}(\xi_{e_i})}{\partial \xi_k} &= ||\xi_{e_i}|| \frac{\partial w_{e_i,j}(\xi_{e_i})}{\partial \xi_{e_i}} \frac{\partial \xi_{e_i}}{\partial \xi_k} \\ &= ||\xi_{e_i}|| \frac{\partial w_{e_i,j}(\xi_{e_i})}{\xi_{e_i}} \left[\frac{\partial \xi'_{e_i}}{\partial \xi_k} ||\xi_{e_i}|| - \xi'_{e_i} \frac{\partial ||\xi_{e_i}||}{\partial \xi_k} \right] \end{aligned}$$

Blended Triangles

Continuity

The linear blending, with $k = 1$ is not continuous across edges. To increase the order of continuity, Dey suggests a general form

$$\mathbf{x}(\xi) = \sum_{i=0}^{3} ||\xi_{e_i}||^k w_{e_i}(\xi_{e_i})\mathbf{x}(\xi_{e_i}) - \sum_{i=0}^{3} \xi_i^k \mathbf{x}_{v_i}$$

However, this is only useful up to $k = 2$, where the blending no longer preserves a linear element.

Blended Triangles

Consider a linear triangle, with vertices $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2$ and barycentric coordinates u, v, w. We can write the blending as

$$\begin{aligned}
\mathbf{x} \;=\; & (u+v)^k \left(\left(1 - \frac{v}{u+v} \right) \mathbf{v}_0 + \frac{v}{u+v} \mathbf{v}_1 \right) + \\
& (v+w)^k \left(\left(1 - \frac{w}{v+w} \right) \mathbf{v}_1 + \frac{w}{v+w} \mathbf{v}_2 \right) + \\
& (w+u)^k \left(\left(1 - \frac{u}{w+u} \right) \mathbf{v}_2 + \frac{u}{w+u} \mathbf{v}_0 \right) + \\
& -u^k \mathbf{v}_0 - v^k \mathbf{v}_1 - w^k \mathbf{v}_2
\end{aligned}$$

Blended Triangles

We can rewrite this as

$$\begin{aligned}\mathbf{x} &= (u+v)^{k-1}(u\mathbf{v}_0 + v\mathbf{v}_1) + \\ &\quad (v+w)^{k-1}(v\mathbf{v}_1 + w\mathbf{v}_2) + \\ &\quad (w+u)^{k-1}(w\mathbf{v}_2 + u\mathbf{v}_0) + \\ &\quad -u^k\mathbf{v}_0 - v^k\mathbf{v}_1 - w^k\mathbf{v}_2\end{aligned}$$

for $k = 1$, we reduce to the exact linear mapping.

$$\mathbf{x} = u\mathbf{v}_0 + v\mathbf{v}_1 + w\mathbf{v}_2$$

For $k = 2$, we again reduce to the exact linear mapping.

$$\mathbf{x} = (u\mathbf{v}_0 + v\mathbf{v}_1 + w\mathbf{v}_2)(u+v+w) = u\mathbf{v}_0 + v\mathbf{v}_1 + w\mathbf{v}_2$$

Blended Triangles

For $k = 3$, we get

$$\mathbf{x} = (u^3 + 2u^2v + 2u^2w + uv^2 + uw^2)\mathbf{v}_0$$

Which is a valid mapping, but does not preserve the linear behavior we would hope for. Of interest is if there exists a function $g(x)$ such that the linear behavior is preserved for all k.

$$\begin{aligned}
\mathbf{x} \;=\; & g(u+v)^k \left(\left(1 - \frac{v}{u+v} \right) \mathbf{v}_0 + \frac{v}{u+v} \mathbf{v}_1 \right) + \\
& g(v+w)^k \left(\left(1 - \frac{w}{v+w} \right) \mathbf{v}_1 + \frac{w}{v+w} \mathbf{v}_2 \right) + \\
& g(w+u)^k \left(\left(1 - \frac{u}{w+u} \right) \mathbf{v}_2 + \frac{u}{w+u} \mathbf{v}_0 \right) + \\
& -u^k \mathbf{v}_0 - v^k \mathbf{v}_1 - w^k \mathbf{v}_2
\end{aligned}$$

Blended Triangles

The goal would be to find a $g(x)$ such that

$$\mathbf{x} = (u\mathbf{v}_0 + v\mathbf{v}_1 + w\mathbf{v}_2)(u+v+w)^{k-1}$$

which reduces to the linear mapping. If we consider the coefficient of the first vertex $\mathbf{v}_0$, we get

$$\left(g(u+v)^k \left(1 - \frac{v}{u+v} \right) + g(w+u)^k \frac{u}{w+u} - u^k \right)$$

Observing that $(u+v+w)^{k-1} = (vw^{k-2} + \ldots + v^{k-2}w) + \ldots$, we note that there is no way for the coefficient of $\mathbf{v}_0$ to contain a vw term, and therefore this formulation makes it impossible to preserve the linear triangle.

Blended Tetrahedra

For blended tetrahedra, we have

$$\mathbf{x}(\xi) = \sum_{i=0}^{4} ||\xi_{f_i}|| w_{f_i}(\xi_{f_i})\mathbf{x}(\xi_{f_i}) - \sum_{i=0}^{6} ||\xi_{e_i}|| w_{e_i}(\xi_{e_i})\mathbf{x}(\xi_{e_i}) + \sum_{i=0}^{4} \xi_i \mathbf{x}_{v_i}$$

The math is identical to triangles, though more consideration to ordering is needed, specifically the face contributions from their edges. Similar to triangles, when any of $||\xi_{e_i}|| = 0$, $\frac{\xi_1}{\xi_0+\xi_1} = \frac{1}{2}$ for edges and now when $||\xi_{e_f}|| = 0$, $\frac{\xi_0}{\xi_0+\xi_1+\xi_2} = \frac{1}{3}$, again, only relevant when taking derivatives.

Table of Contents

Interpolating Bézier Triangles

To fit a Bézier triangle to a known geometry, interpolating points, $\mathbf{P}_{ijk} = \mathbf{P}_\mathbf{i} = \mathbf{B}(\xi) = \mathbf{B}(u, v, w)$, are first needed at $\xi_0, \ldots, \xi_N$, for N total interpolation points before the control points can be calculated. To simplify notation, $\mathbf{i} = ijk$. We can write the triangle as

$$\mathbf{B}(\xi_j) = \sum_{|\mathbf{i}|=P} B_\mathbf{i}(\xi_j)\mathbf{C}_\mathbf{i} = \mathbf{P}_\mathbf{i}$$

at each interpolating point, resulting in a linear system of equations for $\mathbf{C}_i$,

$$\mathbf{P} = \mathbf{B}(\xi_j)\mathbf{C}$$

The control points are then $\mathbf{C} = \mathbf{B}^{-1}(\xi_j)\mathbf{P}$.

For each set of ξ_j, we can precompute and store $\mathbf{B}^{-1}(\xi_j)$. Due to the structure of $\mathbf{B}$ and since edges of Bézier triangles are Bézier curves, we only need to store the rows of $\mathbf{B}^{-1}(\xi_j)$ for the internal points of the triangle, a $\frac{(P-1)(P-2)}{2} \times \frac{(P+1)(P+2)}{2}$ matrix. For edges, a $(P-1) \times (P+1)$ matrix is stored.

To convert interpolation points to control points, the internal triangle control points are computed and then the edge control points. This avoids storing both sets of points.

Several choices of ξ_j exist. Using equally spaced ξ_j's works fairly well, but closer to optimal points are in a paper by Chen and Babuska, 1995.

Computing Internal Control Points of Full Bézier Shapes

For a Bézier triangle or tetrahedra that has a curved boundary (edges or faces), the internal control points should be placed at a suitable location in parametric space. Without a parametrized geometry to interpolate, shape blending is used to determine the internal interpolation points.

We choose a set of local parametric locations, ξ_j, one for each internal point. We can determine this interpolation point using our blended shapes.

