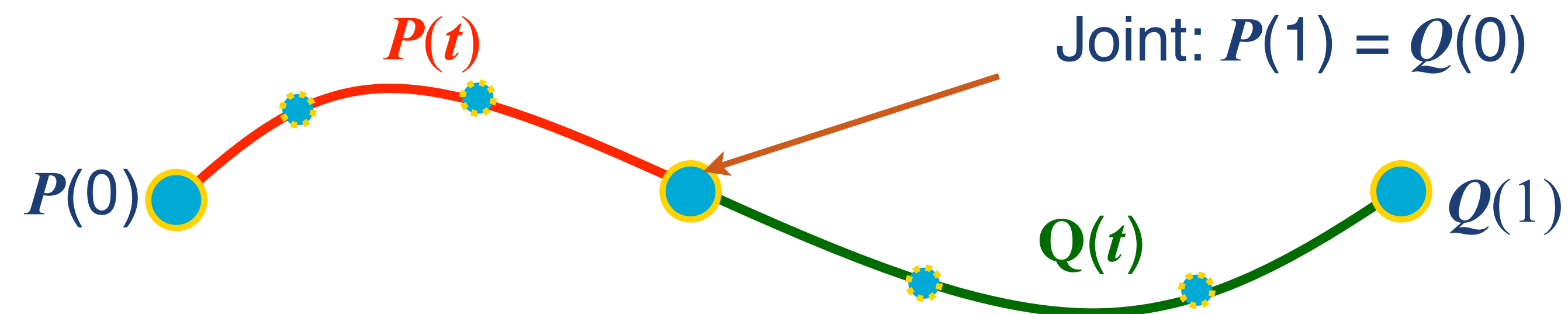




Spline = Joined Curve Segments

- Join cubic curves P and Q
 - ➔ Each interpolates four points
 - ➔ But now we have a seven point curve
- Easy to evaluate, given t
 - ➔ And which component curve to evaluate
- But what about smoothness at the joint?



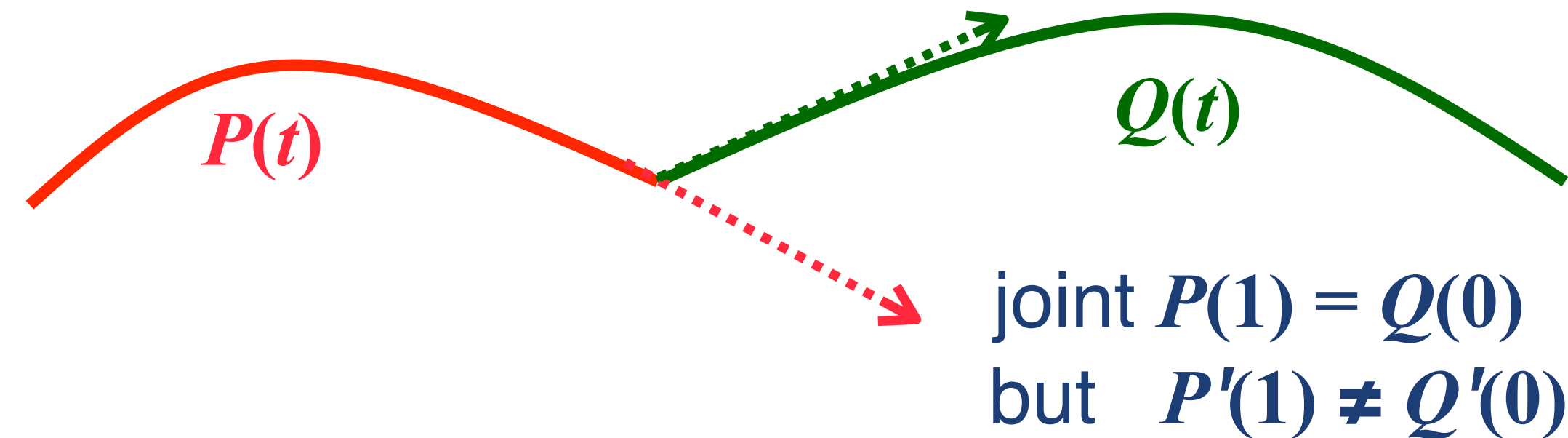
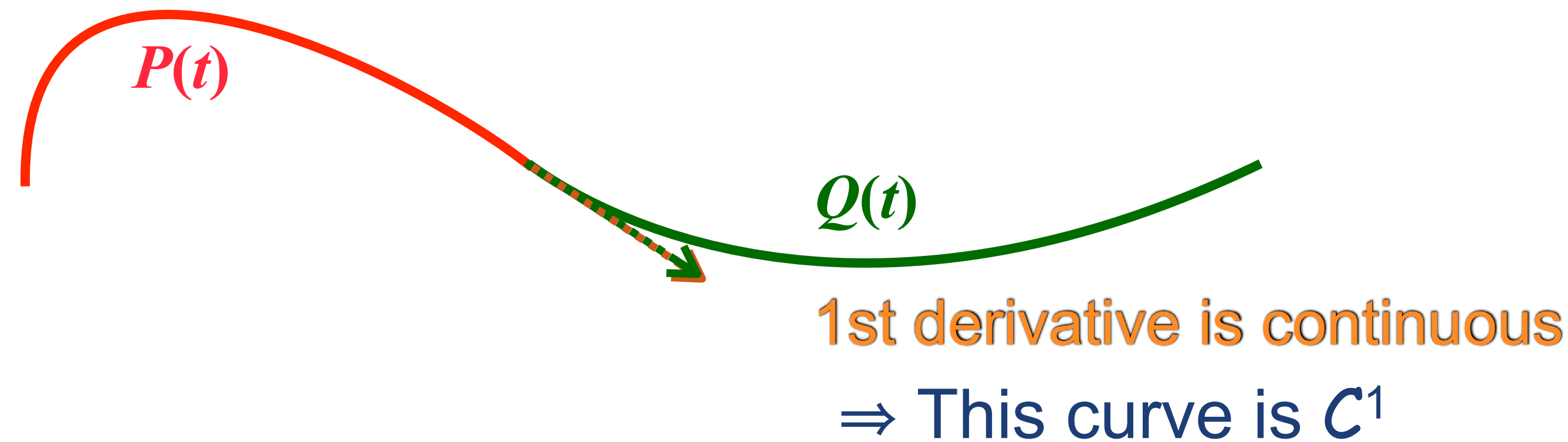


Polynomial Spline: Continuity

- Define continuity at joints

- ➔ $C^i := i^{\text{th}}$ derivative is continuous

- ➔ must be so in all coordinates



i.e., 1st derivative is not continuous

⇒ This curve is C^0

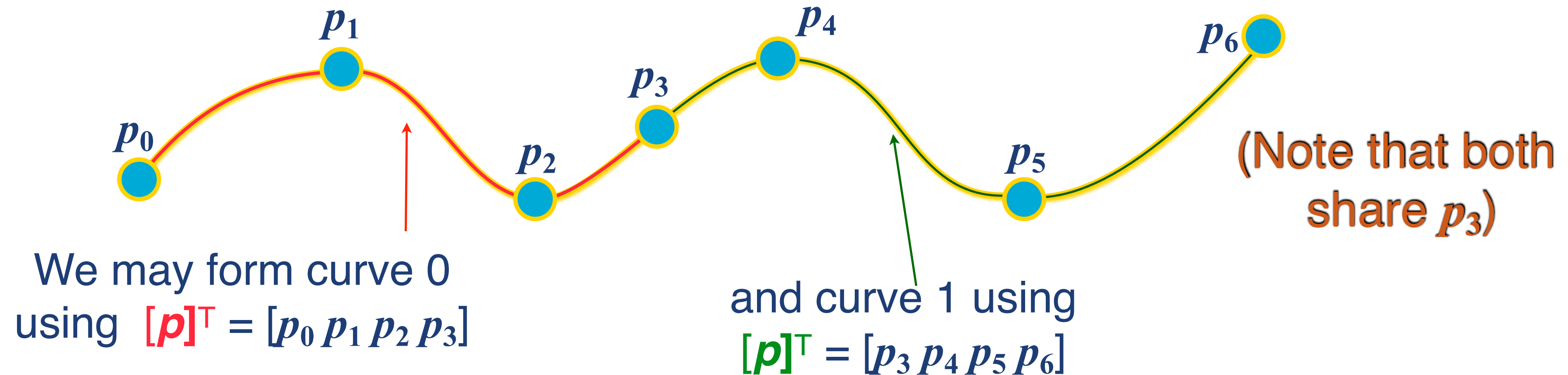
This is parametric continuity.
A related concept is
Geometric continuity.

$G^1 \Rightarrow$ tangent in the same direction



Multi-Segment Interpolation

Suppose 7 data points are given, and we want a cubic spline:



and separately find coefficients $[c]$ for each interpolating curve?

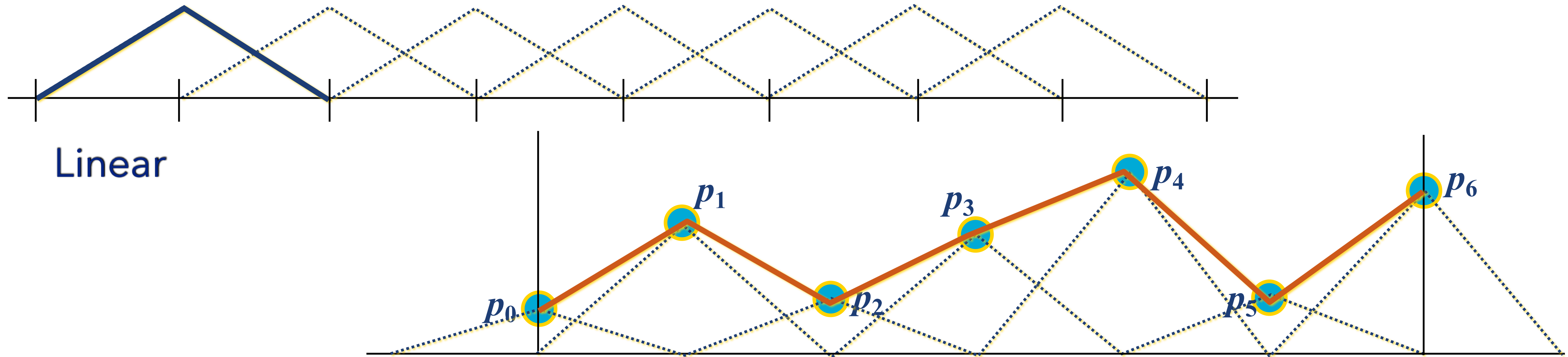
The overall curve will be guaranteed to be interpolating and C^0

To make it C^1 , we will need a way to compute (and use) tangents:

$$\frac{d}{dt} P(t)$$

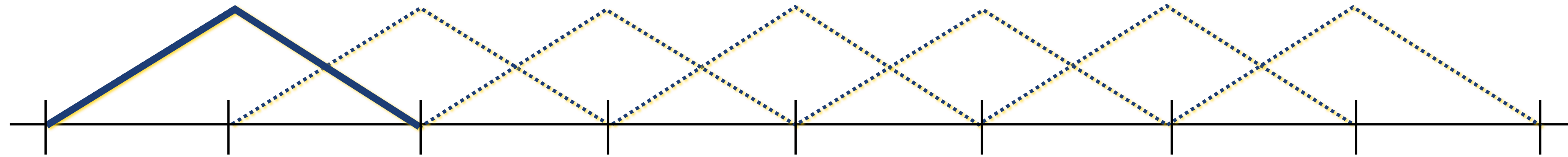


Basis

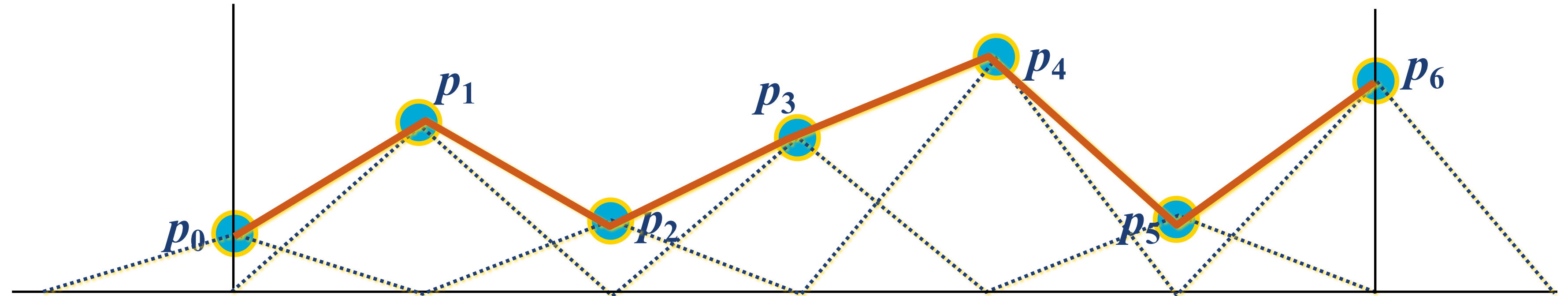




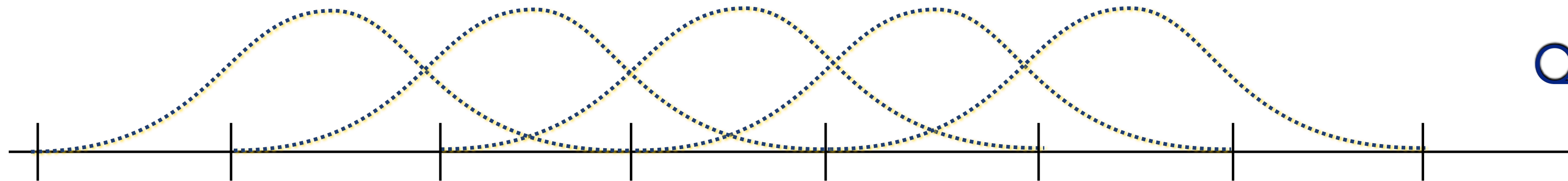
Basis



Linear

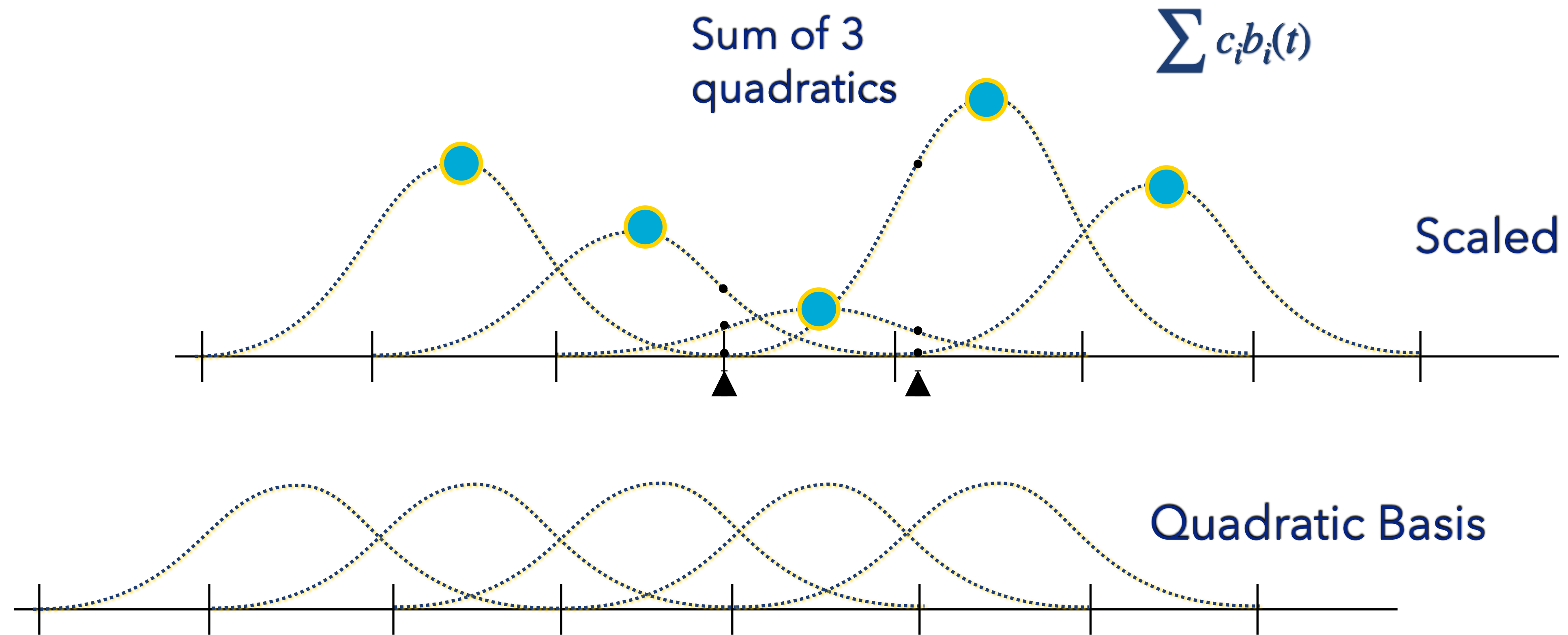


Quadratic



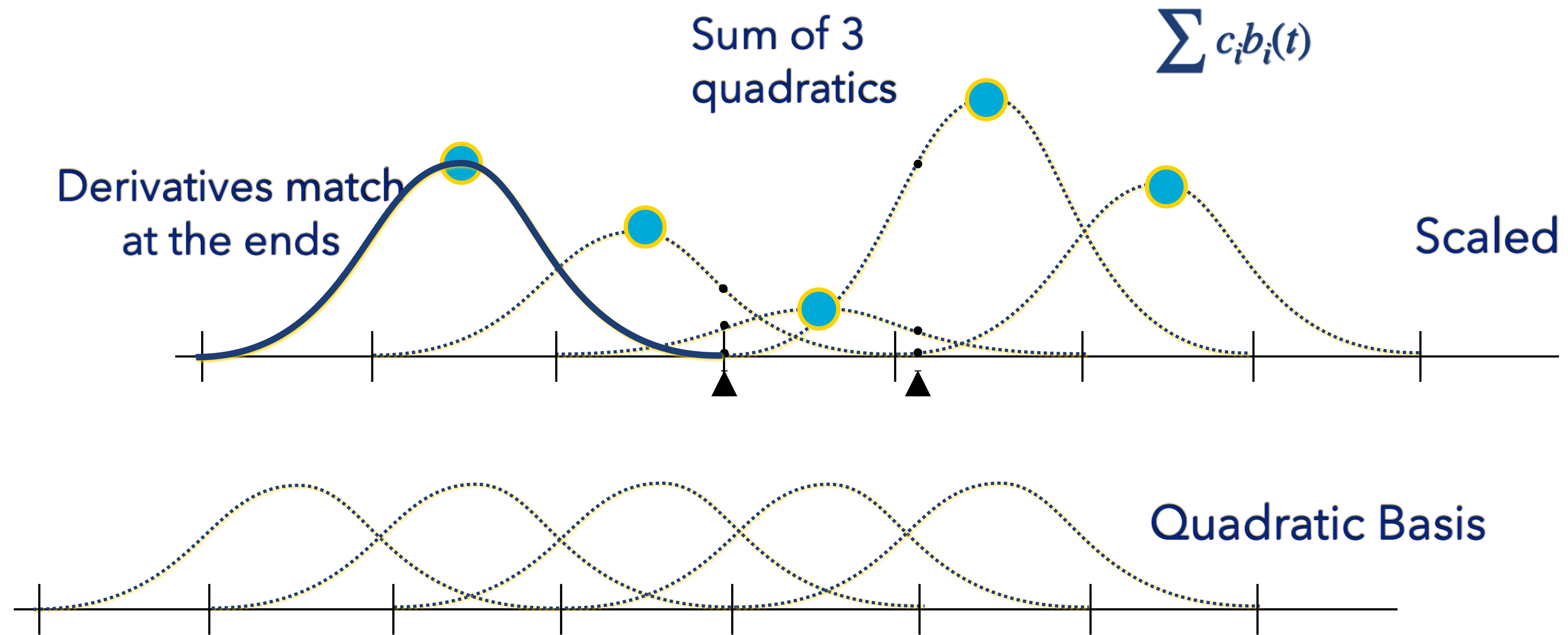


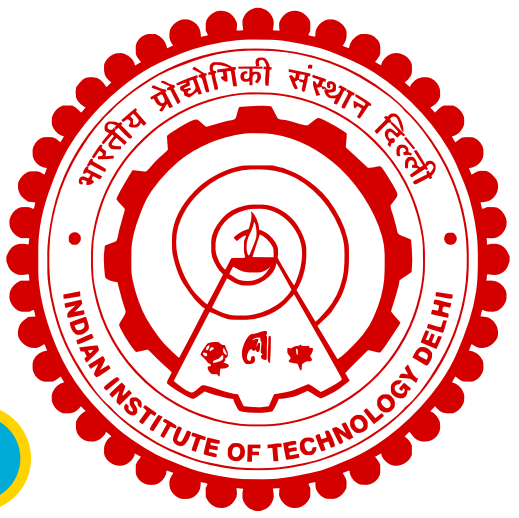
Basis





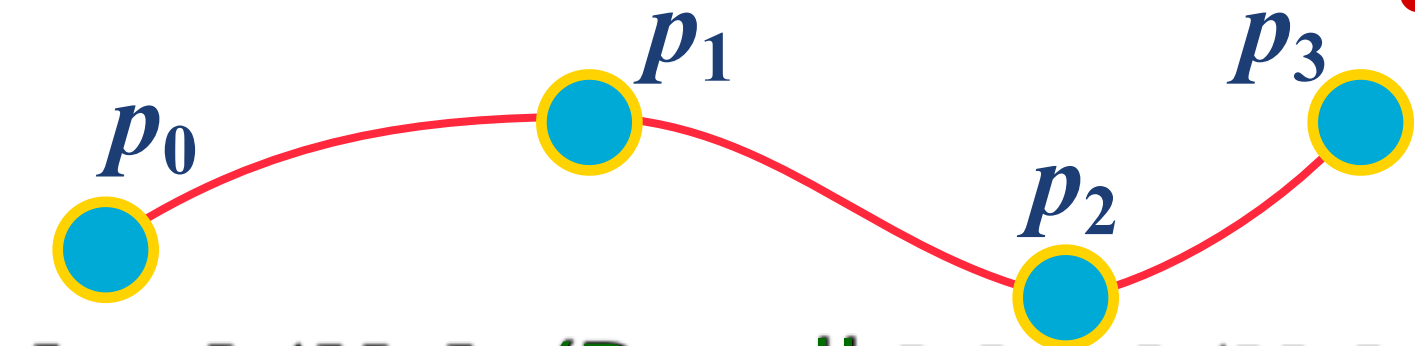
Basis





Cubic Blending Functions

Recall: $P(t) = \sum_{k=0}^3 (c_k t^k)$, with $[c] = [A]^{-1}[p]$ (Recall $[p] = [A][c]$ (Recall $[p] = [A][c]$ from Eq 1)