Computing Internal Control Points

Write the blended shape as

$$\mathbf{P}(\xi) = \sum_{\mathbf{i} \in shape\ boundaries} Bl_{\mathbf{i}}(\xi)\mathbf{C_i}$$

where the control points on the shape boundaries have been determined from interpolating locations along the bounding entities. The equation to solve is then

$$\mathbf{P}(\xi) = \sum_{\mathbf{i} \in shape\ boundaries} Bl_{\mathbf{i}}(\xi)\mathbf{C_i} = \sum_{|\mathbf{i}|=P} B_{\mathbf{i}}(\xi)\mathbf{C_i}$$

This is an equation for the internal points
$\mathbf{C}_{ijk(l)}, i > 0, j > 0, k > 0, (l > 0), i + j + k(+l) = P.$

Computing Internal Control Points

$$\sum_{\mathbf{i} \in shape\ boundaries} (Bl_{\mathbf{i}}(\xi) - B_{\mathbf{i}}(\xi))\mathbf{C_i} = \sum_{\mathbf{i} \in internal\ points} B_{\mathbf{i}}(\xi)\mathbf{C_i}$$

Which is a system of equations the size of the number of internal points. We can now solve for those locations. In matrix form, setting $Bl_{\mathbf{i}} = 0$ for the internal control points, we can write the equation for all the control points as

$$\mathbf{C} = \mathbf{B}^{-1}\mathbf{Bl}, \qquad \mathbf{B}^{-1}\mathbf{Bl} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{A} & \mathbf{0} \end{bmatrix}$$

and $\mathbf{A}$ is of size (number of internal points)$\times$(number of boundary points). We can precompute $\mathbf{A}$ for each set of (ξ), and store it, applying it to the control points of each internal entity. This determines the control points for a Bézier shape on the interior. (This is similar to P.L. George's approach for serendipity shapes).

Computing Internal Control Points

We should note that for cubic triangles, with one internal control point, the location of that point can be obtained by choosing $\mathbf{C}_{111}$ such that the interpolation exactly matches a quadratic, such that

$$\mathbf{C}_{111} = \frac{1}{4}(\mathbf{C}_{210} + \mathbf{C}_{120} + \mathbf{C}_{012} + \mathbf{C}_{021} + \mathbf{C}_{201} + \mathbf{C}_{102}) - \frac{1}{6}(\mathbf{C}_{300} + \mathbf{C}_{030} + \mathbf{C}_{003})$$

Which we obtain by assuming some $\alpha_{edge}, \alpha_{vertex}, \alpha_{111}$, elevating the quadratic triangle to cubic, and solving

$$\alpha_{vertex} + \frac{2}{3}\alpha_{edge} = 0, \qquad \frac{4}{3}\alpha_{edge} + \frac{1}{3}\alpha_{111} = 0$$

This is not the only method for doing this, but choosing $\alpha_{111} = -1$ obtains the above coefficients.

Computing Internal Control Points

Using this approach on the fourth order triangle, to exactly represent a quadratic results in this expression, for the three internal points

$$\alpha_{vertex} + \alpha_{edge,outer} + \frac{1}{3}\alpha_{edge,mid} + \frac{1}{6}\alpha_{internal} = 0$$

$$\alpha_{edge,outer} + \frac{1}{2}\alpha_{edge,mid} + \frac{5}{6}\alpha_{internal} = 0$$

which has at least one free choice. Elevating from a cubic provides no useful solution.

Computing Internal Control Points

The math comes from the elevation from the quadratic, for third order, the corners remain the same and the edges are

$$\mathbf{C}^3_{edge} = \frac{1}{3}\mathbf{C}^2_{vertex} + \mathbf{C}^2_{midpoint}$$

For the fourth order case, we have an outer edge point and an edge midpoint, with

$$\mathbf{C}^4_{edge,outer} = \frac{1}{2}\mathbf{C}^2_{vertex} + \frac{1}{2}\mathbf{C}^2_{midpoint}$$

$$\mathbf{C}^4_{edge,midpoint} = \frac{1}{6}\mathbf{C}^2_{vertex} + \frac{1}{2}\mathbf{C}^2_{midpoint} + \frac{1}{6}\mathbf{C}^2_{vertex}$$

$$\mathbf{C}^4_{internal} = \frac{1}{3}\mathbf{C}^2_{midpoint} + \frac{1}{2}\mathbf{C}^2_{midpoint} + \frac{1}{6}\mathbf{C}^2_{midpoint} + \frac{1}{6}\mathbf{C}^2_{vertex}$$

Table of Contents

Jacobian Determinants - Triangle

The Jacobian Determinant is itself a Bézier shape. Consider a triangle in $x - y$, we have

$$\mathbf{J} = \frac{\partial \mathbf{B}(u,v)}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial B_x}{\partial x} & \frac{\partial B_x}{\partial y} \\ \frac{\partial B_y}{\partial x} & \frac{\partial B_y}{\partial y} \end{bmatrix}$$

As the derivative is one order lower, we get

$$\mathbf{J} = \frac{\partial \mathbf{B}(u,v)}{\partial \mathbf{x}} = \begin{bmatrix} \mathcal{O}(P-1) & \mathcal{O}(P-1) \\ \mathcal{O}(P-1) & \mathcal{O}(P-1) \end{bmatrix}$$

And thus the determinant for a tet, denoted $\det(\mathbf{J}) = \mathcal{J}(u,v)$, is $\mathcal{J} = \mathcal{O}(2(P-1))$. If the $\mathcal{J} > 0$ everywhere, the element is valid, otherwise it is invalid.

Jacobian Determinants - Triangle

To evaluate $\mathcal{J}(u, v)$, one option is to compute it directly. Writing the triangle as

$$\mathbf{B}(u, v) = \sum_{i+j+k=P} \frac{P!}{i!j!k!} u^i v^j (1-u-v)^k \mathbf{C}_{ijk}$$

and the derivative vector of its coefficients at each control point, with $\mathbf{u} = (u, v)$

$$\frac{\partial B(\mathbf{u})_{ijk}}{\partial \mathbf{u}} = \frac{P!}{i!j!k!} \frac{\partial}{\partial \mathbf{u}} \left(u^i v^j (1-u-v)^k \right)$$

The Jacobian matrix is

$$\mathbf{J}(\mathbf{u}) = \sum_{i+j+k=P} \mathbf{C}_{ijk} \otimes \frac{\partial B(\mathbf{u})_{ijk}}{\partial \mathbf{u}}$$

and the determinant is straightforward.

Jacobian Determinants - Triangle

The other way to compute it is by first calculating the Bézier form of $\mathcal{J}$, and then evaluating it.

$$\mathcal{J}(u,v) = \sum_{I+J+K+L=2(P-1)} \binom{2(P-1)}{I,J,K} u^I v^J (1-u-v)^K N_{IJK}$$

where the control points, N_{IJK}, (scalars) are sums of determinants of the control points

$$N_{IJK} = P^2 \sum_{|i|=I,|j|=J,|k|=K} \frac{\binom{P-1}{i_1,j_1}\binom{P-1}{i_2,j_2}}{\binom{2(P-1)}{I,J,K}}$$
$$\times |\mathbf{C}_{i_1+1,j_1,k_1} - \mathbf{C}_{i_1,j_1+1,k_1}, \mathbf{C}_{i_2+1,j_2,k_2} - \mathbf{C}_{i_2,j_2,k_2+1}|$$

$$|i| = i_1 + i_2, |j| = j_1 + j_2, |k| = k_1 + k_2, \qquad i_1 + j_1 + k_1 = i_2 + j_2 + k_2 = P - 1$$

Jacobian Determinants - Triangle

Since $\mathcal{J}$ is order $2(P-1)$, the number of control points N_{IJK} is much higher than the number of control points of the original shape. In this table, "Terms" is the number of determinants calculated to compute the control points.