$\Rightarrow$ Rewrite $P(t)$ as: $[t]^T [c] = ([t]^T A^{-1}) [p] = [b]^T [p] = [p]^T [b]$

$$= \sum_{k=0}^3 (p_k b_k(t))$$

(b_k are called basis functions, aka blending functions)

Note:

1. $b_k(t)$ is 0 at all $t_i, i \neq k$
 $\rightarrow$ multiplier ensures $b_k(t) = 1$ at t_k
 $\checkmark$ the curve interpolates p_k at t_k
2. $P(t)$ is a linear combination of p_k
 $\checkmark$ All b_k smooth $\Rightarrow P$ is smooth

3. $b_k(t)$ partition unity, i.e.,

$$\sum_{k=0}^3 b_k(t) = 1 \forall t$$

$\rightarrow P(t)$ is a weighted average of p_k

Solve for $[c]$ and rewrite to get:

$$b_0(t) = -4.5 (t^{-1/3})(t^{-2/3})(t-1)$$

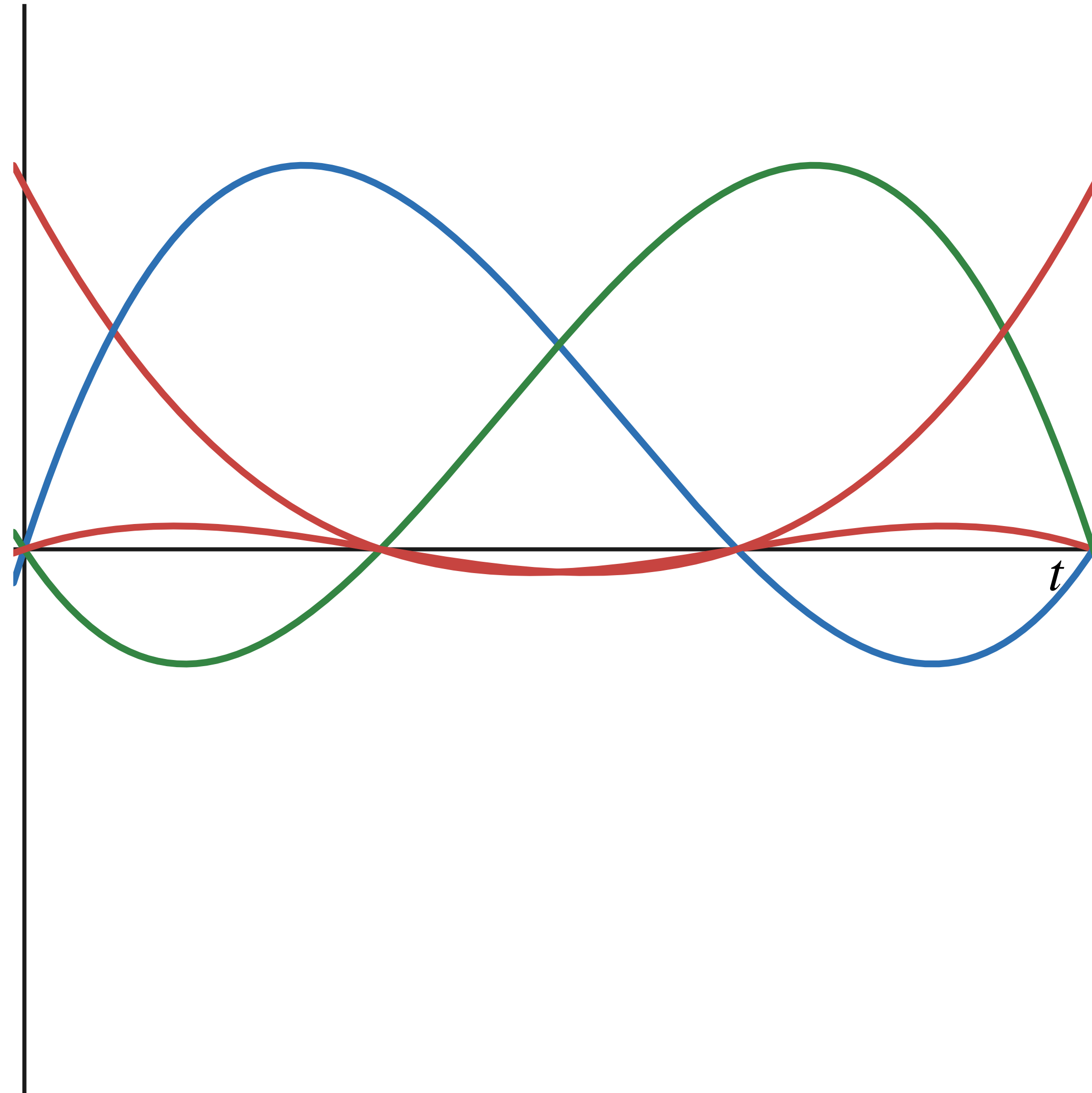
$$b_1(t) = 13.5 t (t^{-2/3})(t-1)$$

$$b_2(t) = -13.5 t (t^{-1/3})(t-1)$$

$$b_3(t) = 4.5 t (t^{-1/3})(t^{-2/3})$$



Blending Function Shape



$$b_0(t) = -4.5 (t^{-1/3})(t^{-2/3})(t-1)$$

$$b_1(t) = 13.5 t (t^{-2/3})(t-1)$$

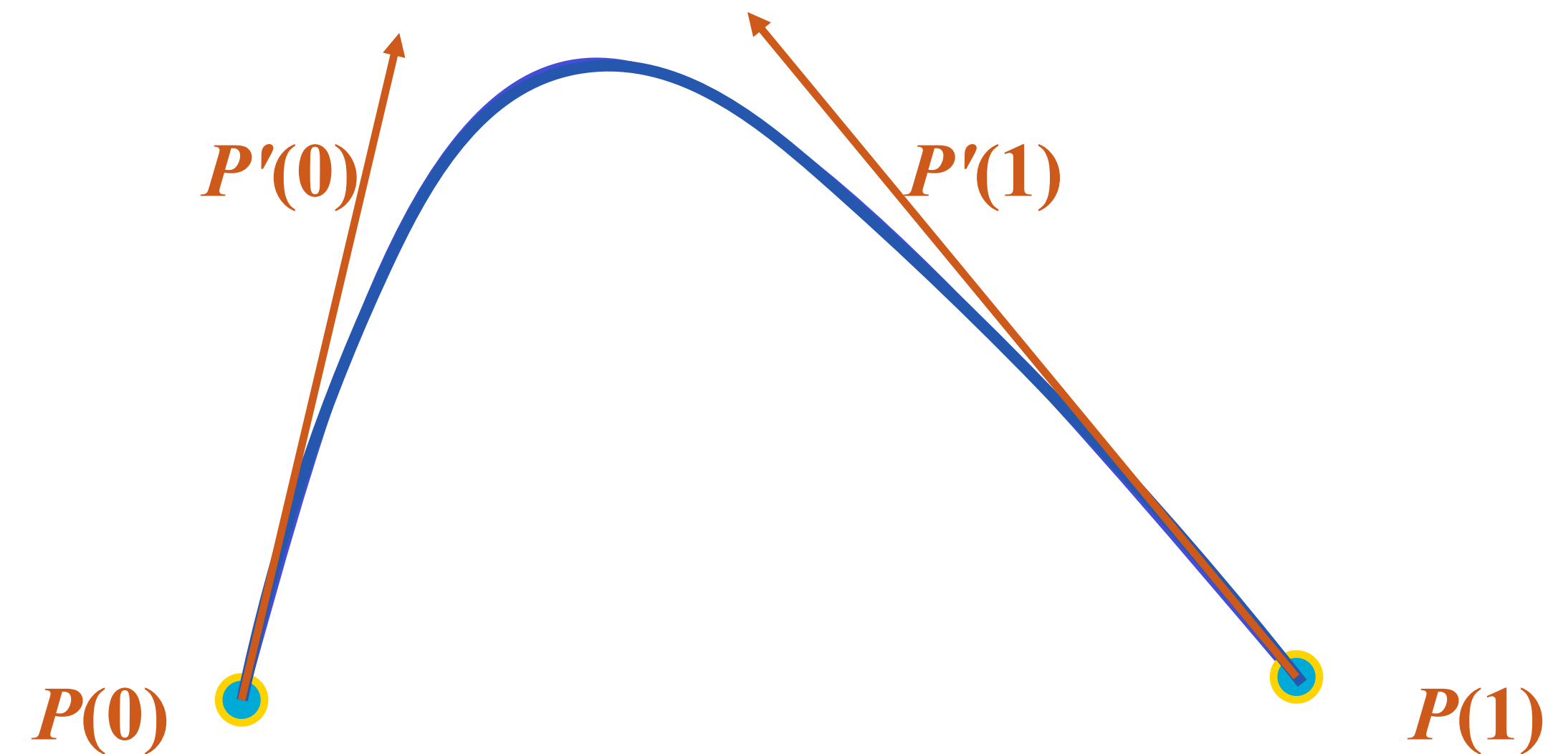
$$b_2(t) = -13.5 t (t^{-1/3})(t-1)$$

$$b_3(t) = 4.5 t (t^{-1/3})(t^{-2/3})$$



Hermite Form

Instead of interpolating a cubic through four points,
we want a cubic between two end-points,
but it should have specified tangents at those end-points.



Four conditions to set up equations:

- Match the given two end-points: $P(0)$ and $P(1)$
- Match their given derivatives: $P'(0)$ and $P'(1)$

The spline is C^1 if these derivatives match the next curve at each joint



Hermite Equations

Suppose $P(t) = c_0 + tc_1 + t^2c_2 + t^3c_3$ (power basis)

Hence: $P(0) = p_0 = c_0$
 $P(1) = p_3 = c_0 + c_1 + c_2 + c_3$ (retaining the notation that the end-point is p_3 , for now)

And: $P'(t) = c_1 + 2tc_2 + 3t^2c_3$

Hence: $P'(0) = p_0' = c_1$
 $P'(1) = p_3' = c_1 + 2c_2 + 3c_3$

Four linear equations in four unknowns: c_0, c_1, c_2, c_3
 p_0, p_3, p_0' , and p_3' are given

derivatives at end-points

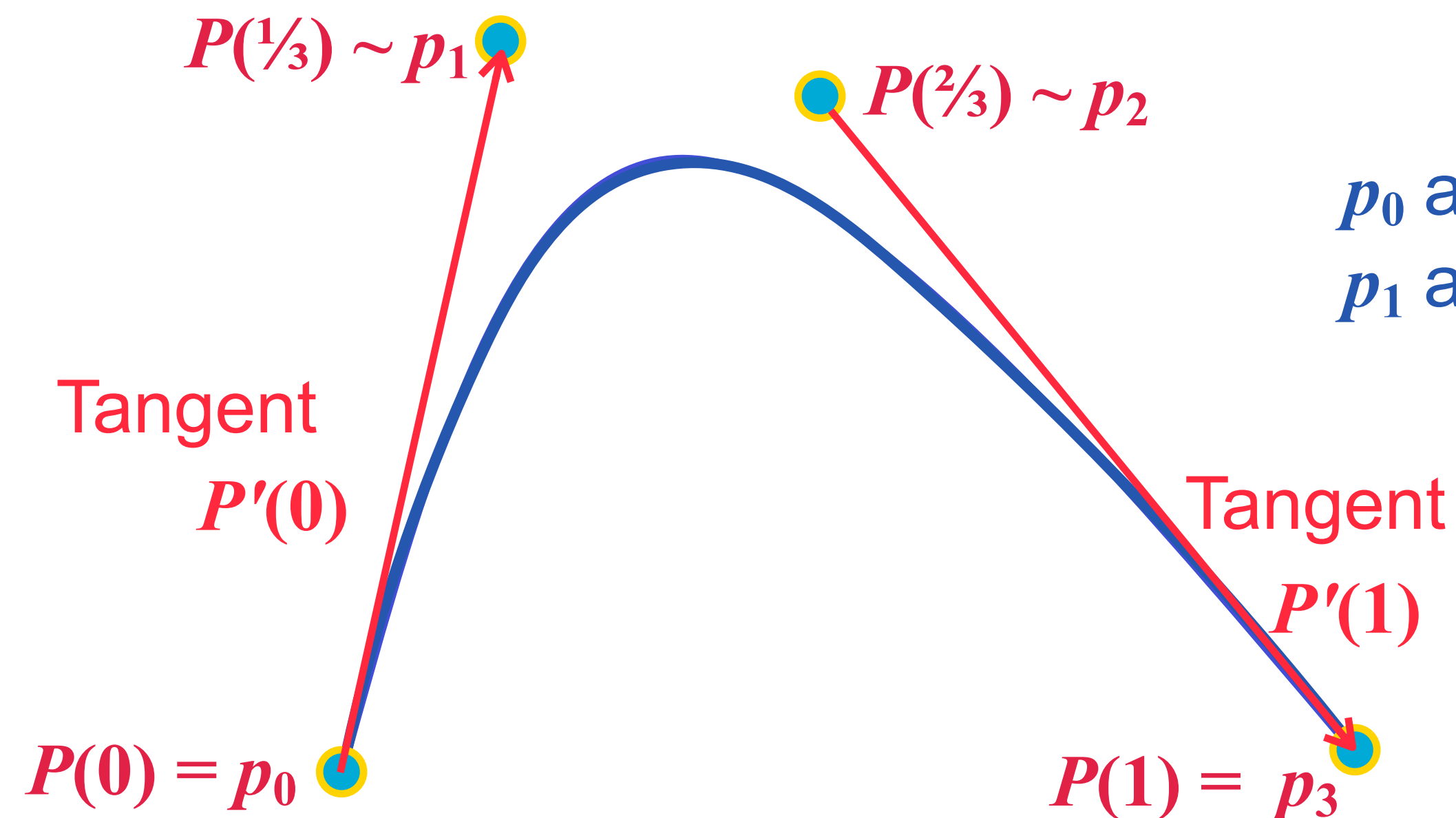


Making Derivatives

An alternate way to indicate derivatives (or tangents):

provide four data-points, but use the intermediate points to form tangents

The curve interpolates only the end-points (but is 'near' the middle two)



p_0 and p_3 are interpolated

p_1 and p_2 are not interpolated (but approximated)

Derivative estimates
using divided difference:

$$P'(0) = \frac{(p_1 - p_0)}{(1/3 - 0)} \quad P'(1) = \frac{(p_3 - p_2)}{(1 - 2/3)}$$

End-point tangents are in the directions
from p_0 to p_1 and p_2 to p_3 , respectively



Hermite's Matrix Form

Recall: $P(t) = c_0 + tc_1 + t^2c_2 + t^3c_3$
 $P'(t) = c_1 + 2tc_2 + 3t^2c_3$

Hence: $P'(0) = 3(p_1 - p_0) = c_1$
 $P'(1) = 3(p_3 - p_2) = c_1 + 2c_2 + 3c_3$

Finally, let: $[t]^T M_B = [b(t)]^T$
to write in terms of basis functions $b_k(t)$

(Hermite Basis) $[b(t)] = \begin{bmatrix} (1-t)^3 \\ 3t(1-t)^2 \\ 3t^2(1-t) \\ t^3 \end{bmatrix}$

Again, $P(t) = [t]^T [c] = [t]^T M_B [p]$

Solve for c to get:

$$M_B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}$$

Recall:

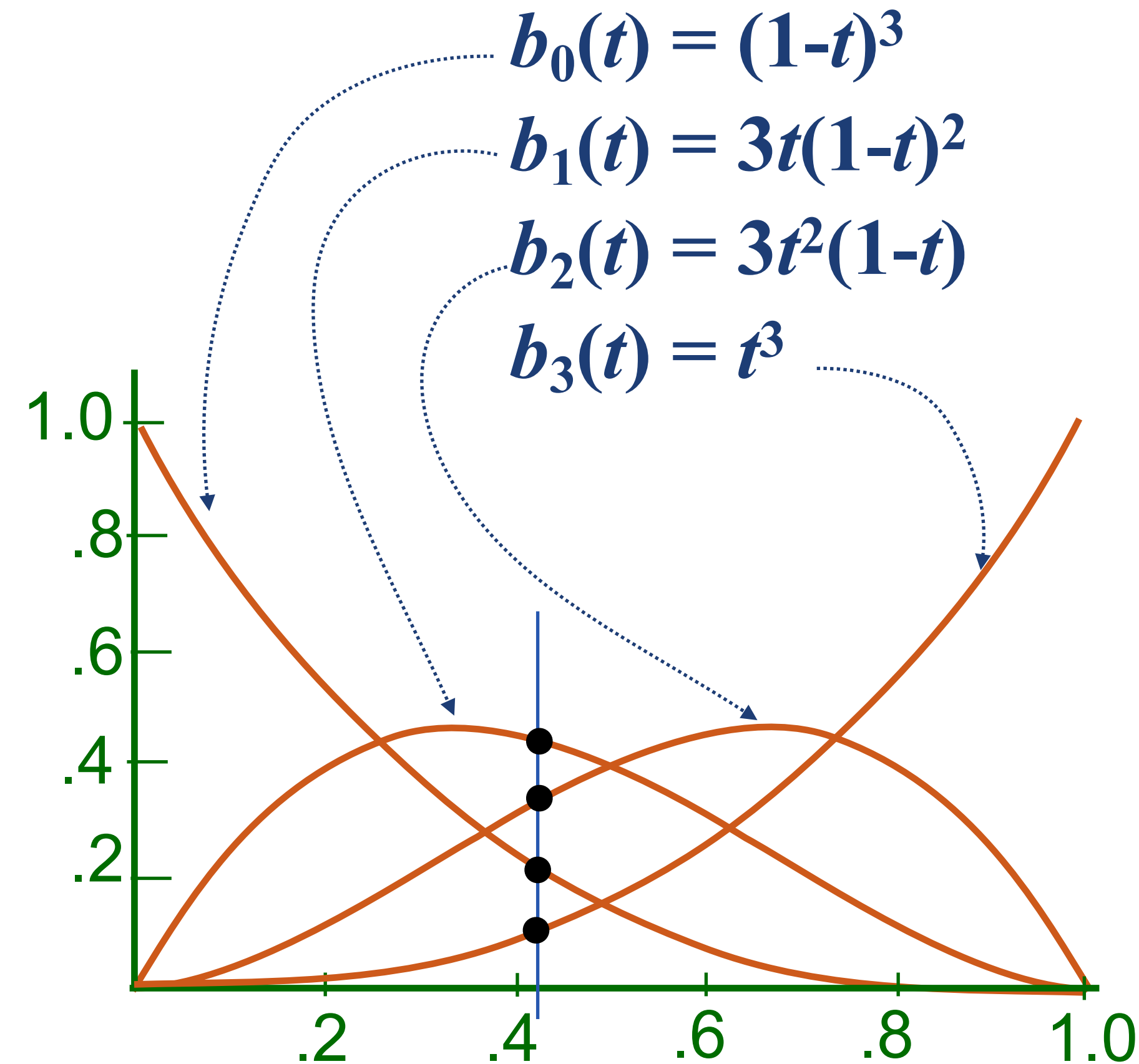
$$[t] = \begin{bmatrix} 1 \\ t \\ t^2 \\ t^3 \end{bmatrix} \quad [p] = \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix}$$



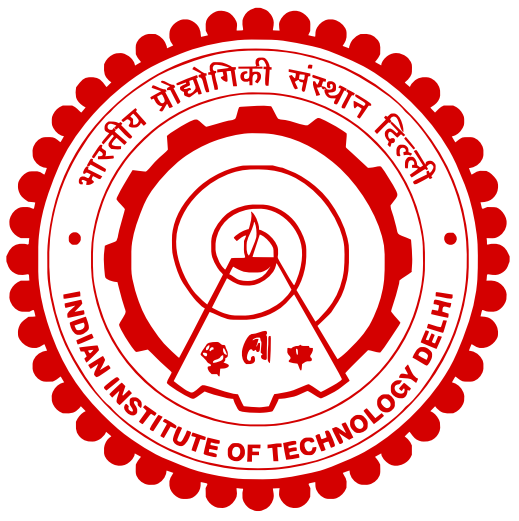
Hermite Blending Functions

Geometric form

$$0 \leq b_k(t) \leq 1 \quad \forall k, t$$



Each control point scales-up the corresponding basis function
These scaled-up values at any given t add up to the final value.



Bernstein Function

- Generalized blending functions of degree d :

$$b_k(t) = \frac{d!}{k!(d-k)!} t^k (1-t)^{d-k} \quad (k = 0..d)$$

- Given degree d , $b_k(t)$ partition unity
 $\Rightarrow 0 \leq b_k(t) \leq 1 \quad \forall k \in 0..d \text{ and } t \in 0..1$
- Sometimes we write $b_{kd}(t)$ to make d explicit

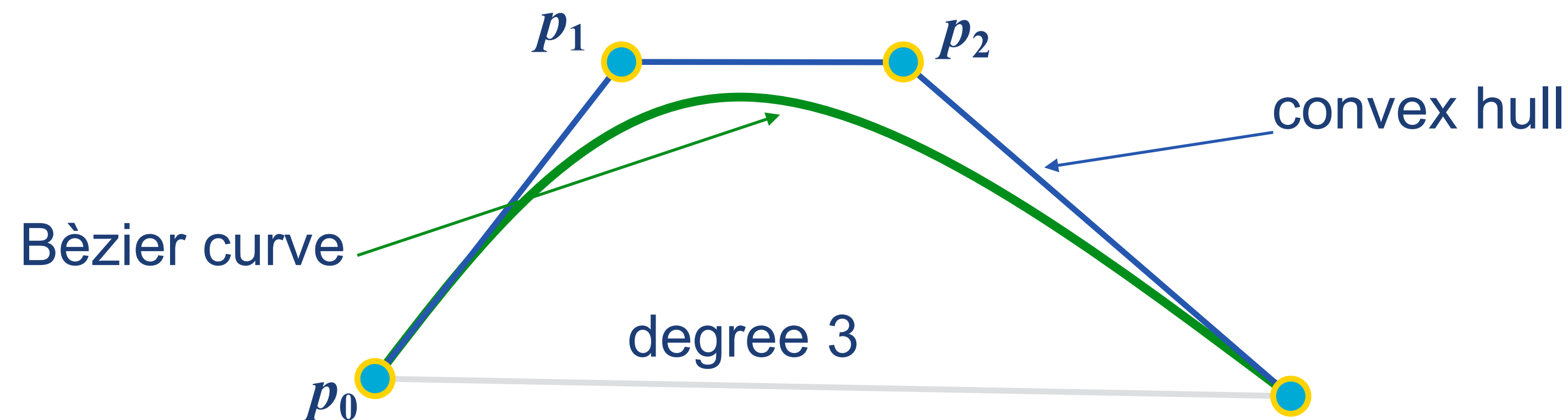


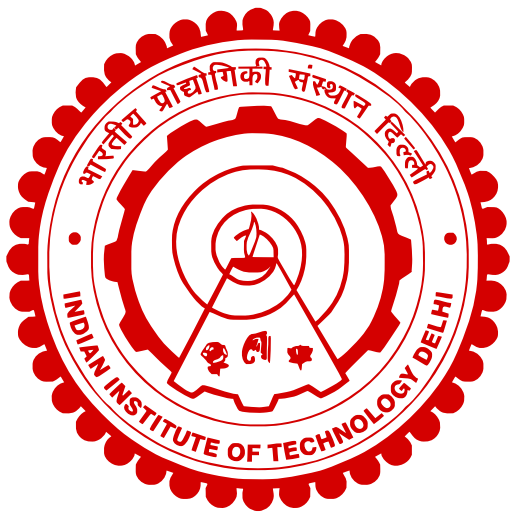
Convex Hull Property

- The curve represented using Bernstein polynomials is also called Bèzier curve

$$P(t) = \sum_{k=0}^d p_k b_{k,d}(t)$$

- Bèzier curves lie in the convex hull of their control points
- Bèzier curves are called non-interpolating



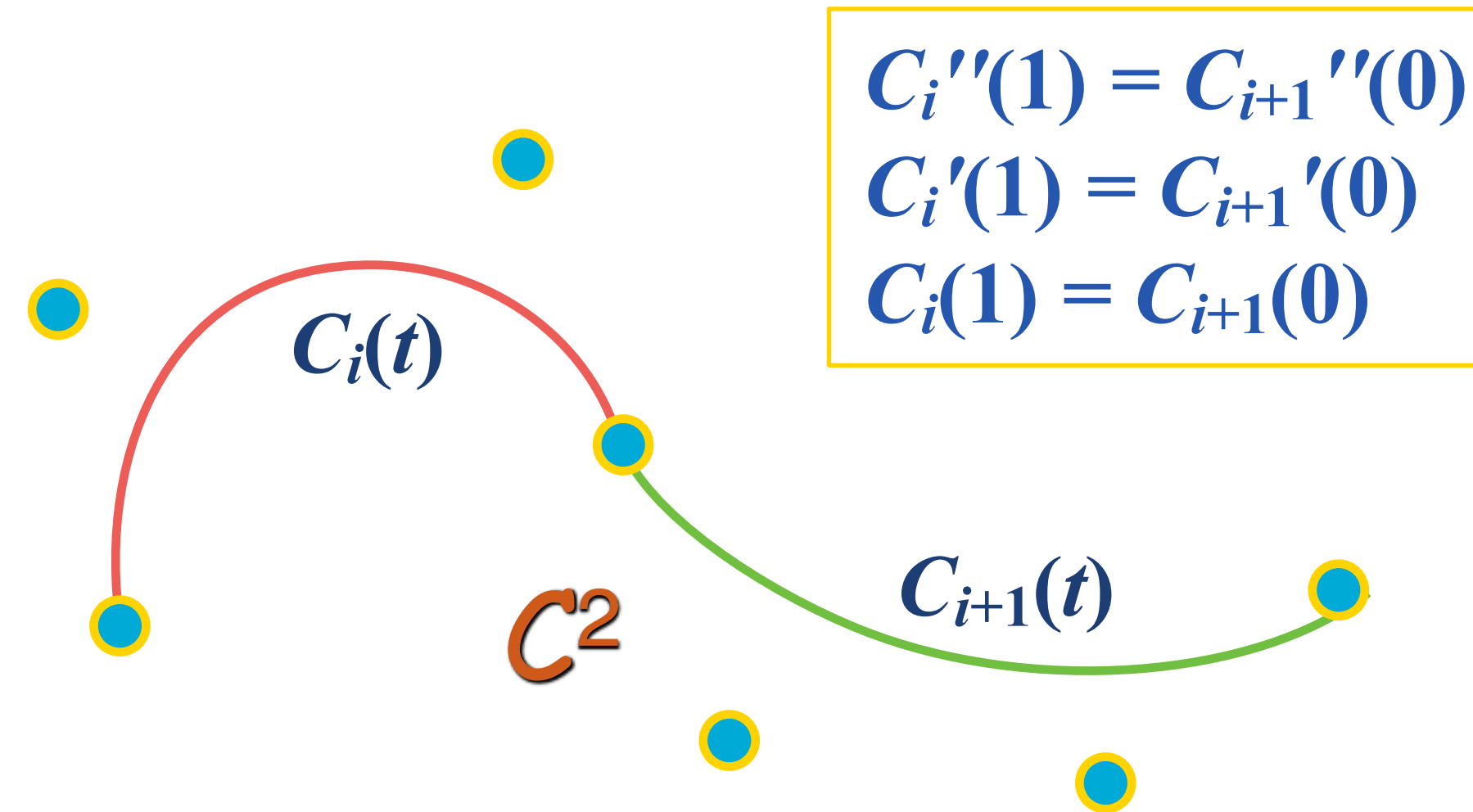


B-Splines

- Piece-together Bezier curves
 - ➡ forming Basis-Splines (*aka* B-splines)
- Form continuity constraints at curves' joints
 - ➡ The highest continuity afforded by cubics is C^2
 - ➡ This means the 1st and 2nd derivatives of two adjoining curves match at each joint
 - ▶ In addition to the joints themselves
 - ▶ Higher degree basis functions can be used for higher continuity
 - ➡ We do not insist on interpolation of control points
- Let C_i be the i^{th} component curve of the B-spline



Continuity Constraints



$\Rightarrow$ Control points of curve $C_i(t)$ cannot be independent of those of $C_{i+1}(t)$ and $C_{i-1}(t)$

Alternatively, we can refactor the basis to pose the entire curve C as:

$$C(u) = \sum_{i=0}^n p_i B_i(u)$$

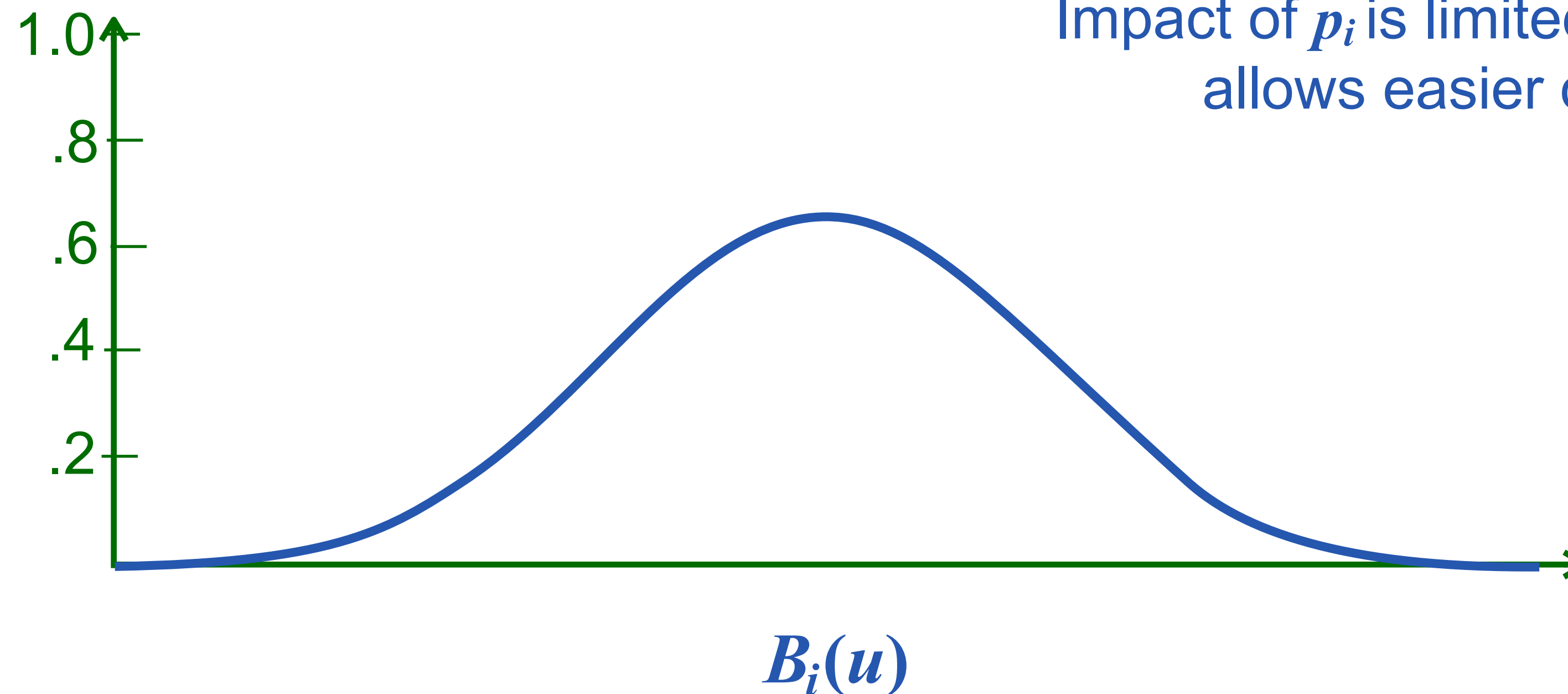
Note the switch to global parameter u .
There is a single u spanning all C_i .

- $n+1$ basis functions B_i not required, depends on degrees of freedom
- Basis functions should be C^2 (for cubic spline) and match at ends



B-Spline Basis

- Create a single C^2 cubic basis function $B(u)$
 - ➔ Want a finite span (where $B(u)$ is non-0)
- $B_i(u)$ are simply translated versions of $B(u)$

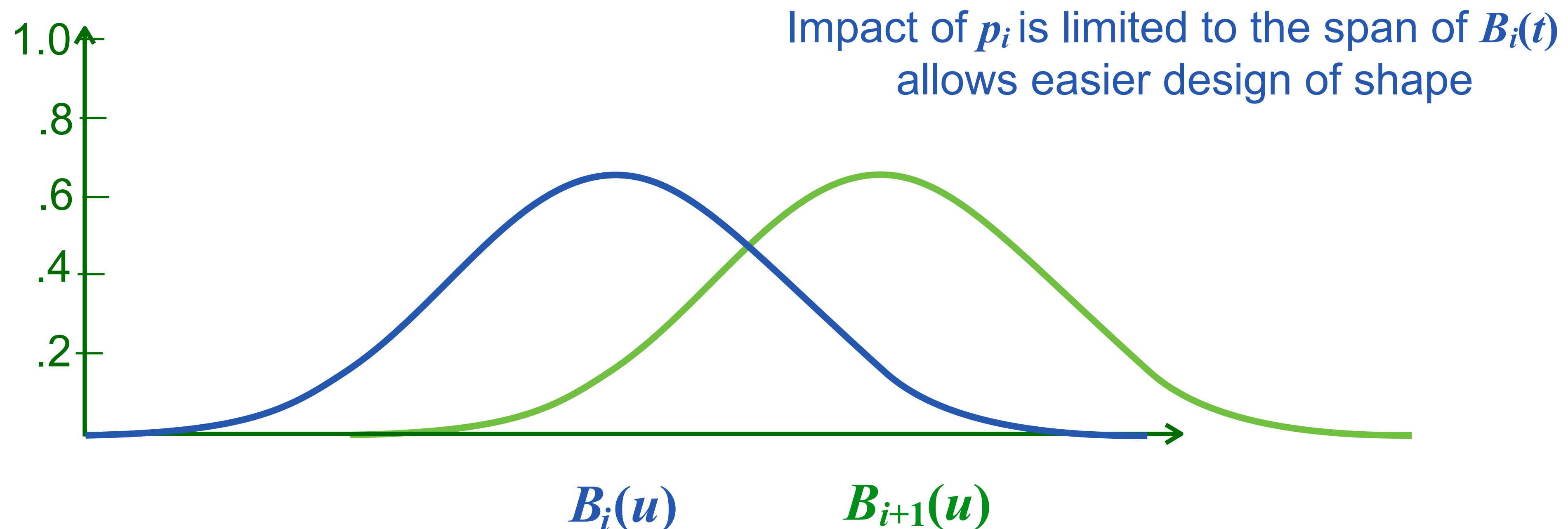


Impact of p_i is limited to the span of $B_i(t)$
allows easier design of shape



B-Spline Basis

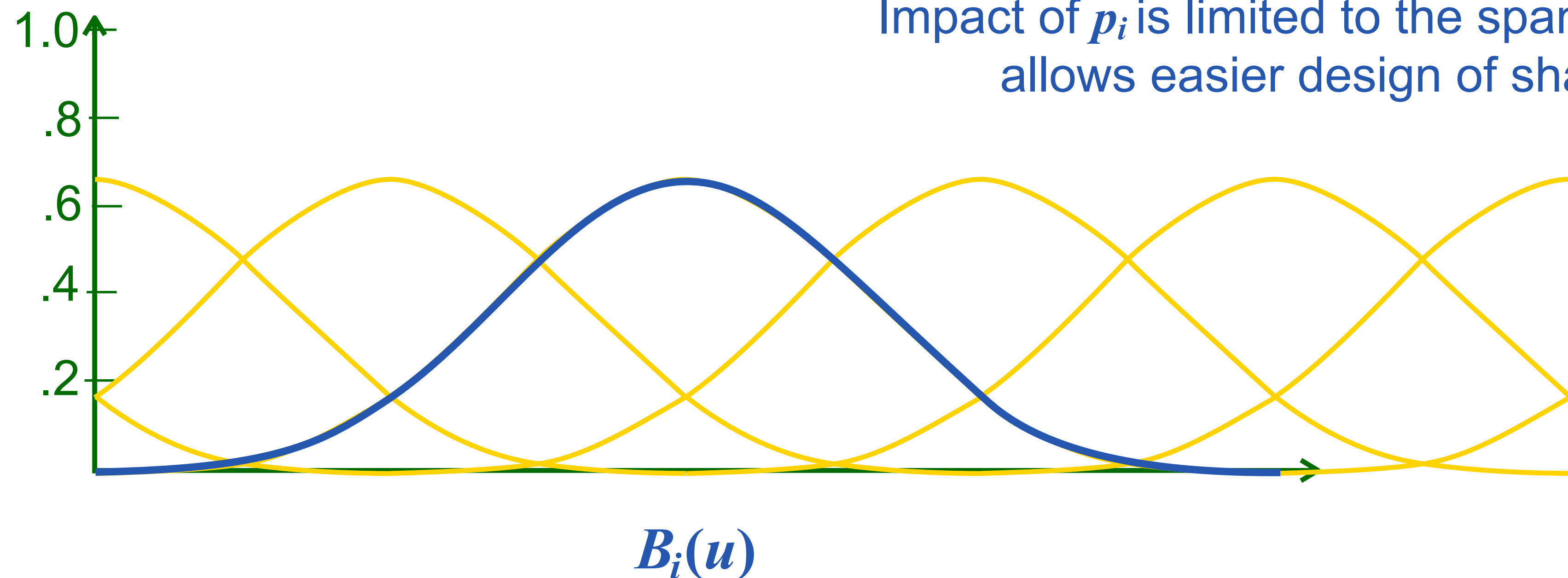
- Create a single C^2 cubic basis function $B(u)$
 - ➔ Want a finite span (where $B(u)$ is non-0)
- $B_i(u)$ are simply translated versions of $B(u)$





B-Spline Basis

- Create a single C^2 cubic basis function $B(u)$
 - ➔ Want a finite span (where $B(u)$ is non-0)
- $B_i(u)$ are simply translated versions of $B(u)$

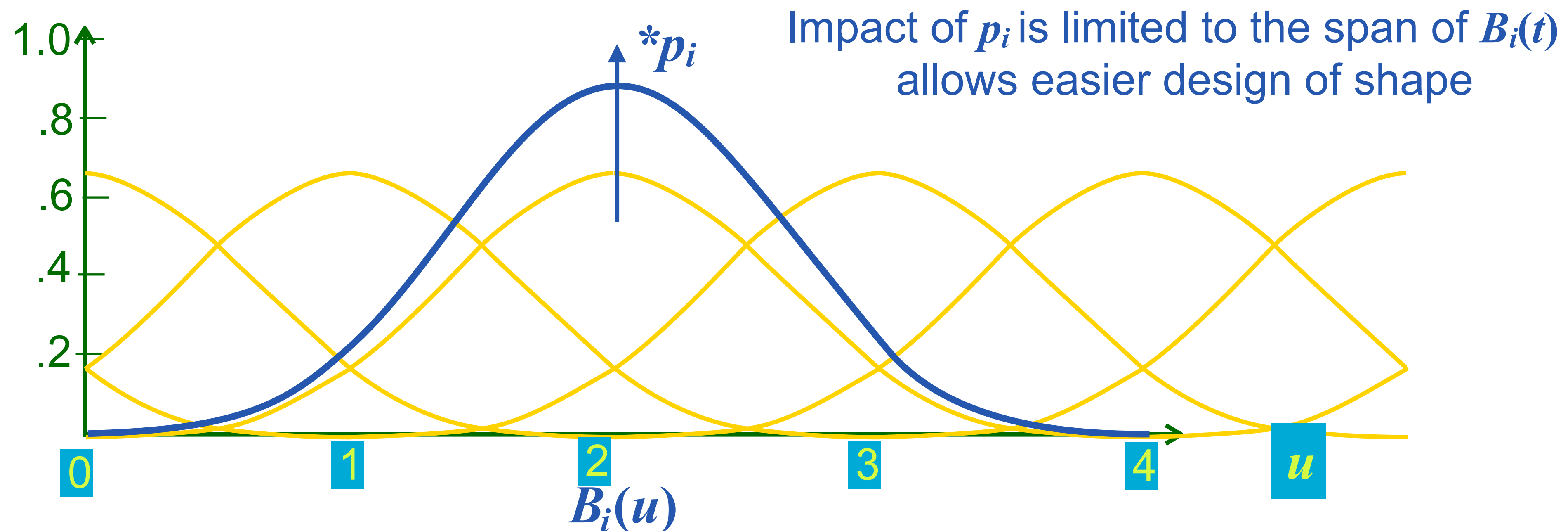


Impact of p_i is limited to the span of $B_i(t)$
allows easier design of shape



B-Spline Basis

- Create a single C^2 cubic basis function $B(u)$
 - ➔ Want a finite span (where $B(u)$ is non-0)
- $B_i(u)$ are simply translated versions of $B(u)$





B-Spline Basis

The basis can be formed as a spline itself with four component curves:

$$P_0(t) = c_{00} + tc_{01} + t^2c_{02} + t^3c_{03}$$

$$P_1(t) = c_{10} + tc_{11} + t^2c_{12} + t^3c_{13}$$

$$P_2(t) = c_{20} + tc_{21} + t^2c_{22} + t^3c_{23}$$

$$P_3(t) = c_{30} + tc_{31} + t^2c_{32} + t^3c_{33}$$

Constraining equations:

C^0 condition:

$$P_0(1) = P_1(0)$$

$$P_1(1) = P_2(0)$$

$$P_2(1) = P_3(0)$$

C^1 condition:

$$P_0'(1) = P_1'(0)$$

$$P_1'(1) = P_2'(0)$$

$$P_2'(1) = P_3'(0)$$

C^2 condition:

$$P_0''(1) = P_1''(0)$$

$$P_1''(1) = P_2''(0)$$

$$P_2''(1) = P_3''(0)$$

Partition of unity:

$$P_0(0) + P_1(0) + P_2(0) + P_3(0) = 1$$

Limit span, and end smoothly:

$$P_0(0) = 0$$

$$P_3(1) = 0$$

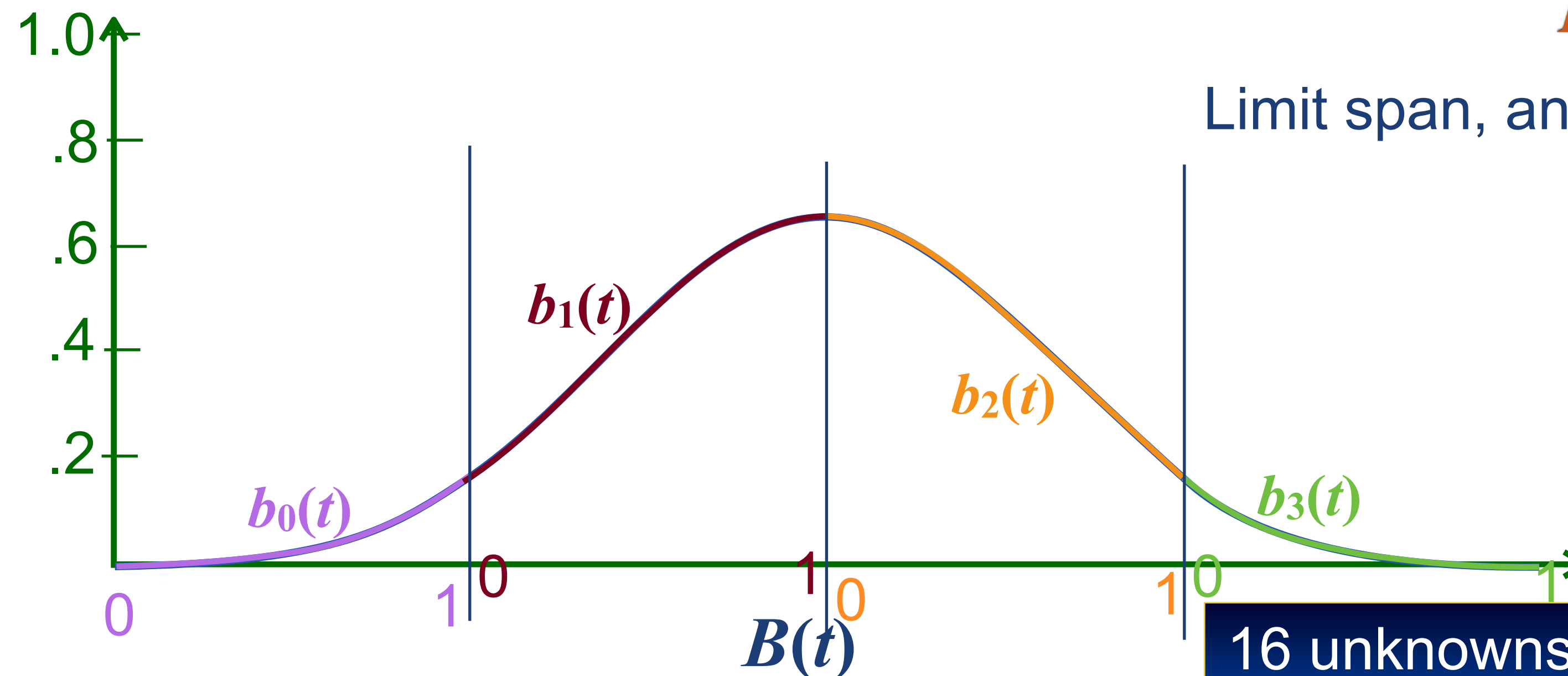
$$P_0'(0) = 0$$

$$P_3'(1) = 0$$

Normal constraints:

$$P_0''(0) = 0$$

$$P_3''(1) = 0$$



16 unknowns (c_{ij}), 16 equations.



B-Spline Basis & Matrix Form

Solution: $P_0(t) = \frac{1}{6} t^3$

$$P_1(t) = \frac{1}{6}(1 + 3t + 3t^2 - 3t^3)$$

$$P_2(t) = \frac{1}{6}(4 - 6t^2 + 3t^3)$$

$$P_3(t) = \frac{1}{6}(1 - 3t + 3t^2 - 3t^3)$$

(B-Spline Basis)

$$[b(t)] = \frac{1}{6} \begin{bmatrix} (1-t)^3 \\ 4 - 6t^2 + 3t^3 \\ 1 + 3t + 3t^2 - 3t^3 \\ t^3 \end{bmatrix}$$

We can again use the matrix form

$$[P(t)]^T = [t]^T M_S$$

where:

$$M_S = \begin{bmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}$$

We usually write the rows in the reverse order, because P_3 at any u is contributed by the left-most basis function, and P_0 by the right most one.

(refer to curve drawing on previous and next slides)



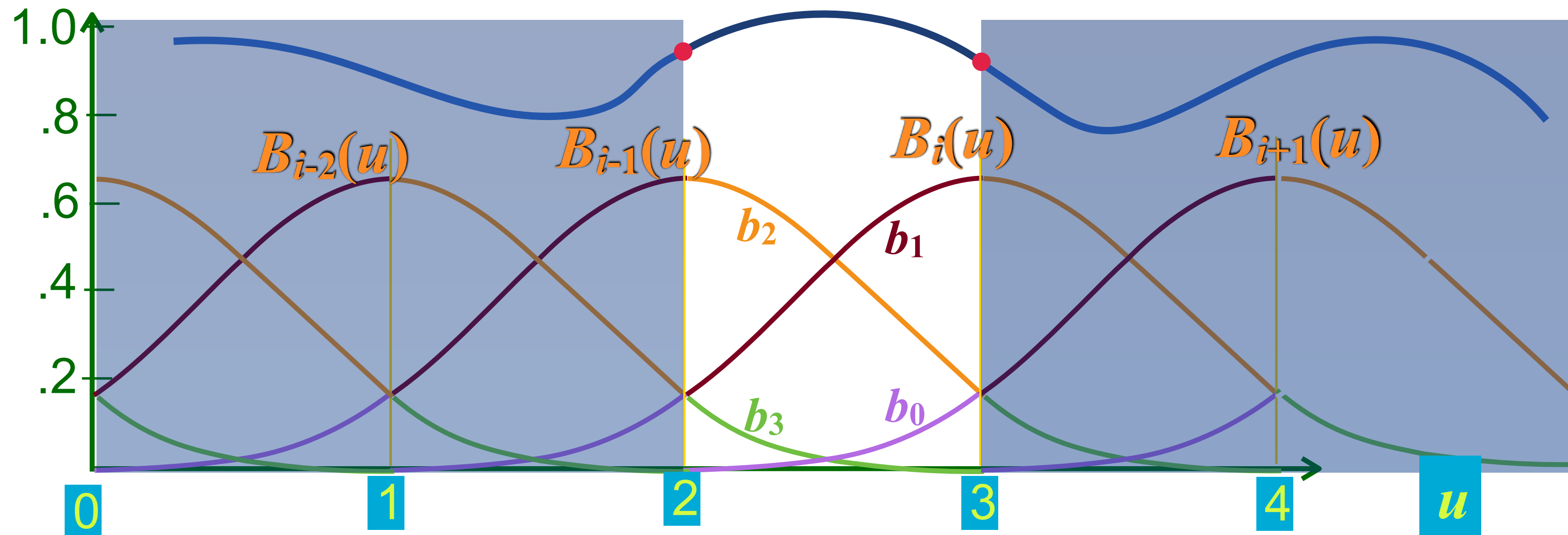
B-Spline & Knots

$$C(u) = \sum_{i=0}^n p_i B_i(u) \quad C_i \text{ for } i-1 \leq u \leq i$$

$$C_i(u) = p_{i-2}B_{i-2}(u) + p_{i-1}B_{i-1}(u) + p_iB_i(u) + p_{i+1}B_{i+1}(u)$$

(Other B are 0 in this range.)

$$C_i(u) = p_{i-2}b_3(t) + p_{i-1}b_2(t) + p_i b_1(t) + p_{i+1}b_0(t), \quad t = u - i + 1$$





B-Spline & Knots

$$C(u) = \sum_{i=0}^n p_i B_i(u)$$

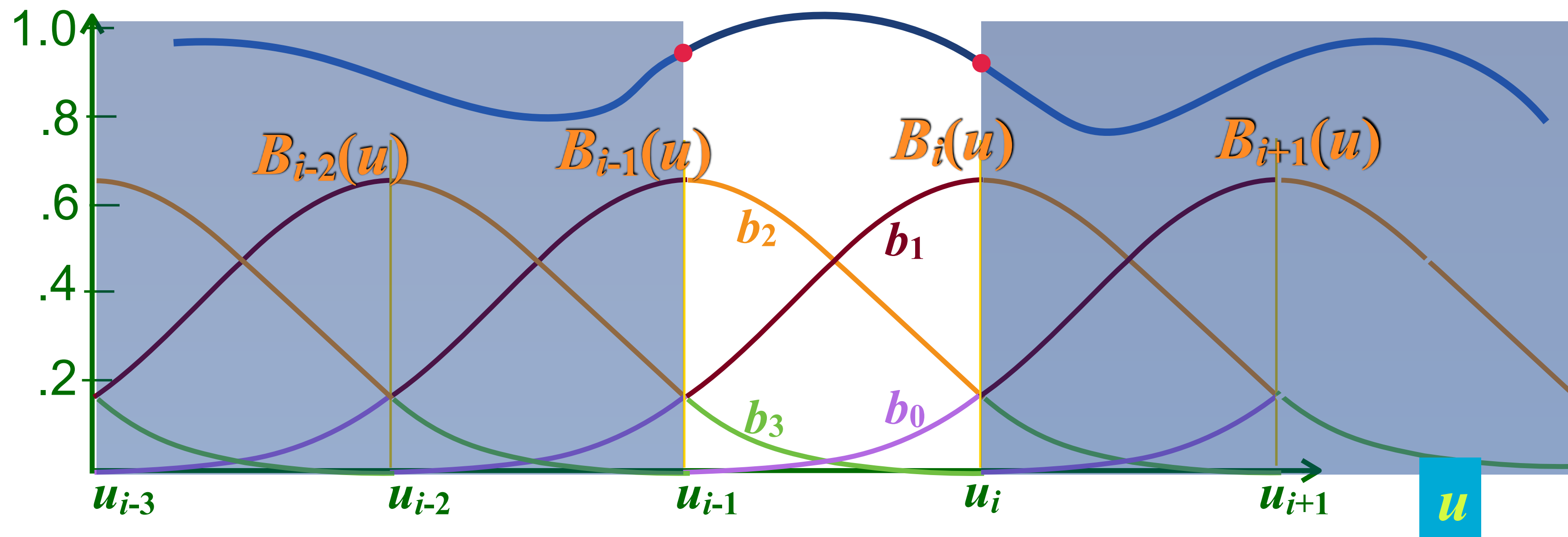
$$C_i \text{ for } u_{i-1} \leq u \leq u_i$$

$$C_i(u) = p_{i-2}B_{i-2}(u) + p_{i-1}B_{i-1}(u) + p_iB_i(u) + p_{i+1}B_{i+1}(u)$$

(Other B are 0 in this range.)

$$C_i(u) = p_{i-2}b_3(t) + p_{i-1}b_2(t) + p_i b_1(t) + p_{i+1}b_0(t), \quad t = u - i + 1$$

$$t = \frac{u - u_{i-1}}{u_i - u_{i-1}}$$





B-Spline & Knots

$$C(u) = \sum_{i=0}^n p_i B_i(u)$$

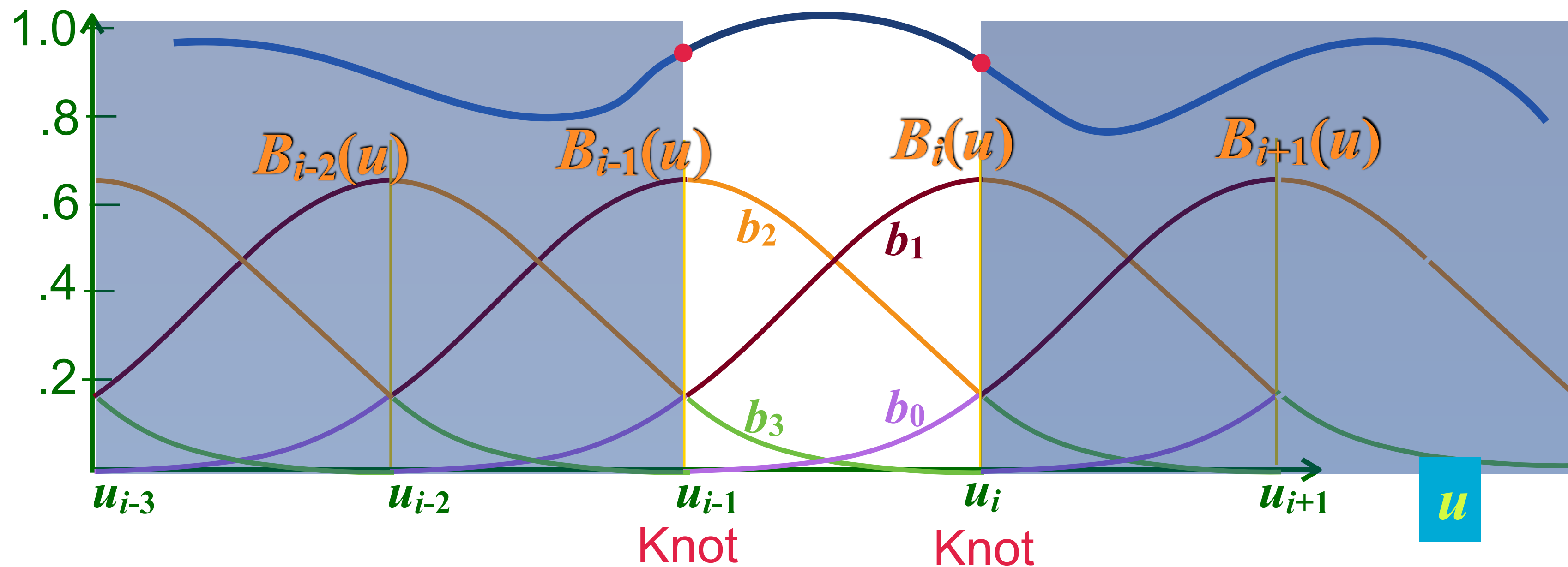
$$C_i \text{ for } u_{i-1} \leq u \leq u_i$$

$$C_i(u) = p_{i-2}B_{i-2}(u) + p_{i-1}B_{i-1}(u) + p_iB_i(u) + p_{i+1}B_{i+1}(u)$$

(Other B are 0 in this range.)

$$C_i(u) = p_{i-2}b_3(t) + p_{i-1}b_2(t) + p_i b_1(t) + p_{i+1}b_0(t), \quad t = u - i + 1$$

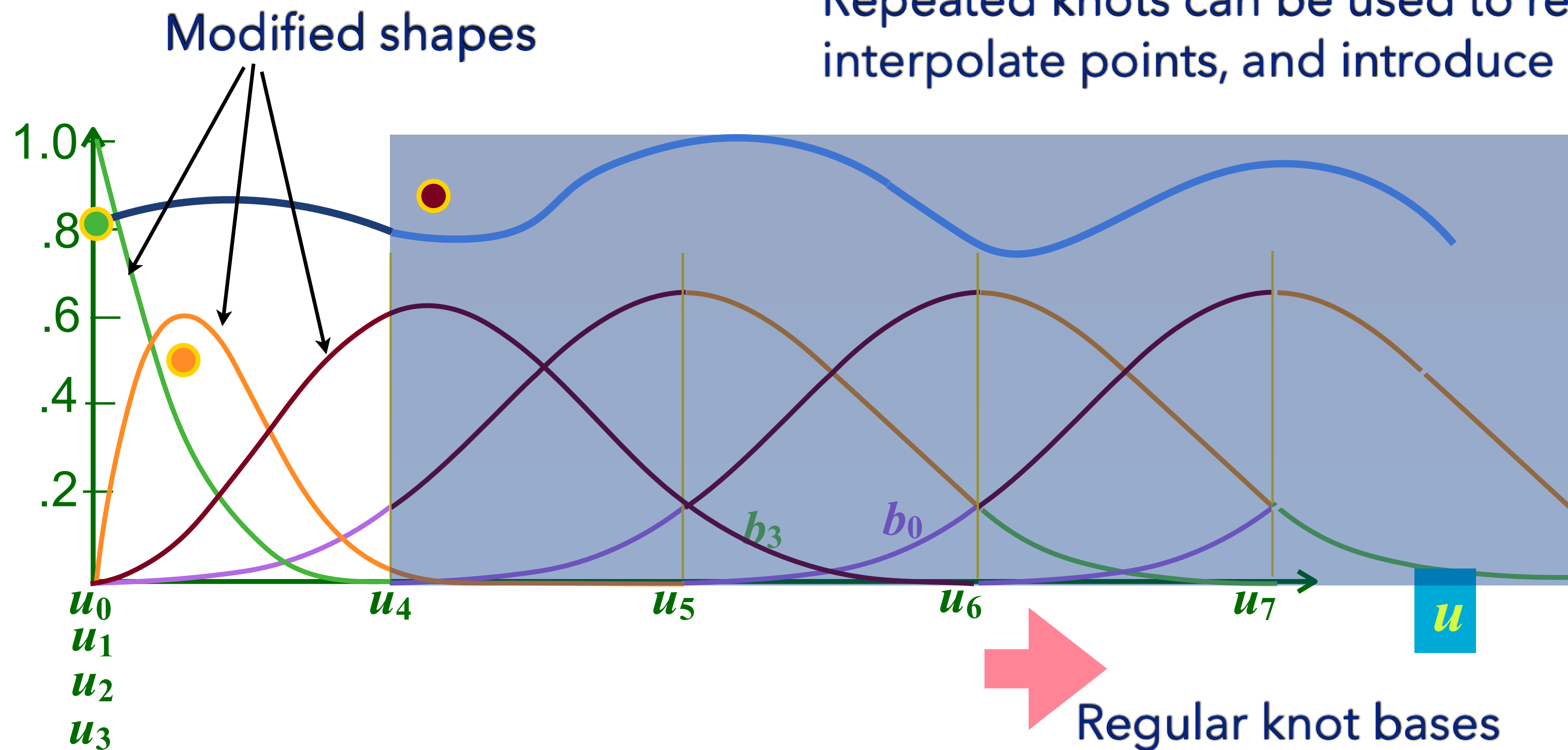
$$t = \frac{u - u_{i-1}}{u_i - u_{i-1}}$$



Repeated Knots

$$C_0(u) = p_{-2}b_3(t) + p_{-1}b_2(t) + p_0b_1(t) + p_1b_0(t)$$

Repeated knots can be used to reduce continuity, interpolate points, and introduce corners.