Bezier Triangle		Jacobian Determinant		
Order	Num Ctrl Pts	Order	Num Ctrl Pts	Terms
1	3	0	1	1
2	6	2	6	9
3	10	4	15	36
4	15	6	28	100
5	21	8	45	225
6	28	10	66	441
7	36	12	91	784

Jacobian Determinants - Triangle

For a second order triangle, the Jacobian determinant is also second order, with control points

$$
\begin{aligned}
N_{200} &= 4\,|\mathbf{C}_{110}-\mathbf{C}_{200},\mathbf{C}_{101}-\mathbf{C}_{200}| \\
N_{020} &= 4\,|\mathbf{C}_{020}-\mathbf{C}_{110},\mathbf{C}_{011}-\mathbf{C}_{110}| \\
N_{002} &= 4\,|\mathbf{C}_{011}-\mathbf{C}_{101},\mathbf{C}_{002}-\mathbf{C}_{101}| \\
N_{110} &= 2\,|\mathbf{C}_{110}-\mathbf{C}_{200},\mathbf{C}_{011}-\mathbf{C}_{110}| + \\
&\quad\ 2\,|\mathbf{C}_{020}-\mathbf{C}_{110},\mathbf{C}_{101}-\mathbf{C}_{200}| \\
N_{011} &= 2\,|\mathbf{C}_{011}-\mathbf{C}_{101},\mathbf{C}_{011}-\mathbf{C}_{110}| + \\
&\quad\ 2\,|\mathbf{C}_{020}-\mathbf{C}_{110},\mathbf{C}_{002}-\mathbf{C}_{101}| \\
N_{101} &= 2\,|\mathbf{C}_{011}-\mathbf{C}_{101},\mathbf{C}_{101}-\mathbf{C}_{200}| + \\
&\quad\ 2\,|\mathbf{C}_{110}-\mathbf{C}_{200},\mathbf{C}_{002}-\mathbf{C}_{101}|
\end{aligned}
$$

Jacobian Determinants - Triangle

For a third order triangle, the Jacobian determinant is fourth order, with 15 control points, the first 12, corresponding to the vertices and edges of the determinant are

$$
\begin{aligned}
N_{400} &= 9\,|\mathbf{C}_{210}-\mathbf{C}_{300},\mathbf{C}_{201}-\mathbf{C}_{300}| \\
N_{040} &= 9\,|\mathbf{C}_{030}-\mathbf{C}_{120},\mathbf{C}_{021}-\mathbf{C}_{120}| \\
N_{004} &= 9\,|\mathbf{C}_{012}-\mathbf{C}_{102},\mathbf{C}_{003}-\mathbf{C}_{102}| \\
N_{310} &= 4\,|\mathbf{C}_{210}-\mathbf{C}_{300},\mathbf{C}_{111}-\mathbf{C}_{210}| + 4\,|\mathbf{C}_{120}-\mathbf{C}_{210},\mathbf{C}_{201}-\mathbf{C}_{300}| \\
N_{220} &= |\mathbf{C}_{210}-\mathbf{C}_{300},\mathbf{C}_{021}-\mathbf{C}_{120}| + 6\,|\mathbf{C}_{120}-\mathbf{C}_{210},\mathbf{C}_{111}-\mathbf{C}_{210}| + \\
&\quad |\mathbf{C}_{030}-\mathbf{C}_{120},\mathbf{C}_{201}-\mathbf{C}_{300}| \\
N_{130} &= 4\,|\mathbf{C}_{120}-\mathbf{C}_{210},\mathbf{C}_{021}-\mathbf{C}_{120}| + 4\,|\mathbf{C}_{030}-\mathbf{C}_{120},\mathbf{C}_{111}-\mathbf{C}_{210}| \\
N_{031} &= 4\,|\mathbf{C}_{021}-\mathbf{C}_{111},\mathbf{C}_{021}-\mathbf{C}_{120}| + 4\,|\mathbf{C}_{030}-\mathbf{C}_{120},\mathbf{C}_{012}-\mathbf{C}_{111}| \\
N_{022} &= |\mathbf{C}_{012}-\mathbf{C}_{102},\mathbf{C}_{021}-\mathbf{C}_{120}| + 6\,|\mathbf{C}_{021}-\mathbf{C}_{111},\mathbf{C}_{012}-\mathbf{C}_{111}| + \\
&\quad |\mathbf{C}_{030}-\mathbf{C}_{120},\mathbf{C}_{003}-\mathbf{C}_{102}| \\
N_{013} &= 4\,|\mathbf{C}_{012}-\mathbf{C}_{102},\mathbf{C}_{012}-\mathbf{C}_{111}| + 4\,|\mathbf{C}_{021}-\mathbf{C}_{111},\mathbf{C}_{003}-\mathbf{C}_{102}| \\
N_{103} &= 4\,|\mathbf{C}_{012}-\mathbf{C}_{102},\mathbf{C}_{102}-\mathbf{C}_{201}| + 4\,|\mathbf{C}_{111}-\mathbf{C}_{201},\mathbf{C}_{003}-\mathbf{C}_{102}| \\
N_{202} &= |\mathbf{C}_{012}-\mathbf{C}_{102},\mathbf{C}_{201}-\mathbf{C}_{300}| + 6\,|\mathbf{C}_{111}-\mathbf{C}_{201},\mathbf{C}_{102}-\mathbf{C}_{201}| + \\
&\quad |\mathbf{C}_{210}-\mathbf{C}_{300},\mathbf{C}_{003}-\mathbf{C}_{102}| \\
N_{301} &= 4\,|\mathbf{C}_{111}-\mathbf{C}_{201},\mathbf{C}_{201}-\mathbf{C}_{300}| + 4\,|\mathbf{C}_{210}-\mathbf{C}_{300},\mathbf{C}_{102}-\mathbf{C}_{201}|
\end{aligned}
$$

Jacobian Determinant - Triangle

The control points on the interior of the Jacobian determinant triangle are

$$
\begin{aligned}
N_{211} &= 3\,|\mathbf{C}_{111}-\mathbf{C}_{201},\mathbf{C}_{111}-\mathbf{C}_{210}| + |\mathbf{C}_{210}-\mathbf{C}_{300},\mathbf{C}_{012}-\mathbf{C}_{111}| + \\
&\quad |\mathbf{C}_{021}-\mathbf{C}_{111},\mathbf{C}_{201}-\mathbf{C}_{300}| + 3\,|\mathbf{C}_{120}-\mathbf{C}_{210},\mathbf{C}_{102}-\mathbf{C}_{201}| \\
N_{121} &= |\mathbf{C}_{111}-\mathbf{C}_{201},\mathbf{C}_{021}-\mathbf{C}_{120}| + 3\,|\mathbf{C}_{021}-\mathbf{C}_{111},\mathbf{C}_{111}-\mathbf{C}_{210}| + \\
&\quad 3\,|\mathbf{C}_{120}-\mathbf{C}_{210},\mathbf{C}_{012}-\mathbf{C}_{111}| + |\mathbf{C}_{030}-\mathbf{C}_{120},\mathbf{C}_{102}-\mathbf{C}_{201}| \\
N_{112} &= |\mathbf{C}_{012}-\mathbf{C}_{102},\mathbf{C}_{111}-\mathbf{C}_{210}| + 3\,|\mathbf{C}_{111}-\mathbf{C}_{201},\mathbf{C}_{012}-\mathbf{C}_{111}| + \\
&\quad 3\,|\mathbf{C}_{021}-\mathbf{C}_{111},\mathbf{C}_{102}-\mathbf{C}_{201}| + |\mathbf{C}_{120}-\mathbf{C}_{210},\mathbf{C}_{003}-\mathbf{C}_{102}|
\end{aligned}
$$

I wonder if these are always positive, or, if they are negative, then there must be at least one edge control point that is also negative. As in, do I need to ever check these if the interior control point of my geometric shape is well chosen.

Jacobian Determinants - Tet

The Jacobian Determinant is itself a Bézier shape. Consider a tetrahedron, we have

$$\mathbf{J} = \frac{\partial \mathbf{B}(u,v,w)}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial B_x}{\partial x} & \frac{\partial B_x}{\partial y} & \frac{\partial B_x}{\partial z} \\ \frac{\partial B_y}{\partial x} & \frac{\partial B_y}{\partial y} & \frac{\partial B_y}{\partial z} \\ \frac{\partial B_z}{\partial x} & \frac{\partial B_z}{\partial y} & \frac{\partial B_z}{\partial z} \end{bmatrix}$$

As the derivative is one order lower, we get

$$\mathbf{J} = \frac{\partial \mathbf{B}(u,v,w)}{\partial \mathbf{x}} = \begin{bmatrix} \mathcal{O}(P-1) & \mathcal{O}(P-1) & \mathcal{O}(P-1) \\ \mathcal{O}(P-1) & \mathcal{O}(P-1) & \mathcal{O}(P-1) \\ \mathcal{O}(P-1) & \mathcal{O}(P-1) & \mathcal{O}(P-1) \end{bmatrix}$$

And thus the determinant for a tet, denoted $\det(\mathbf{J}) = \mathcal{J}(u,v,w)$, is $\mathcal{J} = \mathcal{O}(3(P-1))$. If the $\mathcal{J} > 0$ everywhere, the element is valid, otherwise it is invalid.

Jacobian Determinants - Tet

To evaluate $\mathcal{J}(u, v, w)$, one option is to compute it directly. Writing the tet as

$$\mathbf{B}(u, v, w) = \sum_{i+j+k+l=P} \frac{P!}{i!j!k!l!} u^i v^j w^k (1-u-v-w)^l \mathbf{C}_{ijkl}$$

and the derivative vector of its coefficients at each control point, with $\mathbf{u} = (u, v, w)$

$$\frac{\partial B(\mathbf{u})_{ijkl}}{\partial \mathbf{u}} = \frac{P!}{i!j!k!l!} \frac{\partial}{\partial \mathbf{u}} \left(u^i v^j w^k (1-u-v-w)^l \right)$$

The Jacobian matrix is

$$\mathbf{J}(\mathbf{u}) = \sum_{i+j+k+l=P} \mathbf{C}_{ijkl} \otimes \frac{\partial B(\mathbf{u})_{ijkl}}{\partial \mathbf{u}}$$

and the determinant is straightforward.

Jacobian Determinants - Tet

The other way to compute it is by first calculating the Bézier form of $\mathcal{J}$, and then evaluating it.

$$\mathcal{J}(u,v,w) = \sum_{I+J+K+L=3(P-1)} \binom{3(P-1)}{I,J,K} u^I v^J w^K (1-u-v-w)^L N_{IJKL}$$

where the control points, N_{IJKL}, (scalars) are sums of determinants of the control points

$$N_{IJKL} = P^3 \sum_{|i|=I,|j|=J,|k|=K,|l|=L} \frac{\binom{P-1}{i_1,j_1,k_1}\binom{P-1}{i_2,j_2,k_2}\binom{P-1}{i_3,j_3,k_3}}{\binom{3(P-1)}{I,J,K}} \times |\mathbf{C}_{i_1+1,j_1,k_1,l_1} - \mathbf{C}_{i_1,j_1+1,k_1,l_1}, \mathbf{C}_{i_2+1,j_2,k_2,l_2} - \mathbf{C}_{i_2,j_2,k_2+1,l_2}, \mathbf{C}_{i_3+1,j_3,k_3,l_3} - \mathbf{C}_{i_3,j_3,k_3,l_3+1}|$$

$$|i| = i_1 + i_2 + i_3, |j| = j_1 + j_2 + j_3, |k| = k_1 + k_2 + k_3, |l| = l_1 + l_2 + l_3$$

$$i_1 + j_1 + k_1 + l_1 = i_2 + j_2 + k_2 + l_2 = i_3 + j_3 + k_3 + l_3 = P - 1$$

For each control point, a large number of determinants are needed.

Jacobian Determinant - Tet

Since $\mathcal{J}$ is order $3(P-1)$, the number of control points N_{IJKL} is much higher than the number of control points of the original shape. In this table, "Terms" is the number of determinants calculated to compute the control points.

Bezier Tetrahedron		Jacobian Determinant		
Order	Num Ctrl Pts	Order	Num Ctrl Pts	Terms
1	4	0	1	1
2	10	3	20	64
3	20	6	84	1000
4	35	9	220	8000
5	56	12	455	42875
6	84	15	816	175616
7	120	18	1330	592704

Jacobian Determinant - Tet

For the quadratic tet with 10 control points, there are 20 control points for $\mathcal{J}$. Here are three of these control points, one on a corner, one on an edge, and one on a face of $\mathcal{J}$

$$
\begin{aligned}
N_{3000} &= 8\,|\mathbf{C}_{1100}-\mathbf{C}_{2000},\mathbf{C}_{1010}-\mathbf{C}_{2000},\mathbf{C}_{1001}-\mathbf{C}_{2000}| \\
N_{2100} &= 2\,|\mathbf{C}_{1100}-\mathbf{C}_{2000},\mathbf{C}_{1010}-\mathbf{C}_{2000},\mathbf{C}_{0101}-\mathbf{C}_{1100}| + \\
&\quad\ 2\,|\mathbf{C}_{1100}-\mathbf{C}_{2000},\mathbf{C}_{0110}-\mathbf{C}_{1100},\mathbf{C}_{1001}-\mathbf{C}_{2000}| + \\
&\quad\ 2\,|\mathbf{C}_{0200}-\mathbf{C}_{1100},\mathbf{C}_{1010}-\mathbf{C}_{2000},\mathbf{C}_{1001}-\mathbf{C}_{2000}| \\
N_{1110} &= |\mathbf{C}_{1100}-\mathbf{C}_{2000},\mathbf{C}_{0110}-\mathbf{C}_{1100},\mathbf{C}_{0011}-\mathbf{C}_{1010}| + \\
&\quad\ |\mathbf{C}_{0200}-\mathbf{C}_{1100},\mathbf{C}_{1010}-\mathbf{C}_{2000},\mathbf{C}_{0011}-\mathbf{C}_{1010}| + \\
&\quad\ |\mathbf{C}_{1100}-\mathbf{C}_{2000},\mathbf{C}_{0020}-\mathbf{C}_{1010},\mathbf{C}_{0101}-\mathbf{C}_{1100}| + \\
&\quad\ |\mathbf{C}_{0200}-\mathbf{C}_{1100},\mathbf{C}_{0020}-\mathbf{C}_{1010},\mathbf{C}_{1001}-\mathbf{C}_{2000}| + \\
&\quad\ |\mathbf{C}_{0110}-\mathbf{C}_{1010},\mathbf{C}_{1010}-\mathbf{C}_{2000},\mathbf{C}_{0101}-\mathbf{C}_{1100}| + \\
&\quad\ |\mathbf{C}_{0110}-\mathbf{C}_{1010},\mathbf{C}_{0110}-\mathbf{C}_{1100},\mathbf{C}_{1001}-\mathbf{C}_{2000}|
\end{aligned}
$$

Jacobian Determinant - Tet

For the cubic tet with 20 control points, there are 84 control points for $\mathcal{J}$. 10 of these are on the "inside", 4 for the corners, and 6 for the edges. Heres one for a corner, which has 24 terms. The edge ones have 33.