General B-spline Basis Function



$$B_i^d(u) = \frac{u - u_i}{u_{i+d} - u_i} B_i^{d-1}(u) + \frac{u_{i+d+1} - u}{u_{i+d+1} - u_{i+1}} B_{i+1}^{d-1}(u)$$

0 denominator $\Rightarrow$ zero the term

$$B_i^0(u) = \begin{cases} 1 & \text{if } u_i \leq u < u_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

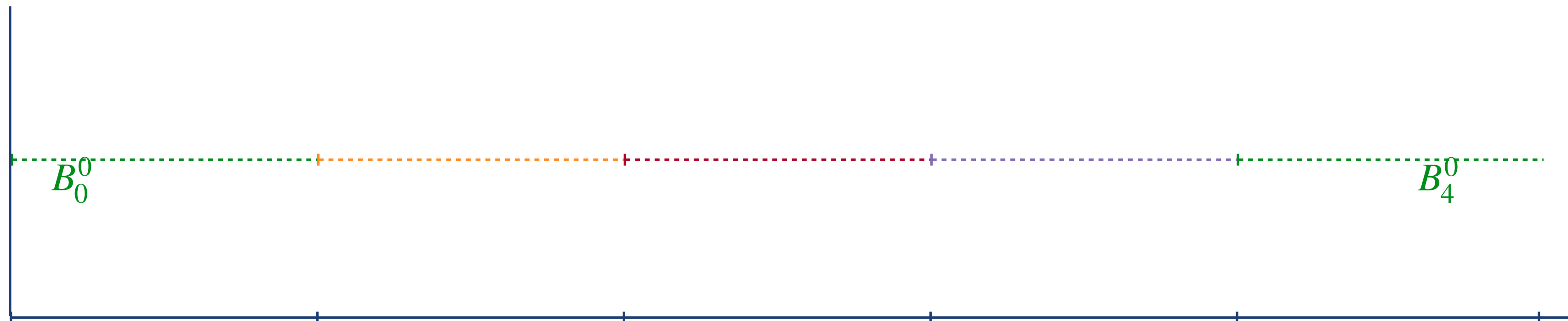
General B-spline Basis Function



$$B_i^d(u) = \frac{u - u_i}{u_{i+d} - u_i} B_i^{d-1}(u) + \frac{u_{i+d+1} - u}{u_{i+d+1} - u_{i+1}} B_{i+1}^{d-1}(u)$$

0 denominator $\Rightarrow$ zero the term

$$B_i^0(u) = \begin{cases} 1 & \text{if } u_i \leq u < u_{i+1} \\ 0 & \text{otherwise} \end{cases}$$



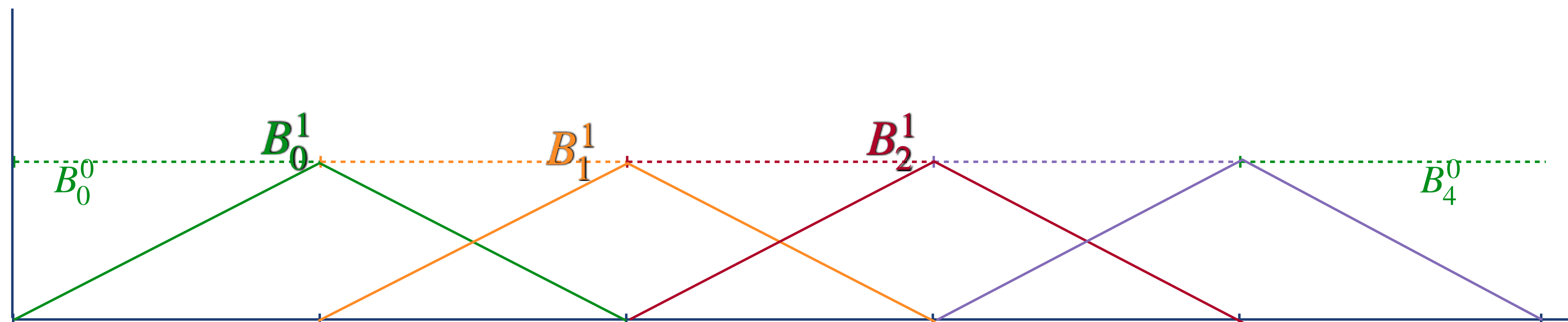


General B-spline Basis Function

$$B_i^d(u) = \frac{u - u_i}{u_{i+d} - u_i} B_i^{d-1}(u) + \frac{u_{i+d+1} - u}{u_{i+d+1} - u_{i+1}} B_{i+1}^{d-1}(u)$$

0 denominator $\Rightarrow$ zero the term

$$B_i^0(u) = \begin{cases} 1 & \text{if } u_i \leq u < u_{i+1} \\ 0 & \text{otherwise} \end{cases}$$



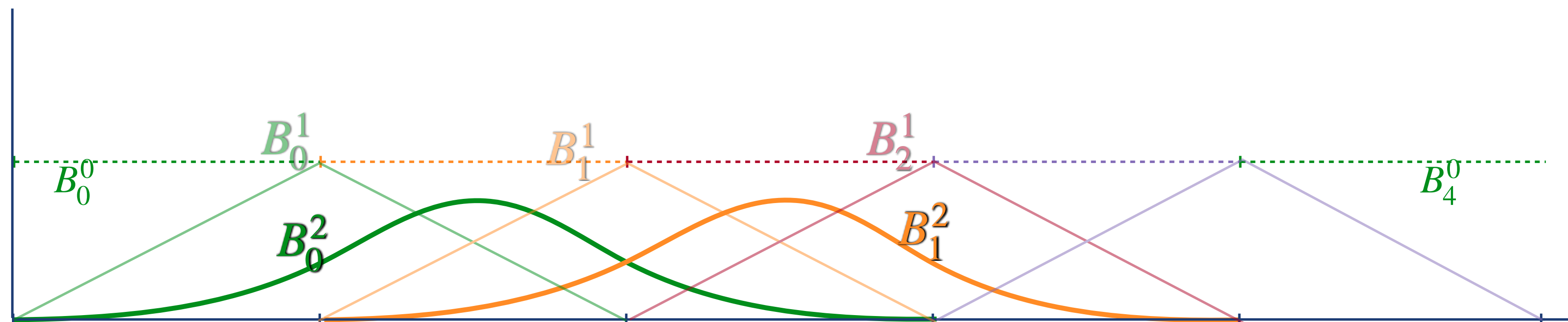


General B-spline Basis Function

$$B_i^d(u) = \frac{u - u_i}{u_{i+d} - u_i} B_i^{d-1}(u) + \frac{u_{i+d+1} - u}{u_{i+d+1} - u_{i+1}} B_{i+1}^{d-1}(u)$$

0 denominator $\Rightarrow$ zero the term

$$B_i^0(u) = \begin{cases} 1 & \text{if } u_i \leq u < u_{i+1} \\ 0 & \text{otherwise} \end{cases}$$





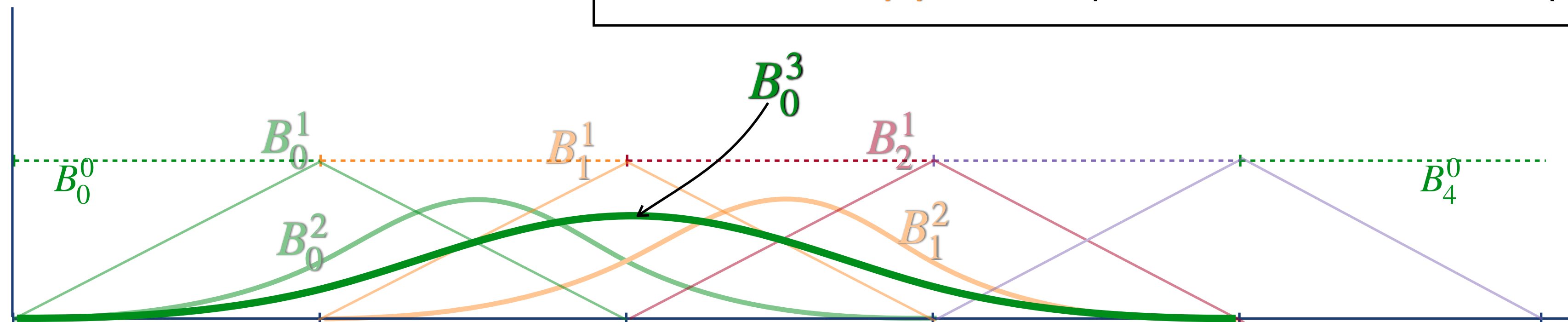
General B-spline Basis Function

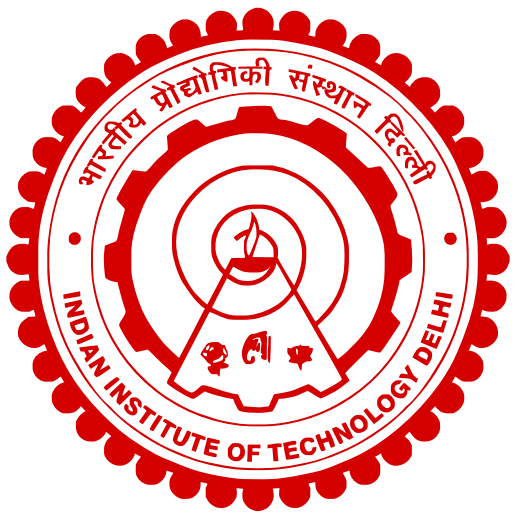
$$B_i^d(u) = \frac{u - u_i}{u_{i+d} - u_i} B_i^{d-1}(u) + \frac{u_{i+d+1} - u}{u_{i+d+1} - u_{i+1}} B_{i+1}^{d-1}(u)$$

0 denominator $\Rightarrow$ zero the term

$$B_i^0(u) = \begin{cases} 1 & \text{if } u_i \leq u < u_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

- **Affine invariance:** Transform surface by transforming control points
- **Partition of unity:** Basis functions sum to 1
- Surface lies inside **convex hull** of control points
- Limited **support:** Impact of each control point is local





General Spline based design

- Choose a basis
 - ➔ Interpolating or approximating
- Choose degree
- A knot vector should also be given
 - ➔ Extremal intervals do not have curve components
 - As they do not have p_{i-2} , p_{i-1} , or p_{i+1}
 - ➔ Uniform or arc-length parameterization is common
 - ➔ Multiple knots can be at the same parameter value
 - Reduces smoothness (continuity)

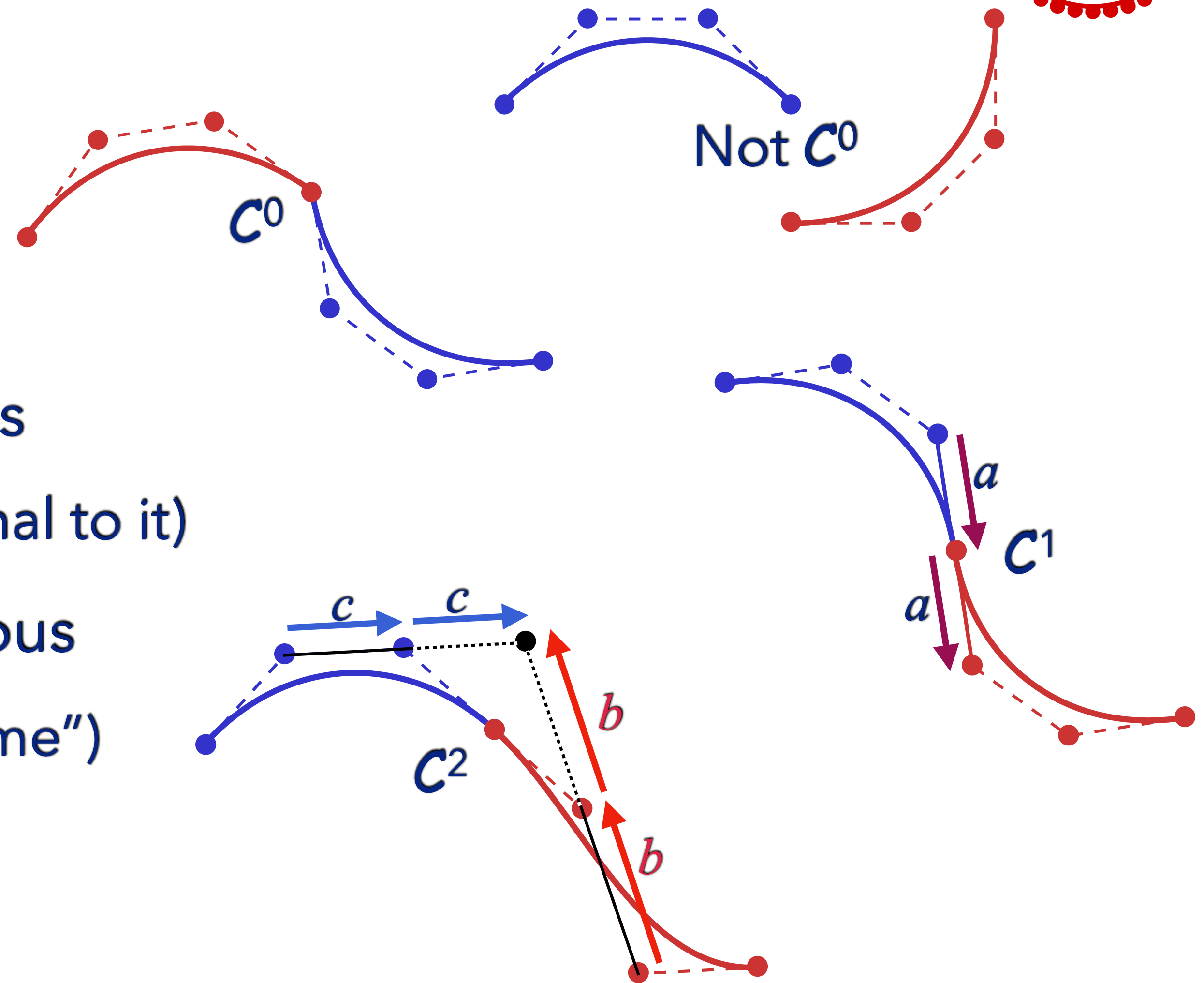
Rational Splines have control points in projective space:

$$x(u) = \frac{\sum w_i x_i B_i(u)}{\sum w_i B_i(u)}$$

Continuity Across Bezier Segments

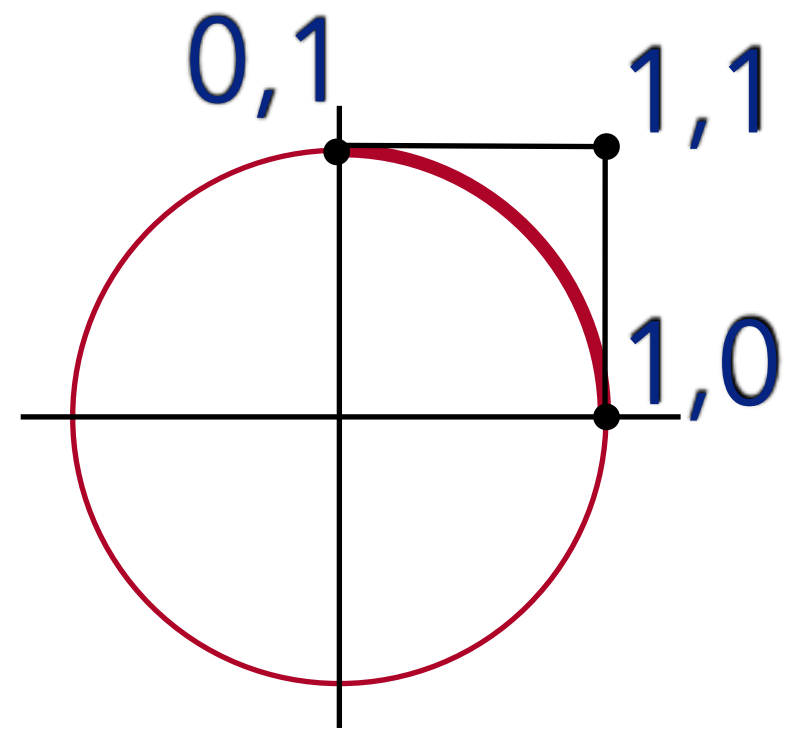


- C^0 continuity: $P(t)$ is continuous in t
 - ⇒ Endpoints match
- C^1 continuity: dP/dt is also continuous
 - ⇒ End edges match (Tangent proportional to it)
- C^2 continuity: d^2P/dt^2 is also continuous
 - ⇒ Extended mid-segments meet ("A-frame")