$$
\begin{aligned}
N_{1113} = \quad & |\mathbf{C}_{0102} - \mathbf{C}_{1002}, \mathbf{C}_{1011} - \mathbf{C}_{2001}, \mathbf{C}_{0111} - \mathbf{C}_{1110}| + \\
& |\mathbf{C}_{1101} - \mathbf{C}_{2001}, \mathbf{C}_{0012} - \mathbf{C}_{1002}, \mathbf{C}_{0111} - \mathbf{C}_{1110}| + \\
& |\mathbf{C}_{0102} - \mathbf{C}_{1002}, \mathbf{C}_{0111} - \mathbf{C}_{1101}, \mathbf{C}_{1011} - \mathbf{C}_{2010}| + \\
& |\mathbf{C}_{0102} - \mathbf{C}_{1002}, \mathbf{C}_{1110} - \mathbf{C}_{2100}, \mathbf{C}_{0012} - \mathbf{C}_{1011}| + \\
& |\mathbf{C}_{1101} - \mathbf{C}_{2001}, \mathbf{C}_{0111} - \mathbf{C}_{1101}, \mathbf{C}_{0012} - \mathbf{C}_{1011}| + \\
& |\mathbf{C}_{0201} - \mathbf{C}_{1101}, \mathbf{C}_{0012} - \mathbf{C}_{1002}, \mathbf{C}_{1011} - \mathbf{C}_{2010}| + \\
& |\mathbf{C}_{0201} - \mathbf{C}_{1101}, \mathbf{C}_{1011} - \mathbf{C}_{2001}, \mathbf{C}_{0012} - \mathbf{C}_{1011}| + \\
& |\mathbf{C}_{1200} - \mathbf{C}_{2100}, \mathbf{C}_{0012} - \mathbf{C}_{1002}, \mathbf{C}_{0012} - \mathbf{C}_{1011}| + \\
& |\mathbf{C}_{0102} - \mathbf{C}_{1002}, \mathbf{C}_{0021} - \mathbf{C}_{1011}, \mathbf{C}_{1101} - \mathbf{C}_{2100}| + \\
& |\mathbf{C}_{0102} - \mathbf{C}_{1002}, \mathbf{C}_{1020} - \mathbf{C}_{2010}, \mathbf{C}_{0102} - \mathbf{C}_{1101}| + \\
& |\mathbf{C}_{1101} - \mathbf{C}_{2001}, \mathbf{C}_{0021} - \mathbf{C}_{1011}, \mathbf{C}_{0102} - \mathbf{C}_{1101}| + \\
& |\mathbf{C}_{0102} - \mathbf{C}_{1002}, \mathbf{C}_{0120} - \mathbf{C}_{1110}, \mathbf{C}_{1002} - \mathbf{C}_{2001}| + \ldots
\end{aligned}
$$

I raise the same question as before, can those 10 be negative without an edge or face control point being negative. Are they bound by the other control points?

Table of Contents

Determining Validity

The convex hull property guarantees that if all control points of the Jacobian determinant are positive, the Jacobian determinant is positive everywhere, and thus the corresponding entity provably valid. For triangles (tetrahedra) the condition is

$$N_{IJK(L)} > 0, \text{for all control points}$$

The contrary, however is not true. If one control point is negative, the Jacobian determinant is not necessarily negative everywhere, and thus the entity may be valid.

We use adaptive algorithms, converging the control points of the Jacobian determinant until we are confident of their sign.

Determining Validity

There are two approaches to adapting.

1. **Subdivision**. Any Bezier can be split into multiple Bezier shapes that are identical to the original shape, with new control points.
2. **Elevation**. Any Bezier shape can be elevated in order, preserving the original shape, with new control points.

Since we have that the bounding shapes of Bezier shapes are themselves Beziers, that is, the edges of a Bezier triangle are Bezier edges and so forth. This reduces the cost of the adaptive algorithm - e.g, if a coefficient of the Jacobian determinant on the edge is negative, we can simply adapt that edge repeatedly.

Validity Check - Subdivision of an Edge

Consider an invalid triangle or tetrahedron, with one N negative, corresponding to an edge control point of the Jacobian determinant.

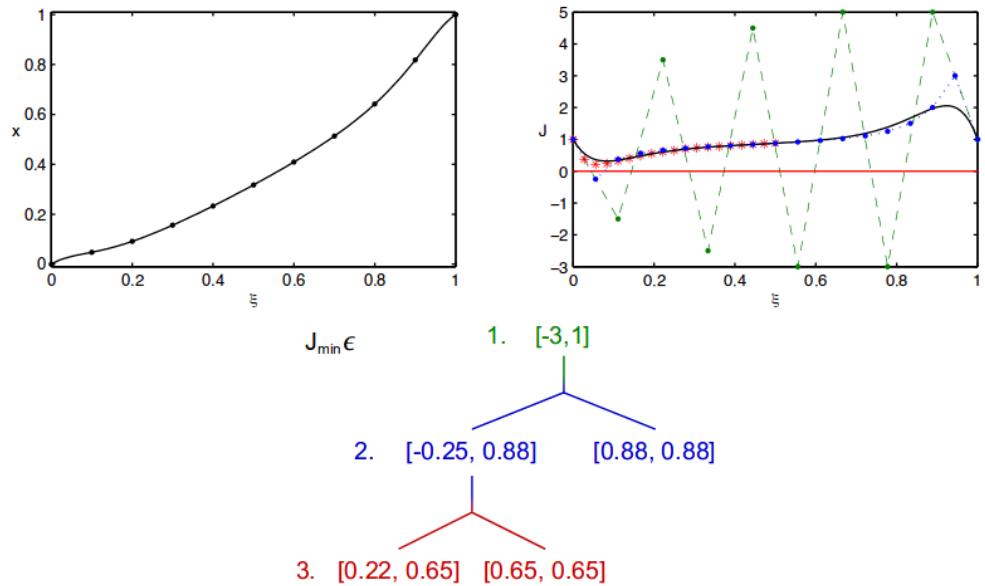


Fig. 4. Top left: One-dimensional element mapping $x(\xi)$. Top right: Exact Jacobian determinant $J(\xi)$ (black), control values on the original control points (green) and two adaptive subdivisions (blue and red). Bottom: Estimates of $J_{\min}$ at each step in the adaptive subdivision process.

Figure from *The Generation of Valid Curvilinear Meshes*, Geuzaine et al.

Validity Check - Subdivision of an Triangle

Consider an invalid triangle or tetrahedron, with one N negative, corresponding to a triangle control point of the Jacobian determinant.

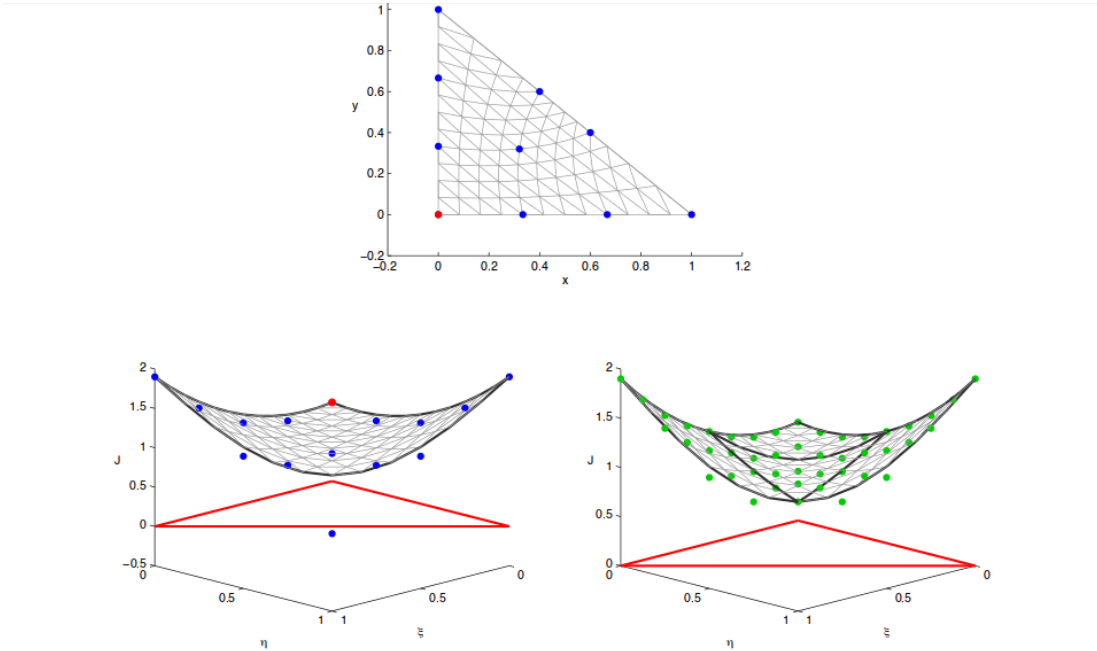


Fig. 5. Top: third-order planar triangle. Bottom left: exact Jacobian determinant and control values (dots) on the original control points; the validity of the element cannot be asserted. Bottom right: exact Jacobian determinant and control values (dots) after one subdivision; the element is provably correct.