Circle as Bezier Curve

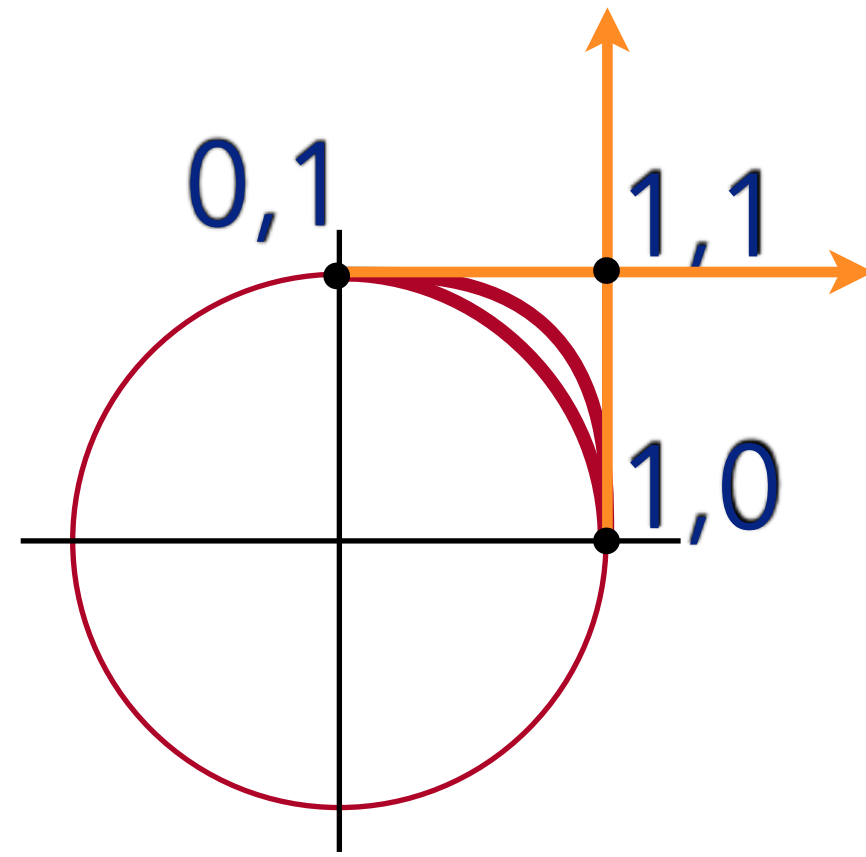


$$b_{d,k}(t) = \frac{d!}{k!(d-k)!} t^k (1-t)^{d-k}$$

$$\frac{db}{dt} = d (b_{d-1,k-1}(t) - b_{d-1,k}(t))$$



Circle as Bezier Curve

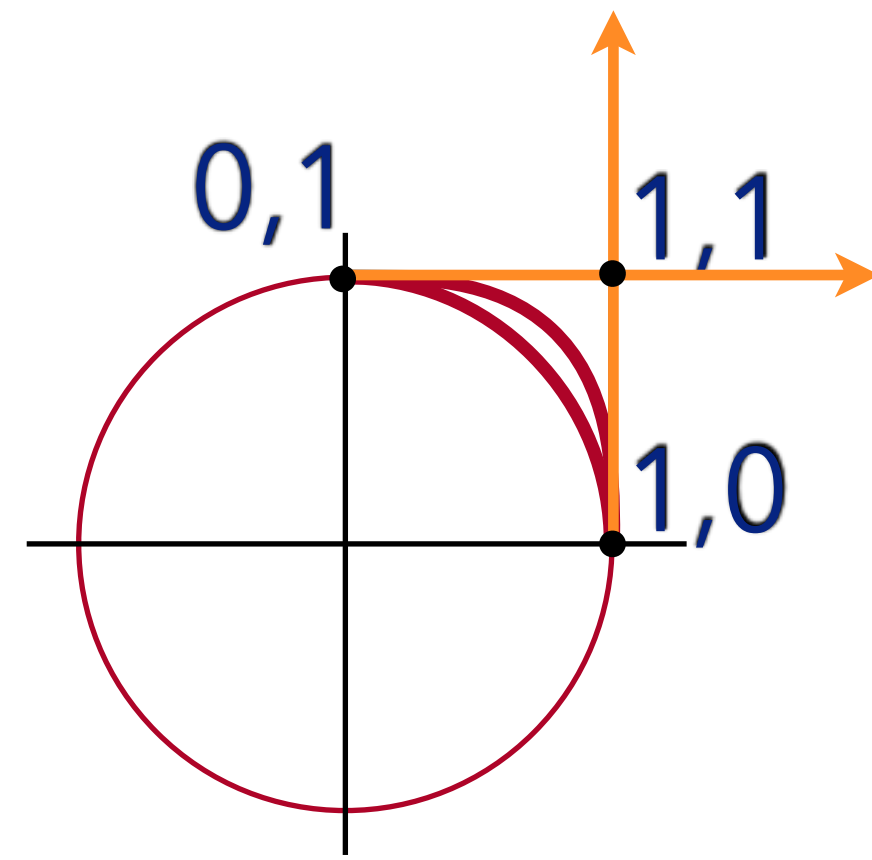


$$b_{d,k}(t) = \frac{d!}{k!(d-k)!} t^k (1-t)^{d-k}$$

$$\frac{db}{dt} = d (b_{d-1,k-1}(t) - b_{d-1,k}(t))$$

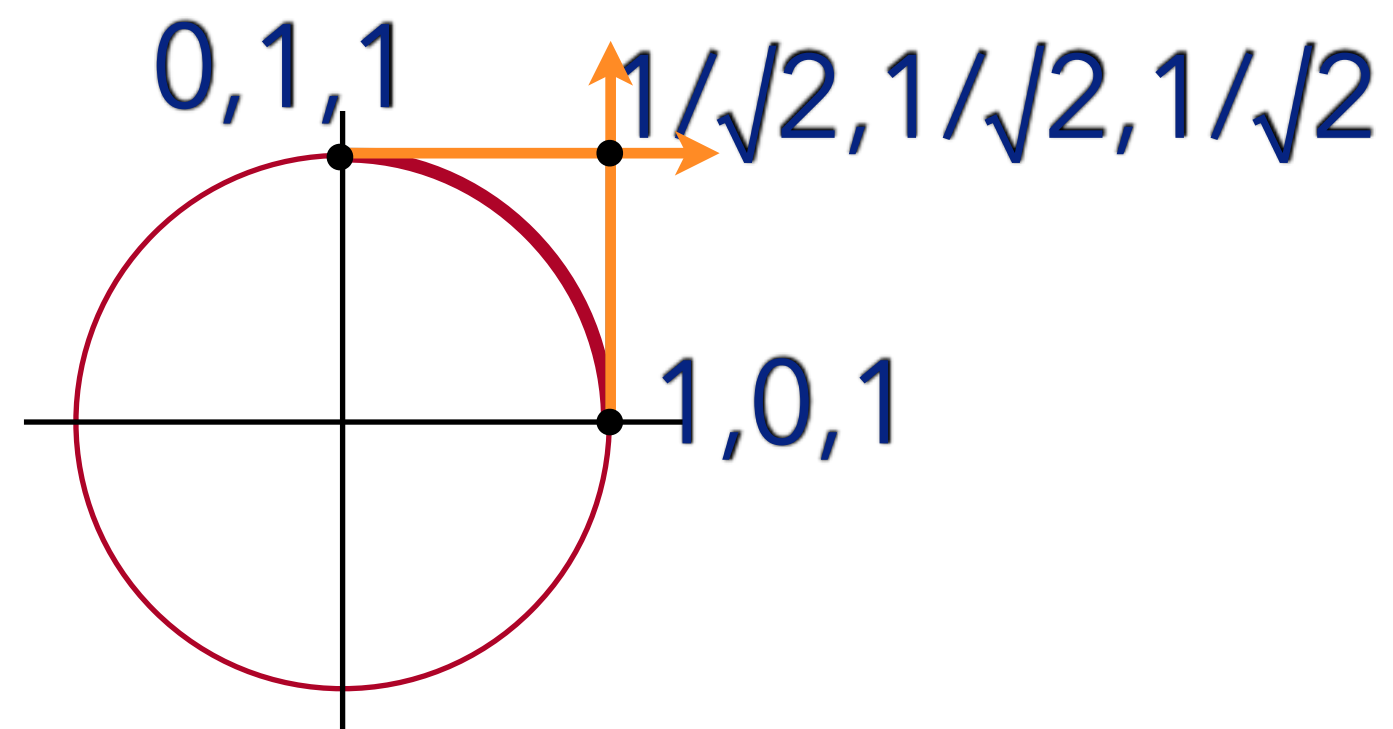


Circle as Bezier Curve



$$b_{d,k}(t) = \frac{d!}{k!(d-k)!} t^k (1-t)^{d-k}$$

$$\frac{db}{dt} = d (b_{d-1,k-1}(t) - b_{d-1,k}(t))$$

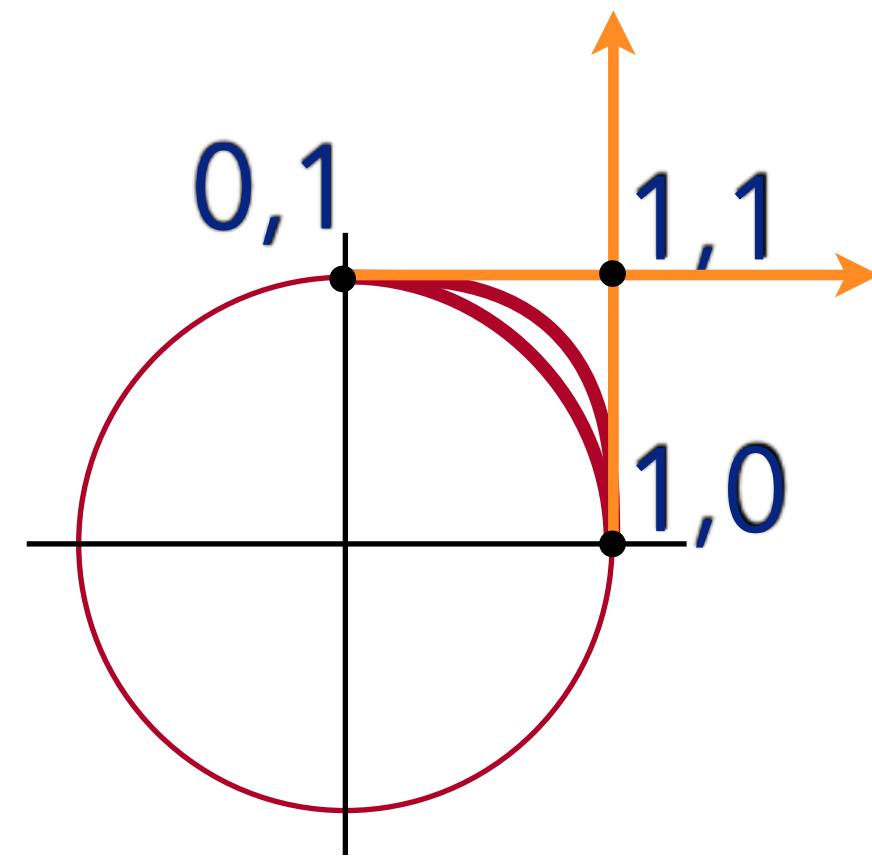


Circle is a rational curve

$$x(u) = \frac{\sum w_i x_i B_i(u)}{\sum w_i B_i(u)}, y(u) = \dots$$

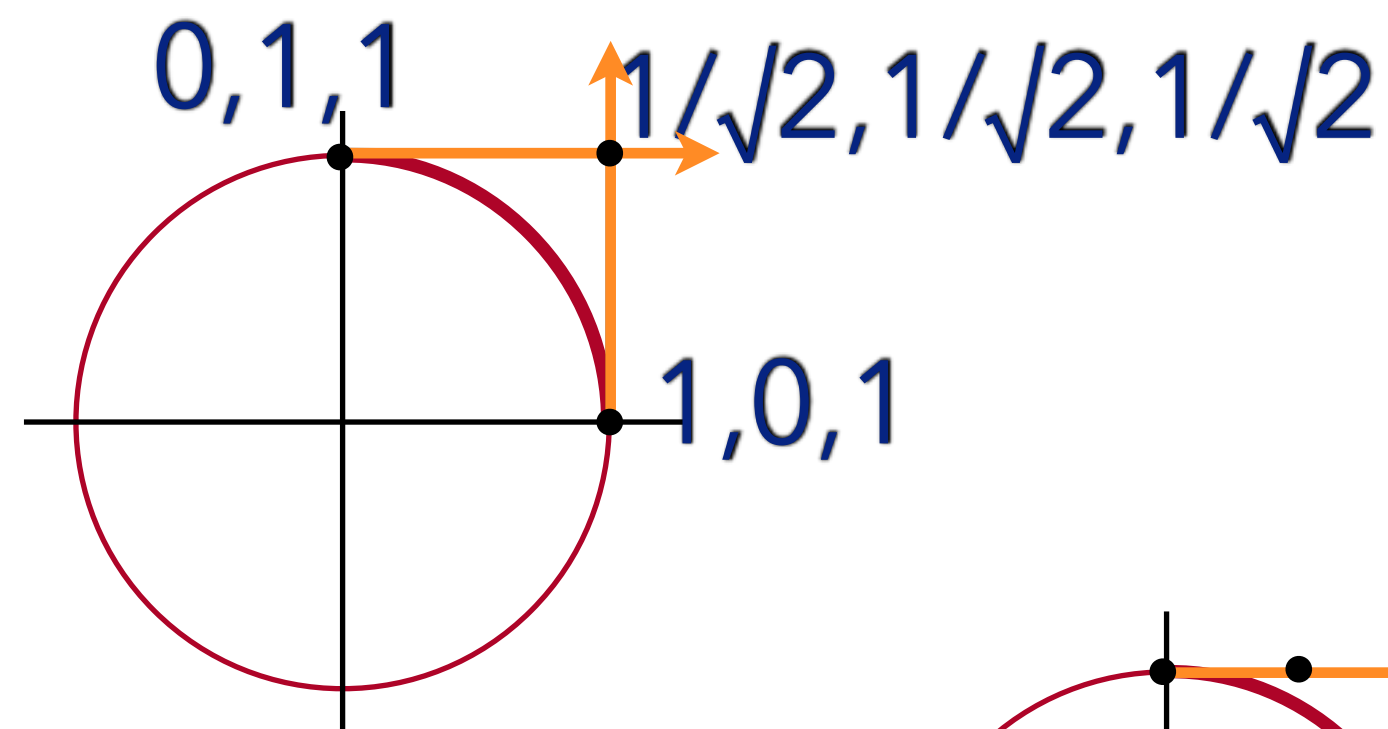


Circle as Bezier Curve



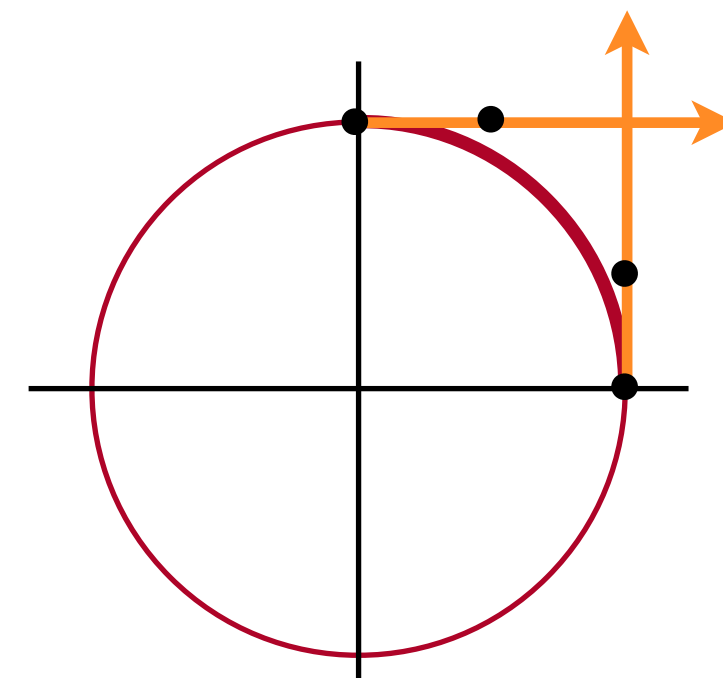
$$b_{d,k}(t) = \frac{d!}{k!(d-k)!} t^k (1-t)^{d-k}$$

$$\frac{db}{dt} = d (b_{d-1,k-1}(t) - b_{d-1,k}(t))$$



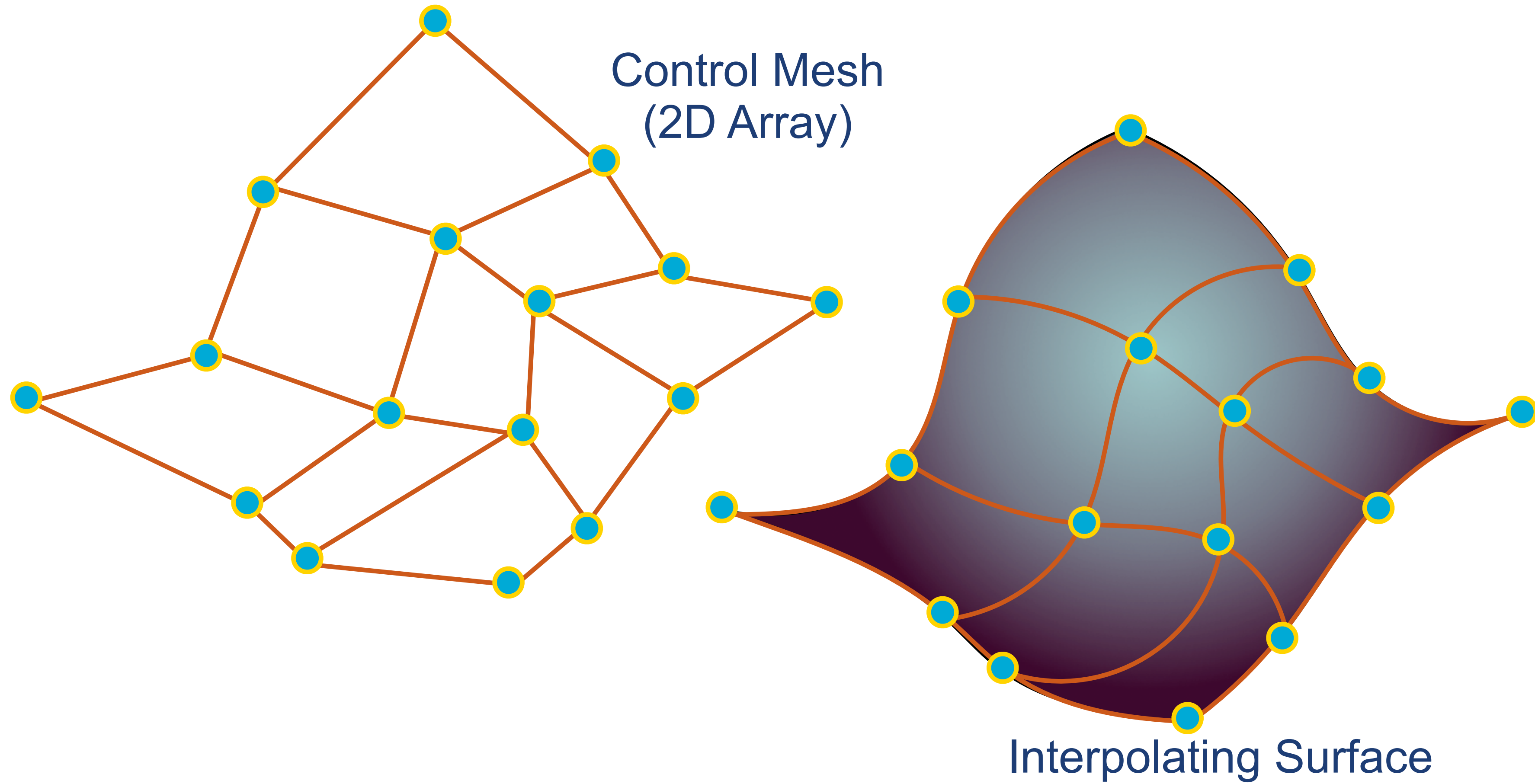
Circle is a rational curve

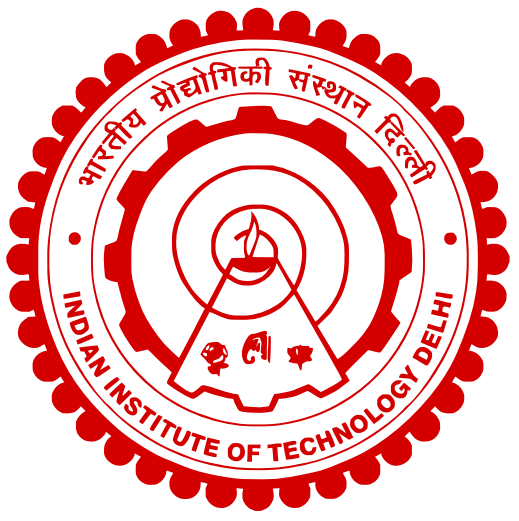
$$x(u) = \frac{\sum w_i x_i B_i(u)}{\sum w_i B_i(u)}, y(u) = \dots$$



Can be approximated well with cubic Bezier

Surfaces





Surface Formation

- A surface consists of an infinite number of curves
 - ➔ Each curve is determined by its own control points
 - ➔ But surface comprises ∞ curves, means ∞ sequences of control points
- One solution is to have each control point itself lie on a curve
 - ➔ Thus p_i are really $p_i(v)$
 - Each p_i is a point on a curve parameterized by v
 - ➔ The control points of this ' v^{th} ' curve define its shape

$$P(u, v) = \sum_{i=0}^3 (p_i(v)B_i(u)) \quad (\text{Tensor Product})$$



B-Spline Surface

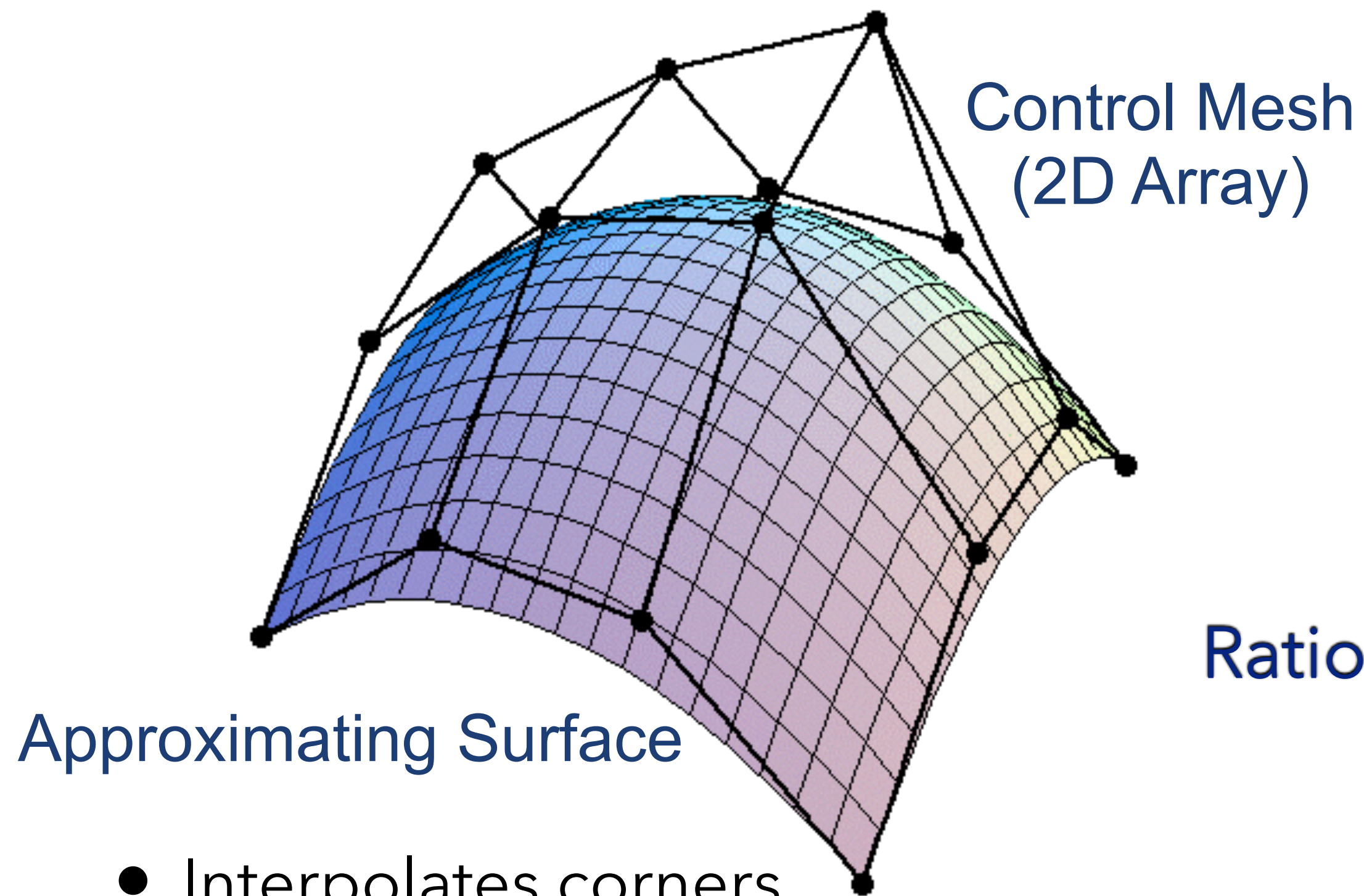
$$P(u, v) = \sum_{i=0}^m (p_i(v) B_i(u)) \quad , \text{ where } p_i(v) = \sum_{j=0}^n (p_{ij} B_j(v))$$

This can be re-written as:

$$P(u, v) = \sum_{i=0}^m \sum_{j=0}^n (p_{ij} B_{ij}(u, v))$$

- p_{ij} form an $m \times n$ mesh of control points
 - ➔ compared to control polygons for curves
- $B_{ij}(u, v)$ is formed by a tensor product of $B_i(u)$ and $B_j(v)$
 - ➔ i.e., expand this equation

Cubic Bezier Surface



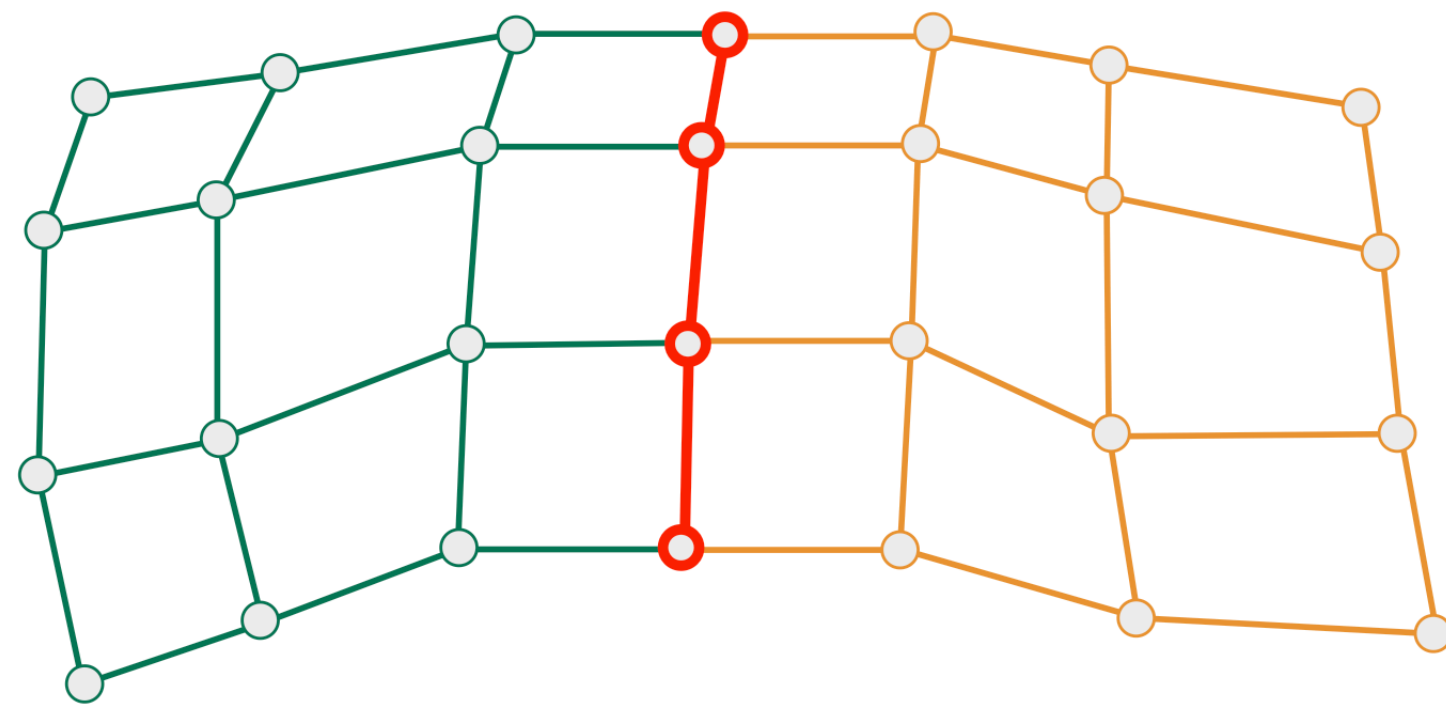
$$P(u, v) = \sum_{i=0}^m \sum_{j=0}^n P_{ij} b_i^m(u) b_j^n(v)$$