Figure from *The Generation of Valid Curvilinear Meshes*, Geuzaine et al.

Validity Check - Algorithm

The algorithm is divided into two parts, a first screening, and a second, adaptive part (*The Generation of Valid Curvilinear Meshes*, Geuzaine et al.).

The first part computes $\mathscr{J}$ at a prescribed set of $\mathbf{u}$, and if a negative value is found, the element is invalid. If all of these are positive, continue on to the adaptive part, to be sure. The adaptive part can use either elevation or subdivision.

Validity Check - Initial

```
1: function CheckValidity(entity)
2:     compute B_i for entity
3:     if corner B_i < 0 then
4:         return INVALID, vertex(es)                ▷ Adaptation doesn't improve this
5:     end if
6:     if min B_i > 0 then                           ▷ may want B_i > constant
7:         return VALID, empty
8:     else
9:         find B_i < 0, determine edges, triangles to adapt
10:        for each lower entity to adapt do
11:            compute B_i for that entity
12:            if CheckValidityAdaptive(B_{i,entity}, ϵ,0) is False then
13:                entities ← entity
14:            end if
15:        end for
16:    end if
17:    if entities is empty then
18:        return VALID
19:    else
20:        return INVALID
21:    end if
22: end function
```

Validity Check - Adaptive

```
1: function CheckValidityAdaptive(B_{i,entity}, ϵ,nIter)
2:     if min B_i > 0 then
3:         return VALID
4:     else if nIter > maxIter or change in min B_i < ϵ then      ▷ Convergence check
5:         return INVALID
6:     else
7:         determine subdivisions or elevations
8:         for each entity in subdivisions or elevations do
9:             nIter ← nIter + 1
10:             checkValidityAdaptive(B_{i,entity}, ϵ,nIter)
11:         end for
12:     end if
13: end function
```

Table of Contents

G^1 Continuous Patches

G1 continuous patches are implemented following Walton and Meek, *A triangular G1 patch from boundary curves*. The mesh is initially set to 4th order, with 6 face nodes for the G1 patch. First, all edges on a geometric surface have control points set (the first two nodes of the three) using surface normals and the vertex locations. Edges on geometric edges have their control points set using the tangents at their endpoints. These edge control points use the algorithm of Walton and Meek to set the six face nodes. Once the internal points are determined, the edges are formally elevated, by resetting all of the points to the elevated control points.

G^1 Continuous Patches

The weights on these nodes are similar to the triangular Bezier, with the evaluation done as if it is a Bezier triangle with internal Bezier points, $\mathbf{B}$, and Gregory points, $\mathbf{G}$.

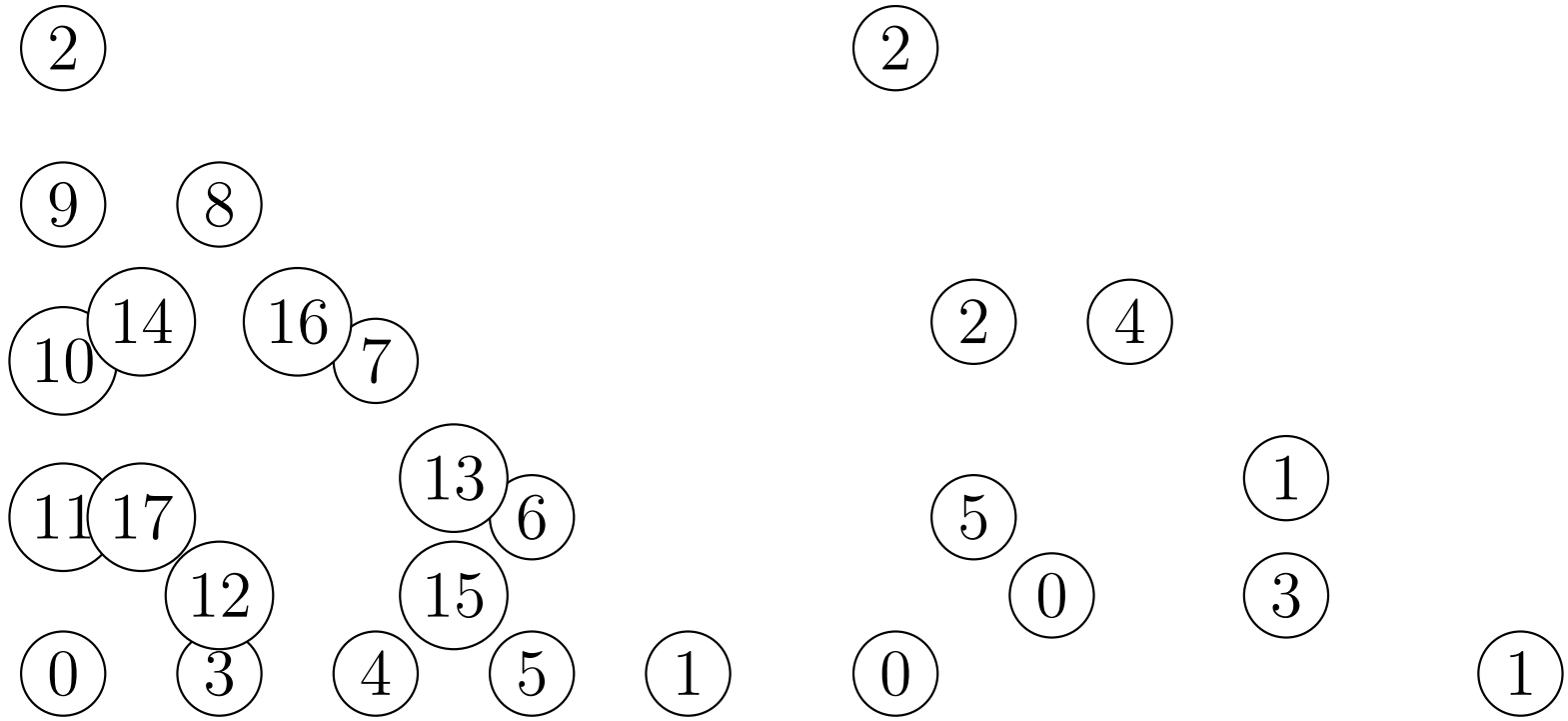
$$\mathbf{B}_{12} = \frac{1}{\xi_1 + \xi_2}(\xi_2 \mathbf{G}_{12} + \xi_1 \mathbf{G}_{17}) \tag{1}$$

$$\mathbf{B}_{13} = \frac{1}{\xi_0 + \xi_2}(\xi_0 \mathbf{G}_{13} + \xi_2 \mathbf{G}_{15}) \tag{2}$$

$$\mathbf{B}_{14} = \frac{1}{\xi_0 + \xi_1}(\xi_1 \mathbf{G}_{14} + \xi_0 \mathbf{G}_{16}) \tag{3}$$

The remainder of the mathematics is the same as the Bezier triangle, with tetrahedral blending, and other aspects maintaining the same form. We should note that choosing $\mathbf{G}_{12} = \mathbf{G}_{17}, \mathbf{G}_{13} = \mathbf{G}_{15}, \mathbf{G}_{14} = \mathbf{G}_{16}$ results in the original Bezier triangle, albeit with three additional nodes stored per triangle.

G^1 Continuous Patches



Global and local node ordering for the 4th order G1 triangle. The ordering is chosen as these points are set based on edges, so node 0 and node 3 correspond to edge 0 and so forth.