Rational Bezier surfaces have projective invariance

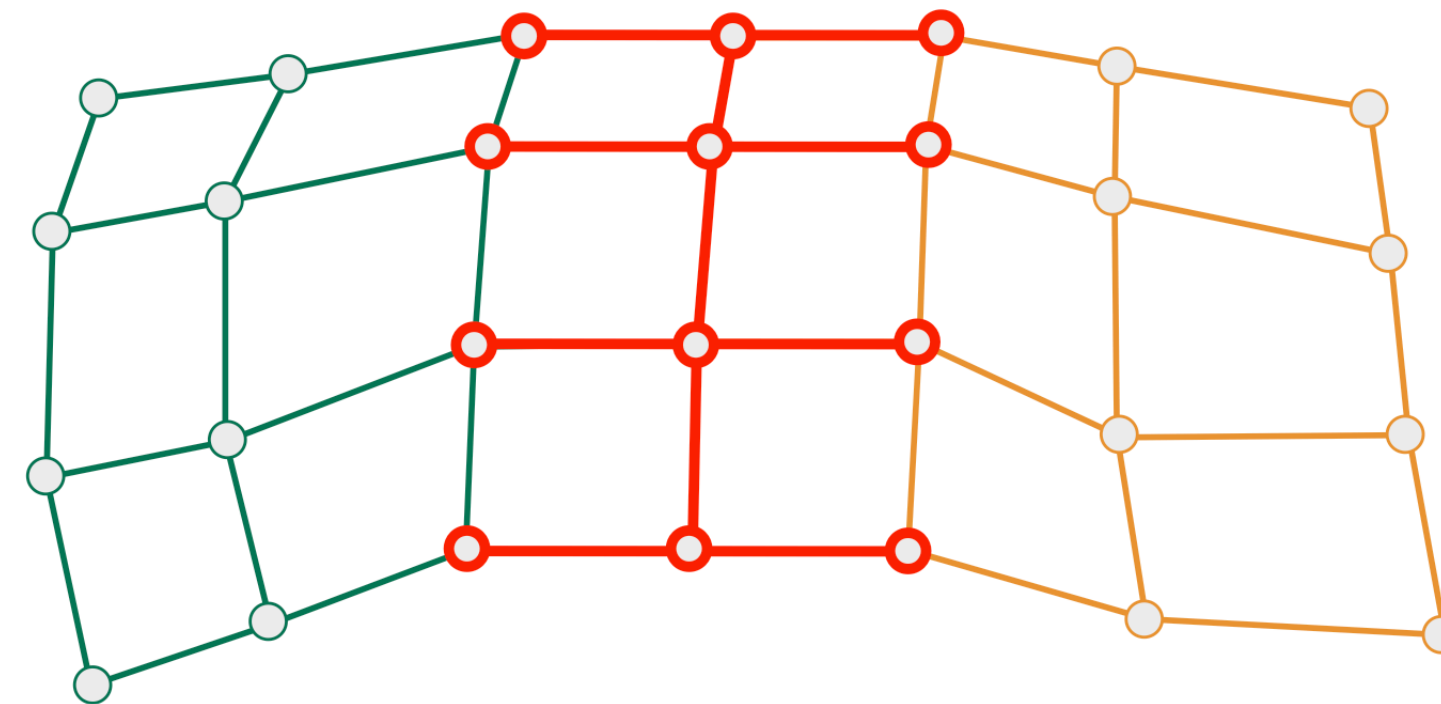
- Interpolates corners
- Tangents in u and v directions, respectively, are parallel to the control edges
- **Affine invariance:** Surface follows affine transformation of control points (polynomial Bezier)
- Surface lies inside **convex hull** of control points
- Patches can be smoothly combined: within one patch (and neighbors), support is non-local

Continuity

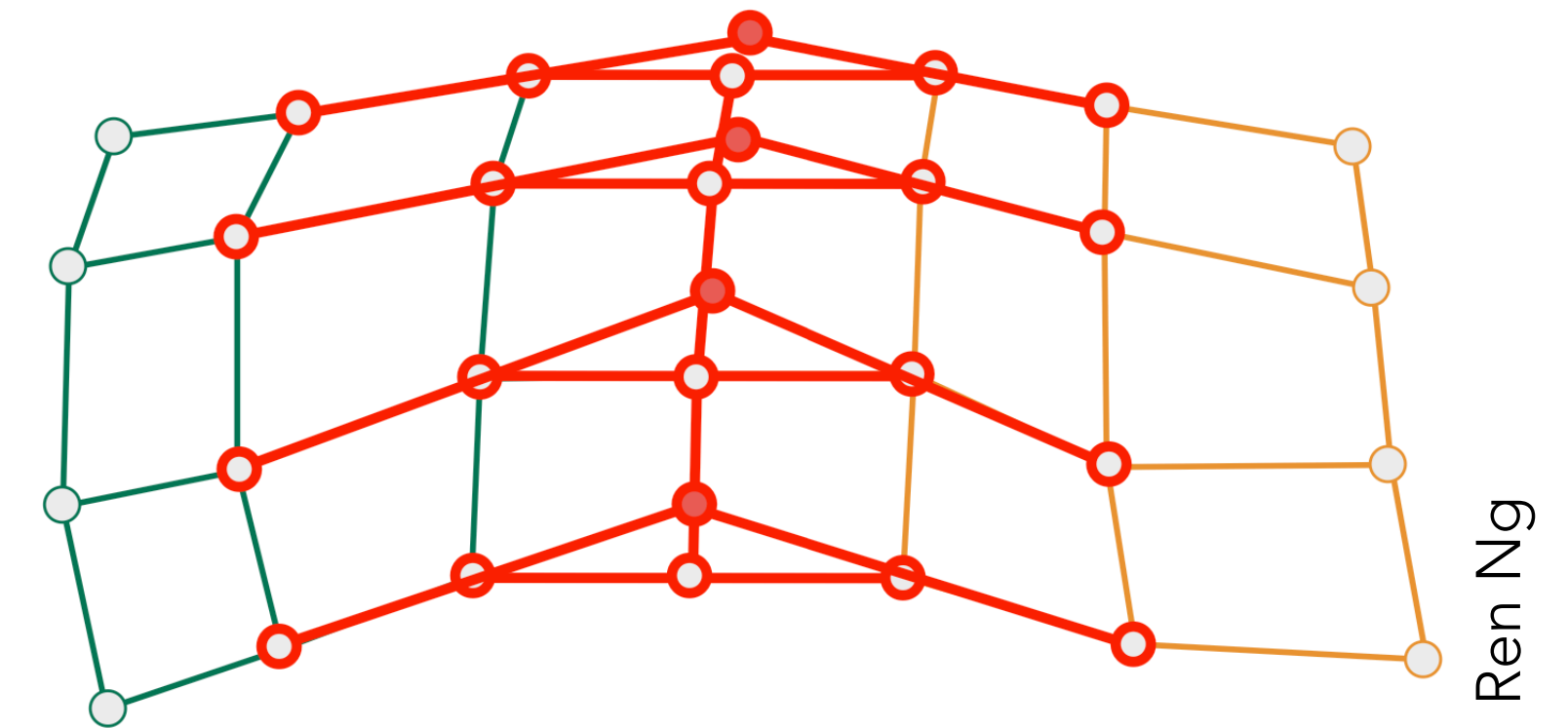
- Determined across the boundary edge between two patches.



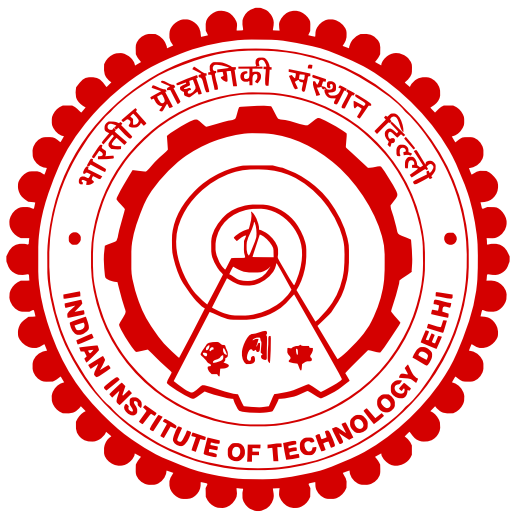
C^0 continuity:
Boundary points agree



C^1 continuity:
Adjacent edges equal



C^2 continuity:
A-frames

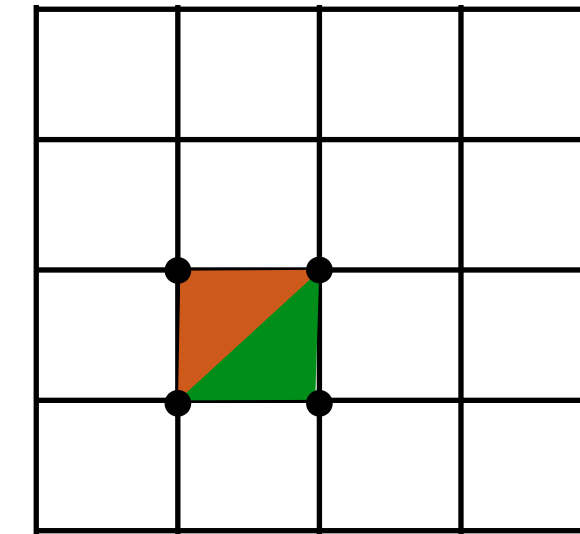


Rendering Splines

- Curves $\rightarrow$ Polyline with $P(i \Delta t)$, for $i \in [0, n]$

$\Rightarrow$ How many points? Uniformly spaced in t ?

$\Rightarrow$ Low curvature requires fewer points



$$|\Delta P|_{\max} = \left| \frac{dP}{dt} \right|_{\max} \Delta t$$

- Surface $\rightarrow$ Points in a uv grid: $P(i \Delta u, j \Delta v)$, for $i \in [0, m], j \in [0, n]$

$\Rightarrow$ Make two triangles per grid-cell

- Adaptive triangulation

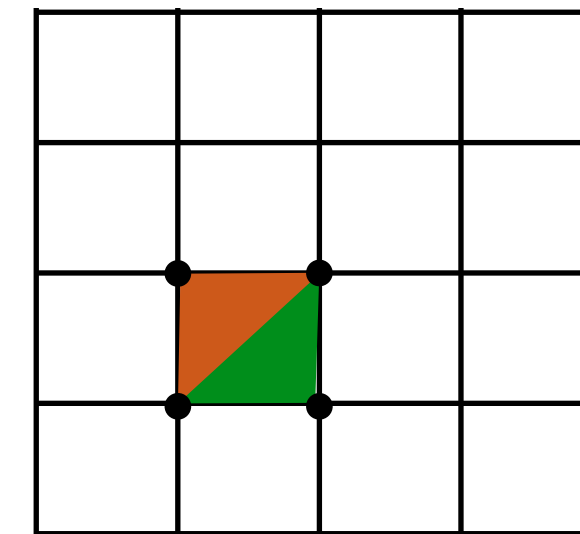


Rendering Splines

- Curves $\rightarrow$ Polyline with $P(i \Delta t)$, for $i \in [0, n]$

$\Rightarrow$ How many points? Uniformly spaced in t ?

$\Rightarrow$ Low curvature requires fewer points

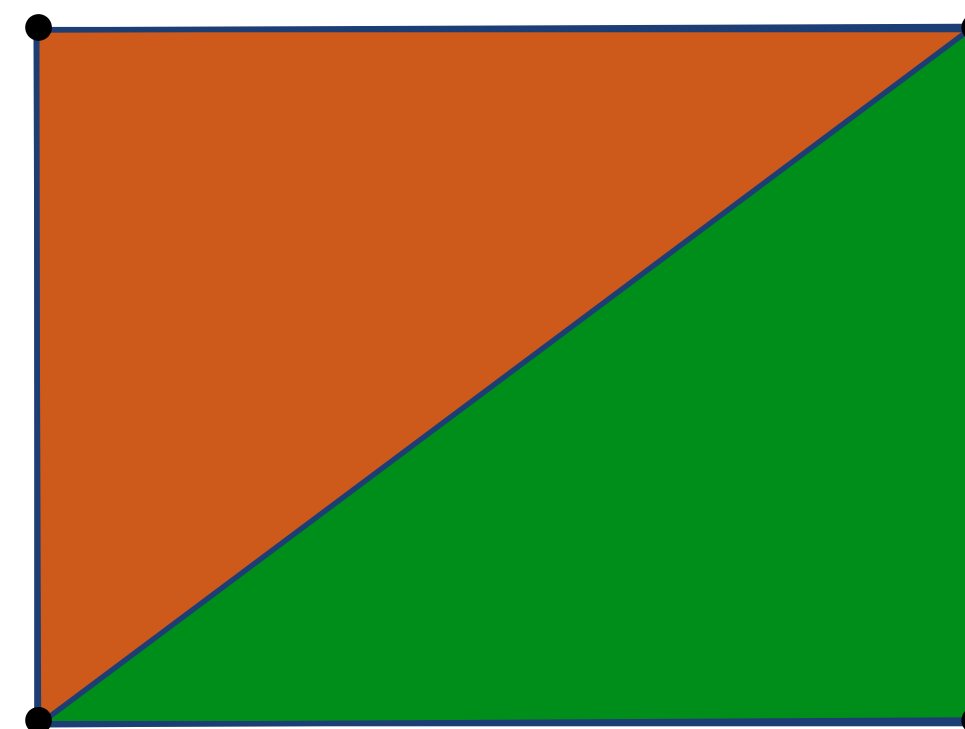


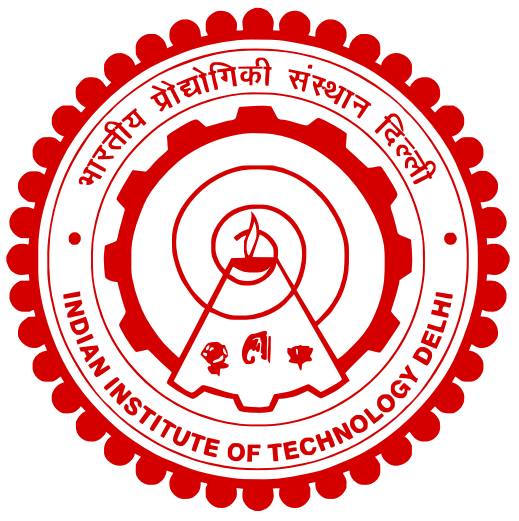
$$|\Delta P|_{\max} = \left| \frac{dP}{dt} \right|_{\max} \Delta t$$

- Surface $\rightarrow$ Points in a uv grid: $P(i \Delta u, j \Delta v)$, for $i \in [0, m], j \in [0, n]$

$\Rightarrow$ Make two triangles per grid-cell

- Adaptive triangulation



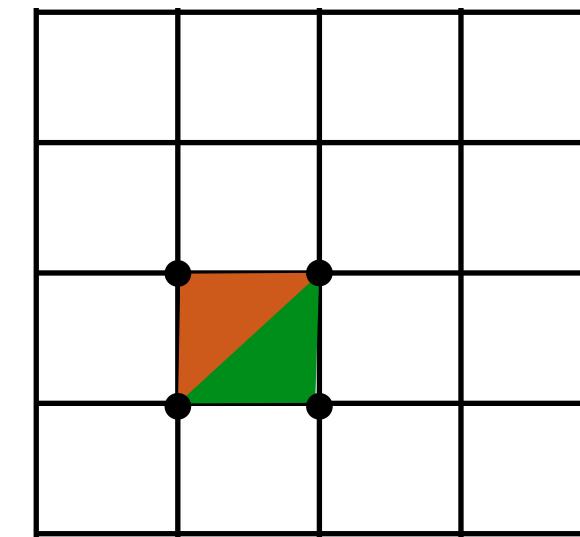


Rendering Splines

- Curves $\rightarrow$ Polyline with $P(i \Delta t)$, for $i \in [0, n]$

$\Rightarrow$ How many points? Uniformly spaced in t ?

$\Rightarrow$ Low curvature requires fewer points

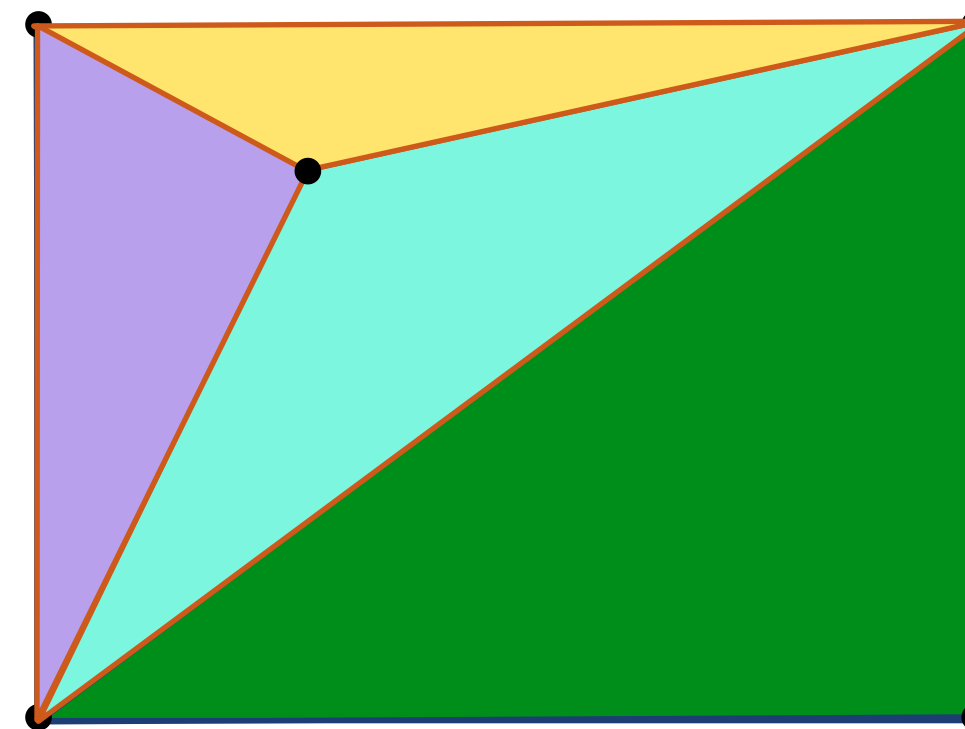


$$|\Delta P|_{\max} = \left| \frac{dP}{dt} \right|_{\max} \Delta t$$

- Surface $\rightarrow$ Points in a uv grid: $P(i \Delta u, j \Delta v)$, for $i \in [0, m], j \in [0, n]$

$\Rightarrow$ Make two triangles per grid-cell

- Adaptive triangulation



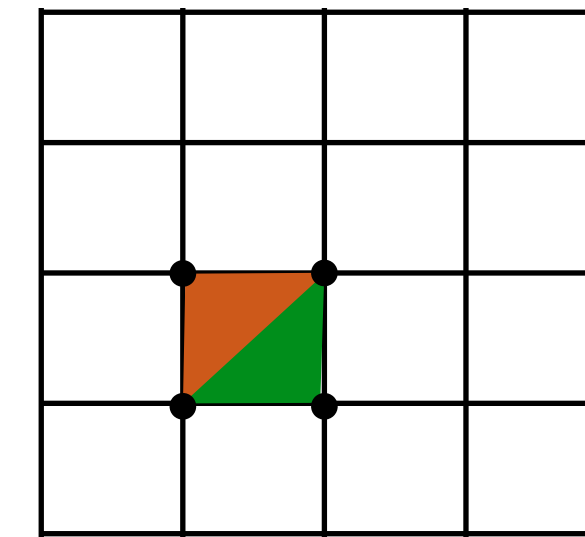
Subdivide if
 $\text{distance}(\text{triangle}, \text{surface}) > \Delta$

Rendering Splines

- Curves $\rightarrow$ Polyline with $P(i \Delta t)$, for $i \in [0, n]$

$\Rightarrow$ How many points? Uniformly spaced in t ?