Table of Contents

Curves

For curves, there is one free parameter, u (v $=$ 1-u), between 0 and 1. In finite element convention, the parameter, $\xi \in [-1, 1]$, such that

$$u = \frac{1}{2}(\xi + 1)$$

When taking derivatives we have that

$$\frac{\mathrm{d}\mathbf{B}(u)}{\mathrm{d}\xi} = \frac{\mathrm{d}\mathbf{B}(u)}{\mathrm{d}u}\frac{\mathrm{d}u}{\mathrm{d}\xi} = \frac{1}{2}\frac{\mathrm{d}\mathbf{B}(u)}{\mathrm{d}u}$$

Triangles

For triangles, there are two free parameters, $u, v, (w = 1 - u - v)$, between 0 and 1. In finite element convention, the parameters, $\xi_0, \xi_1 \in [0, 1]$, are such that $\xi_0 = 1$, corresponds to vertex 1, $\xi_1 = 1$ to vertex 2, such that the barycentric coordinates are

$$(u, v, w) = (1 - \xi_0 - \xi_1, \xi_0, \xi_1)$$

When taking derivatives, writing $\mathbf{B}(\xi_0, \xi_1) = \mathbf{B}(u, v, w)$

$$\frac{\mathrm{d}\mathbf{B}}{\mathrm{d}\xi_0} = \frac{\mathrm{d}\mathbf{B}}{\mathrm{d}u}\frac{\mathrm{d}u}{\mathrm{d}\xi_0} + \frac{\mathrm{d}\mathbf{B}}{\mathrm{d}v}\frac{\mathrm{d}v}{\mathrm{d}\xi_0} = -\frac{\mathrm{d}\mathbf{B}}{\mathrm{d}u} + \frac{\mathrm{d}\mathbf{B}}{\mathrm{d}v}$$

$$\frac{\mathrm{d}\mathbf{B}}{\mathrm{d}\xi_1} = -\frac{\mathrm{d}\mathbf{B}}{\mathrm{d}u} + \frac{\mathrm{d}\mathbf{B}}{\mathrm{d}w}$$

Tetrahedra

For tetrahedra, there are three free parameters, $u, v, w, (t = 1 - u - v - w)$, between 0 and 1. In finite element convention, the parameters, $\xi_0, \xi_1, \xi_2 \in [0, 1]$

$$(u, v, w, t) = (1 - \xi_0 - \xi_1 - \xi_2, \xi_0, \xi_1, \xi_2)$$

When taking derivatives we have that, similar to triangles,

$$\frac{\mathrm{d}\mathbf{B}}{\mathrm{d}\xi_0} = \frac{\mathrm{d}\mathbf{B}}{\mathrm{d}u}\frac{\mathrm{d}u}{\mathrm{d}\xi_0} + \frac{\mathrm{d}\mathbf{B}}{\mathrm{d}v}\frac{\mathrm{d}v}{\mathrm{d}\xi_0} + \frac{\mathrm{d}\mathbf{B}}{\mathrm{d}w}\frac{\mathrm{d}w}{\mathrm{d}\xi_0} + \frac{\mathrm{d}\mathbf{B}}{\mathrm{d}w}\frac{\mathrm{d}t}{\mathrm{d}\xi_0} = -\frac{\mathrm{d}\mathbf{B}}{\mathrm{d}u} + \frac{\mathrm{d}\mathbf{B}}{\mathrm{d}v}$$

$$\frac{\mathrm{d}\mathbf{B}}{\mathrm{d}\xi_1} = -\frac{\mathrm{d}\mathbf{B}}{\mathrm{d}u} + \frac{\mathrm{d}\mathbf{B}}{\mathrm{d}w}$$

$$\frac{\mathrm{d}\mathbf{B}}{\mathrm{d}\xi_2} = -\frac{\mathrm{d}\mathbf{B}}{\mathrm{d}u} + \frac{\mathrm{d}\mathbf{B}}{\mathrm{d}t}$$

Table of Contents

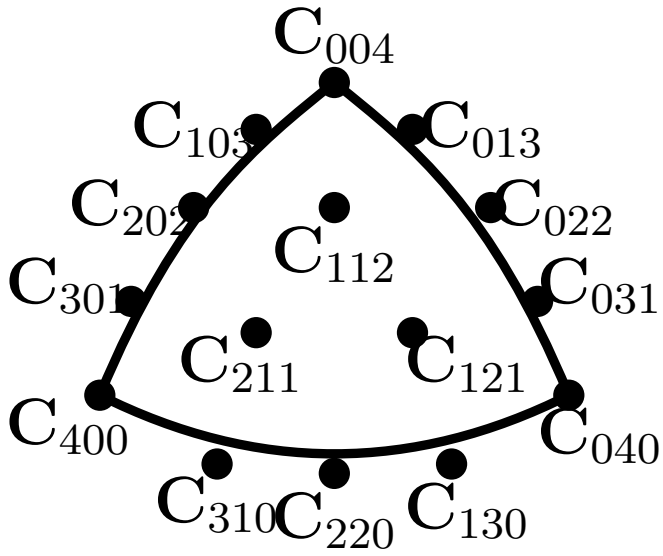
Ordering - Curves

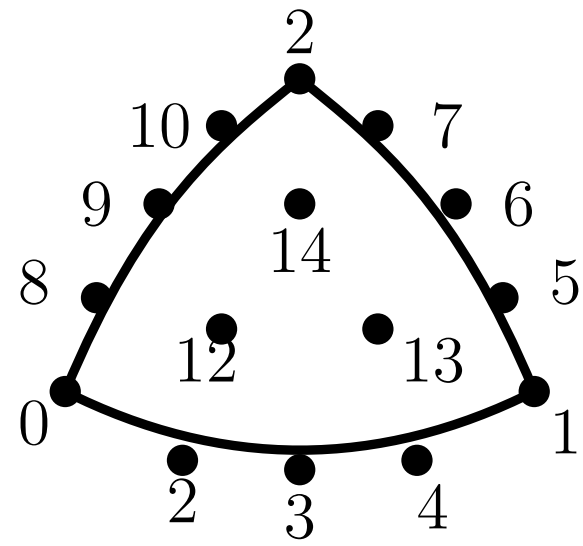
Due to shared entities, there is a specific order to store coordinates in a 1D Array. This ordering is used in `apfShape.h` and nodes from `apf::getVectorNodes` return the coordinates in this specific ordering. The ordering is stored in `crvTables.cc`, e.g for triangles in `getTriNodeIndex(P,i,j)`.

For curves, this is simple. The vertex control points are indexed into 0 and 1, with the remaining points ordered as in the figure below, for a third order curve.

Ordering - Triangles

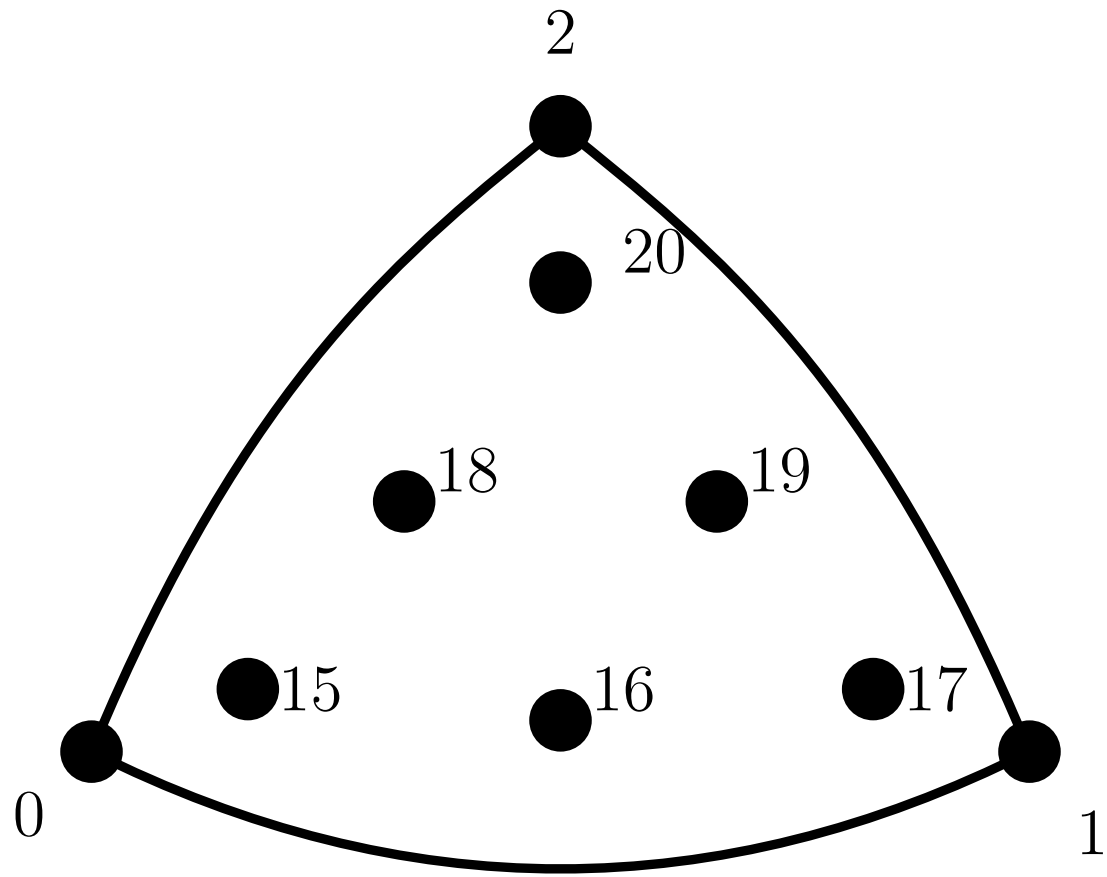
For triangles, this is more complex. The vertices come first, followed by the edge control points, followed by the triangle's internal points.





Ordering - Triangles

On the interior, the points are ordered in a zig zag pattern, e.g for a 5^{th} order triangle



Ordering - Tetrahedron

The ordering for a tetrahedron is similar to that of a triangle, with vertices, edges, faces, and then internal points. For a cubic tetrahedron (no internal control points), the ordering is as below, based on the canonical vertex, edge, and face ordering for a linear tetrahedron.

$\mathbf{C}_3\ \mathbf{C}_{15}\mathbf{C}_{14}\mathbf{C}_2$	$\mathbf{C}_{0003}\mathbf{C}_{0012}\mathbf{C}_{0021}\mathbf{C}_{0030}$
$\mathbf{C}_{13}\mathbf{C}_{18}\mathbf{C}_7$	$\mathbf{C}_{0102}\mathbf{C}_{0111}\mathbf{C}_{0120}$
$\mathbf{C}_{12}\mathbf{C}_6$	$\mathbf{C}_{0201}\mathbf{C}_{0210}$
$\mathbf{C}_1$	$\mathbf{C}_{0300}$
$\mathbf{C}_{11}\mathbf{C}_{19}\mathbf{C}_8$	$\mathbf{C}_{1002}\mathbf{C}_{1011}\mathbf{C}_{1020}$
$\mathbf{C}_{17}\mathbf{C}_{16}$	$\mathbf{C}_{1101}\mathbf{C}_{1110}$
$\mathbf{C}_5$	$\mathbf{C}_{1200}$
$\mathbf{C}_{10}\mathbf{C}_9$	$\mathbf{C}_{2001}\mathbf{C}_{2010}$
$\mathbf{C}_4$	$\mathbf{C}_{2100}$
$\mathbf{C}_0$	$\mathbf{C}_{3000}$

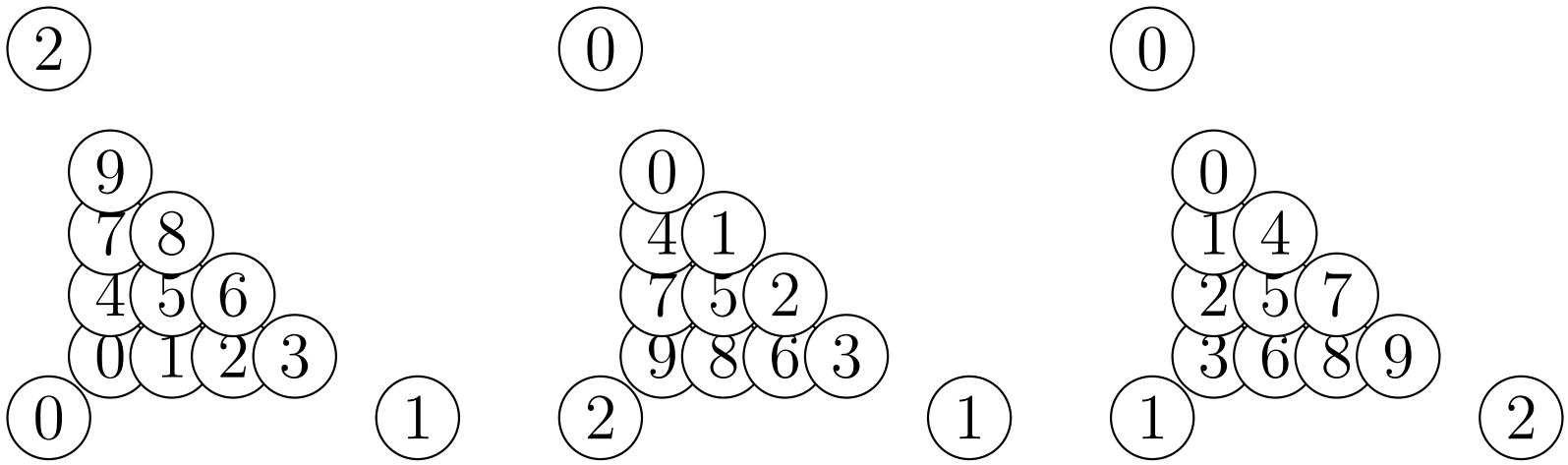
Aligning Shared Nodes

When obtaining the nodes for a particular triangle, specific ordering must be considered. This is due to edges not necessarily being oriented with the face. In `alignSharedNodes`, this problem is addressed by using `getAlignment` and specifying the order. If an edge is flipped with respect to the triangle, the ordering of the edge nodes is reversed and the correct ordering of nodes for the triangle is obtained.

Aligning Shared Nodes

Ordering of edge nodes is the same as for a triangle, reversing them if they are flipped relative to the tetrahedron. For the face nodes in the tetrahedron, these can be flipped and rotated. The correct re-ordering is implemented in a more general form, but uses the fact the interior nodes form loops around the vertices, and rotation is just a shift by the number of nodes per edge. Flipped faces are less obvious, but the correct ordering is obtained by simply examining the flipped triangle's nodes, and matching the ordering (in the example below, node 6 is where node 0 is in the canonical triangle, so the ordering starts with 6). Examples are shown below.

Aligning Shared Nodes



The above diagram describes shared nodes for 6th order Bezier triangle. Original Ordering (left) (0,...,9). Flipped ordering (middle) (9,8,6,3,7,5,2,4,1,0). Rotated ordering by 2 (right) (3,6,8,9,2,5,7,1,4,0). All of this information is in `crvTables.cc`.

Split Point Locations on Triangles

This is an implementation detail used for interpolation points.

Given barycentric coordinates $u, v, w = 1 - u - v$ such that on a triangle in parameter space, $\mathbf{p} = u\mathbf{p}_0 + v\mathbf{p}_1 + w\mathbf{p}_2$ we are interested in splitting it at a given (u, v, w). Due to denegeracy and periodicity, lets define this as a pair of linear splits, such that $\mathbf{q} = a\mathbf{p}_0 + (1 - a)\mathbf{p}_1$ and $\mathbf{p} = b\mathbf{p}_2 + (1 - b)\mathbf{q}$. Solving for (a, b) gives $a = v/(u + v), b = 1 - u - v$. This is implemented in `crvSnap.cc`.