$\Rightarrow$ Low curvature requires fewer points

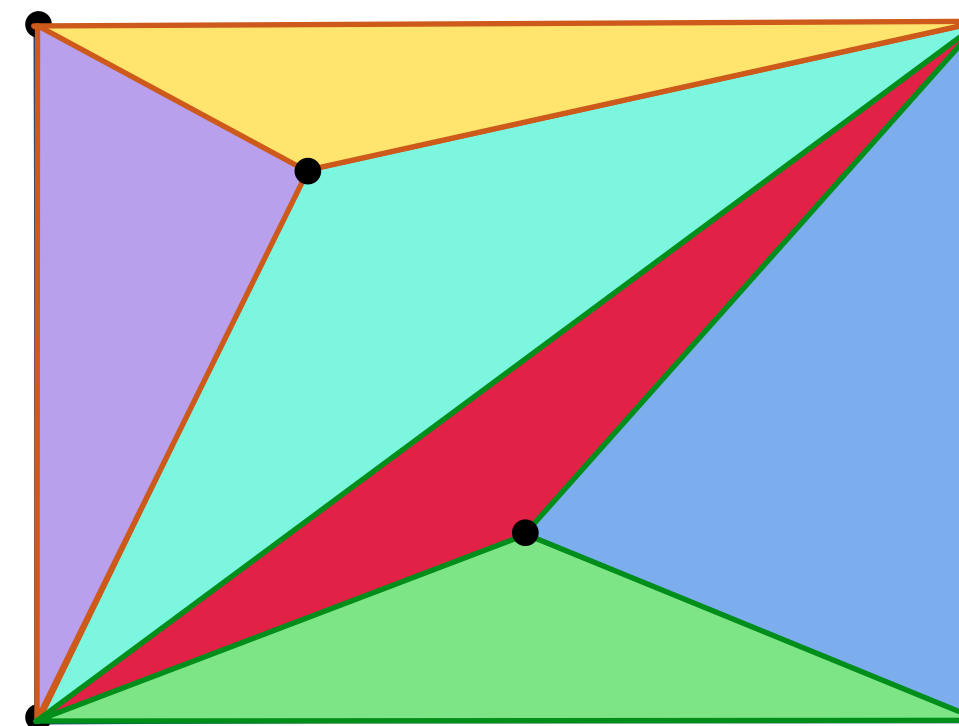


$$|\Delta P|_{\max} = \left| \frac{dP}{dt} \right|_{\max} \Delta t$$

- Surface $\rightarrow$ Points in a uv grid: $P(i \Delta u, j \Delta v)$, for $i \in [0, m], j \in [0, n]$

$\Rightarrow$ Make two triangles per grid-cell

- Adaptive triangulation



Repeatedly

Subdivide if

$\text{distance}(\text{triangle}, \text{surface}) > \Delta$

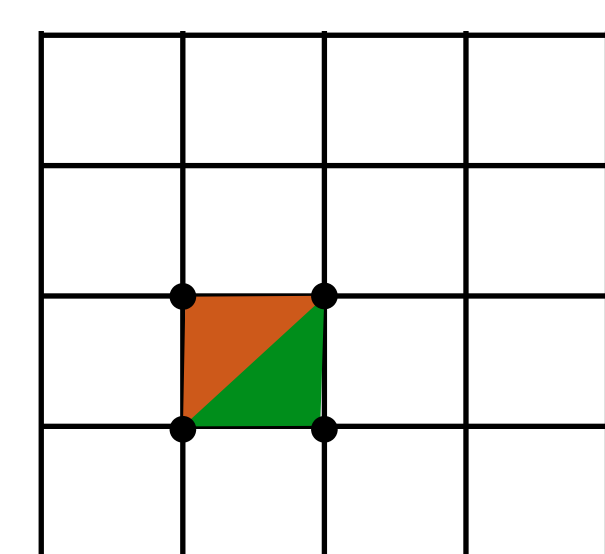


Rendering Splines

- Curves $\rightarrow$ Polyline with $P(i \Delta t)$, for $i \in [0, n]$

$\Rightarrow$ How many points? Uniformly spaced in t ?

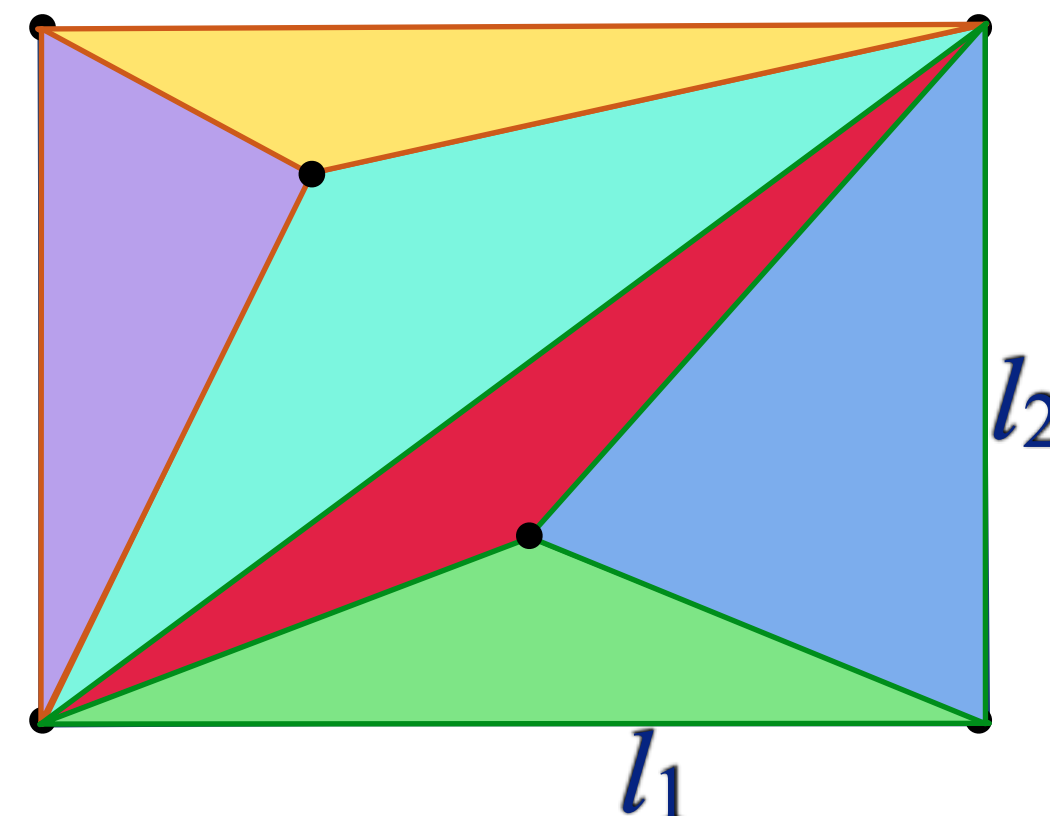
$\Rightarrow$ Low curvature requires fewer points


$$|\Delta P|_{\max} = \left| \frac{dP}{dt} \right|_{\max} \Delta t$$

- Surface $\rightarrow$ Points in a uv grid: $P(i \Delta u, j \Delta v)$, for $i \in [0, m], j \in [0, n]$

$\Rightarrow$ Make two triangles per grid-cell

- Adaptive triangulation

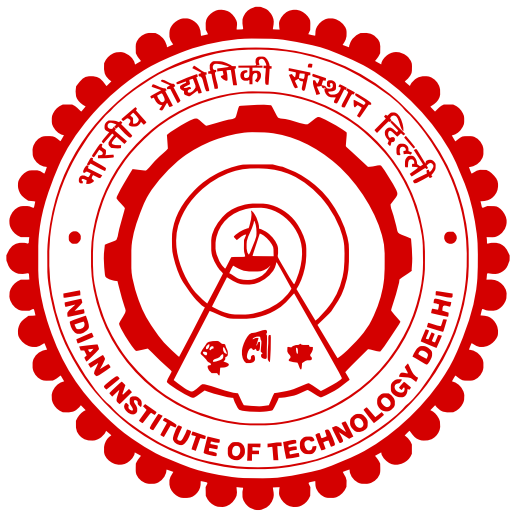


Repeatedly
Subdivide if
 $\text{distance}(\text{triangle}, \text{surface}) > \Delta$

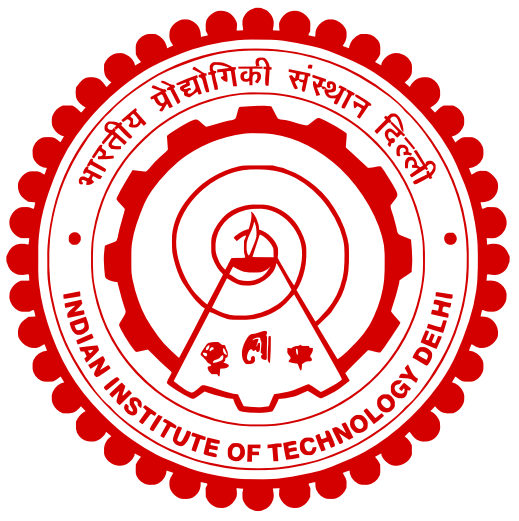
Bounded by

$$\frac{1}{8} \left(l_1^2 |P_{uu}|_{\max} + 2l_1l_2 |P_{uv}|_{\max} + l_2^2 |P_{vv}|_{\max} \right)$$

Making Surfaces

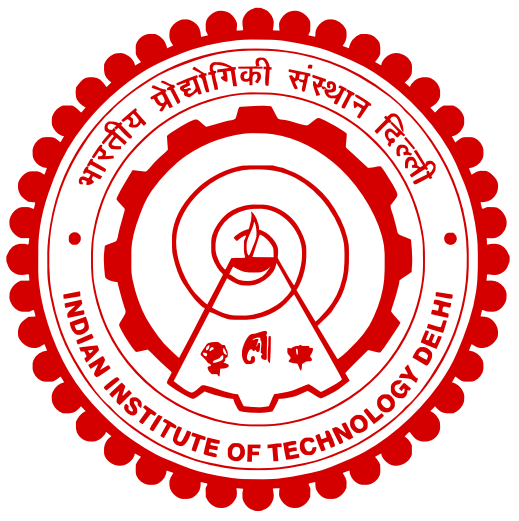


- Control mesh
- Lofted surfaces
- Ruled surfaces
- Surface of revolution
- Free-form design



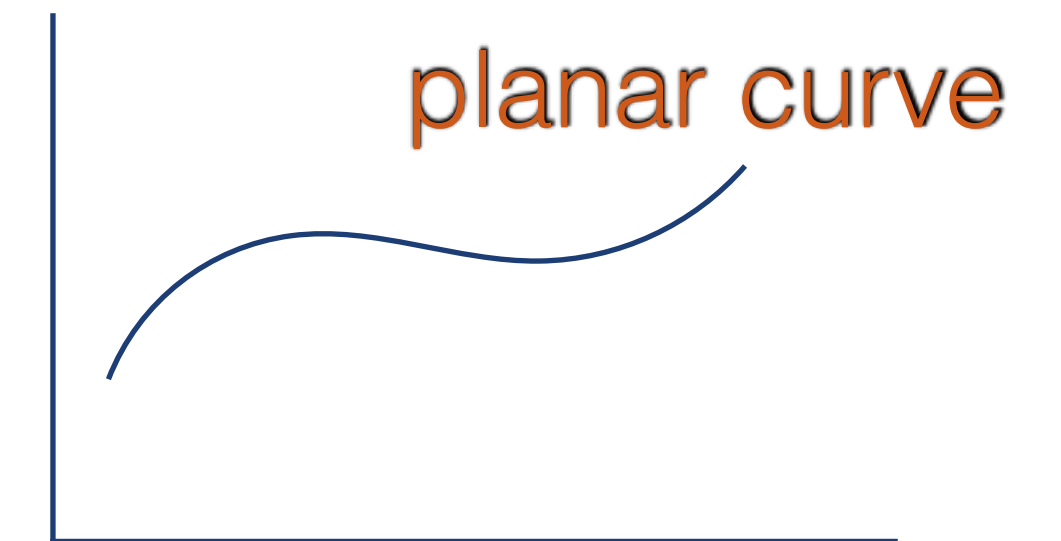
Design Primitives

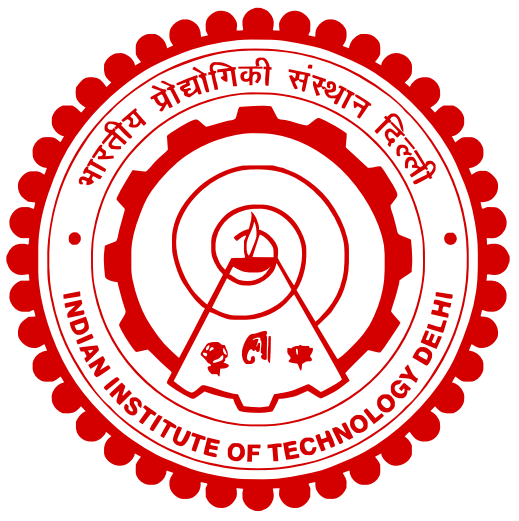
- Simple geometric structures (cones, cylinders, spheres etc.)
- Algebraic half-spaces/Implicit surfaces
- Parametric surfaces
- Surface of revolution/Ruled surfaces
- Sweep/Tool path
- Discretized structures
 - ➔ Triangles, points, subdivision surfaces



Design Primitives

- Simple geometric structures (cones, cylinders, spheres etc.)
- Algebraic half-spaces/Implicit surfaces
- Parametric surfaces
- Surface of revolution/Ruled surfaces
- Sweep/Tool path
- Discretized structures
 - ➔ Triangles, points, subdivision surfaces

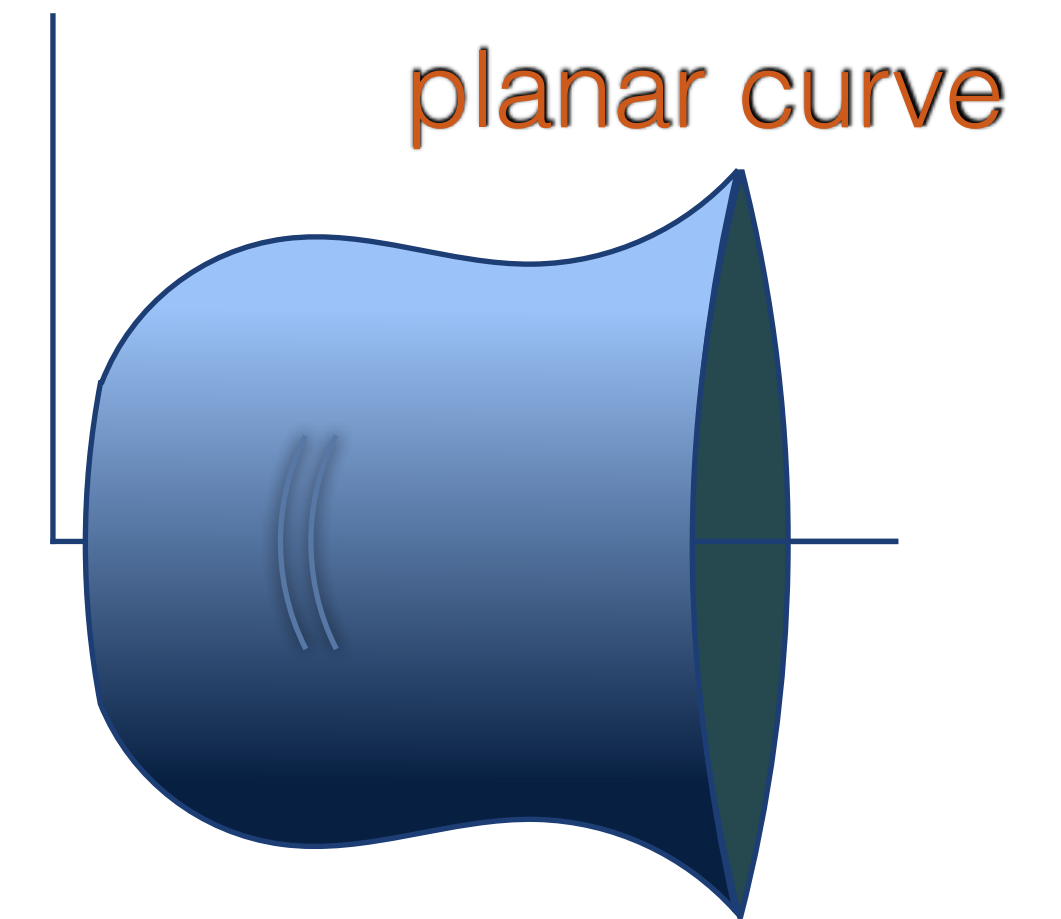


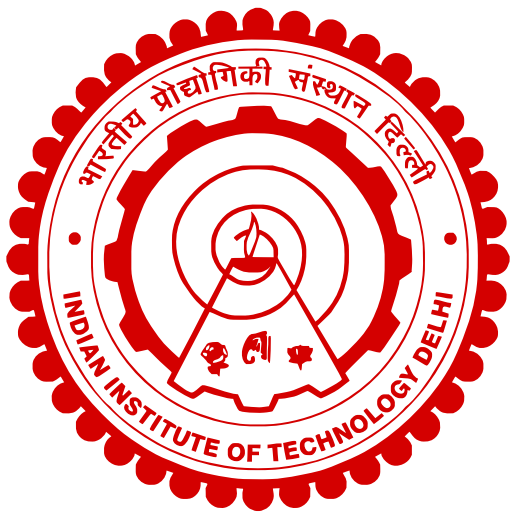


Design Primitives

- Simple geometric structures (cones, cylinders, spheres etc.)
- Algebraic half-spaces/Implicit surfaces
- Parametric surfaces
- Surface of revolution/Ruled surfaces
- Sweep/Tool path
- Discretized structures
 - ➔ Triangles, points, subdivision surfaces

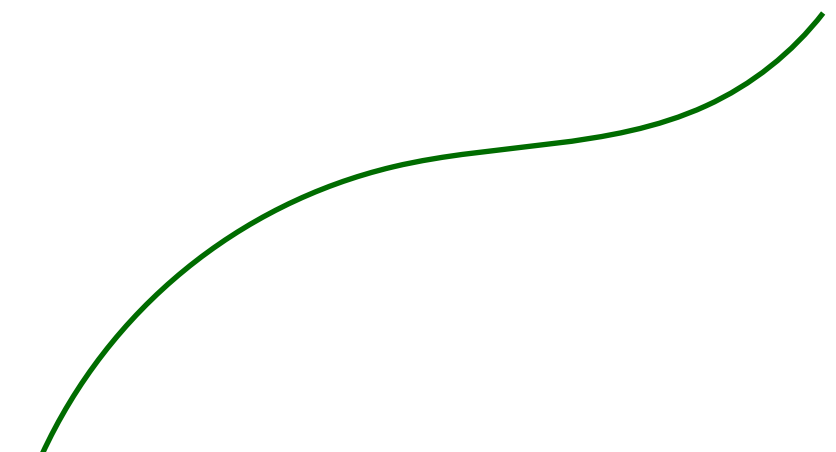
$$x(u), y(u)\cos v, y(u)\sin v$$

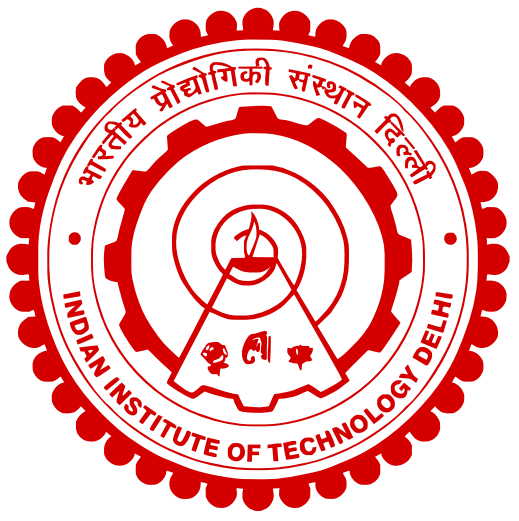




Design Primitives

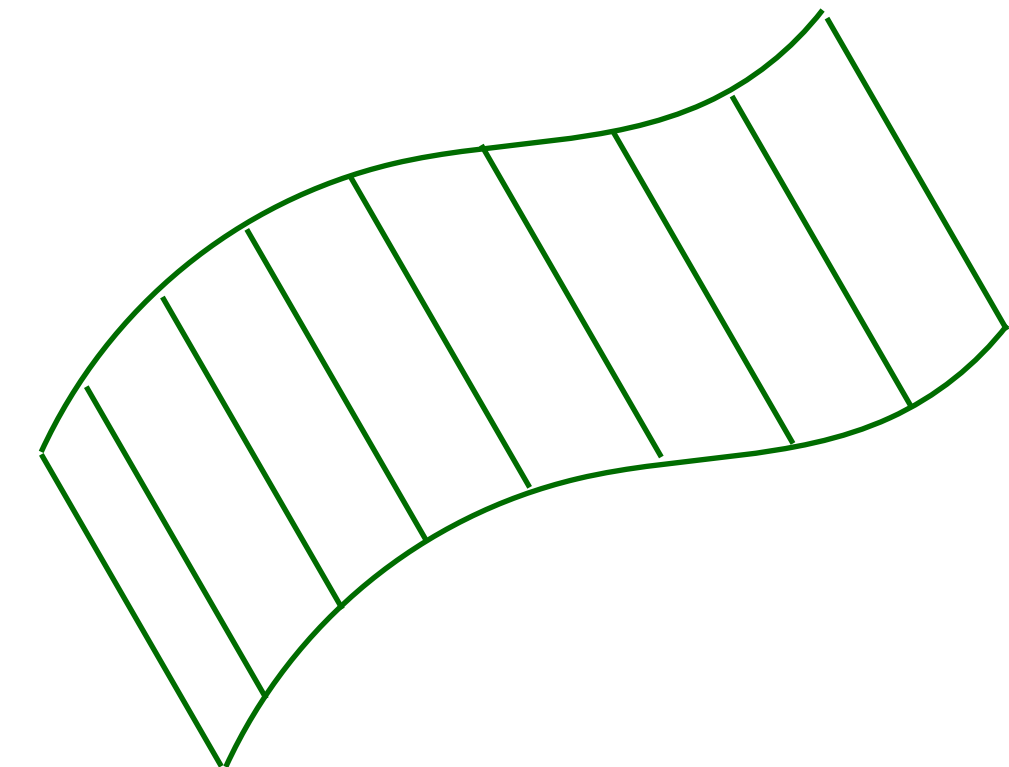
- Simple geometric structures (cones, cylinders, spheres etc.)
- Algebraic half-spaces/Implicit surfaces
- Parametric surfaces
- Surface of revolution/Ruled surfaces
- Sweep/Tool path
- Discretized structures
 - ➔ Triangles, points, subdivision surfaces





Design Primitives

- Simple geometric structures (cones, cylinders, spheres etc.)
- Algebraic half-spaces/Implicit surfaces
- Parametric surfaces
- Surface of revolution/Ruled surfaces
- Sweep/Tool path
- Discretized structures
 - ➔ Triangles, points, subdivision surfaces





Design Primitives

- Simple geometric structures (cones, cylinders, spheres etc.)

- Algebraic half-spaces/Implicit surfaces

- Parametric surfaces

$$x(u), y(u)\cos v, y(u)\sin v$$

- Surface of revolution/Ruled surfaces (Developable)

- Sweep/Tool path

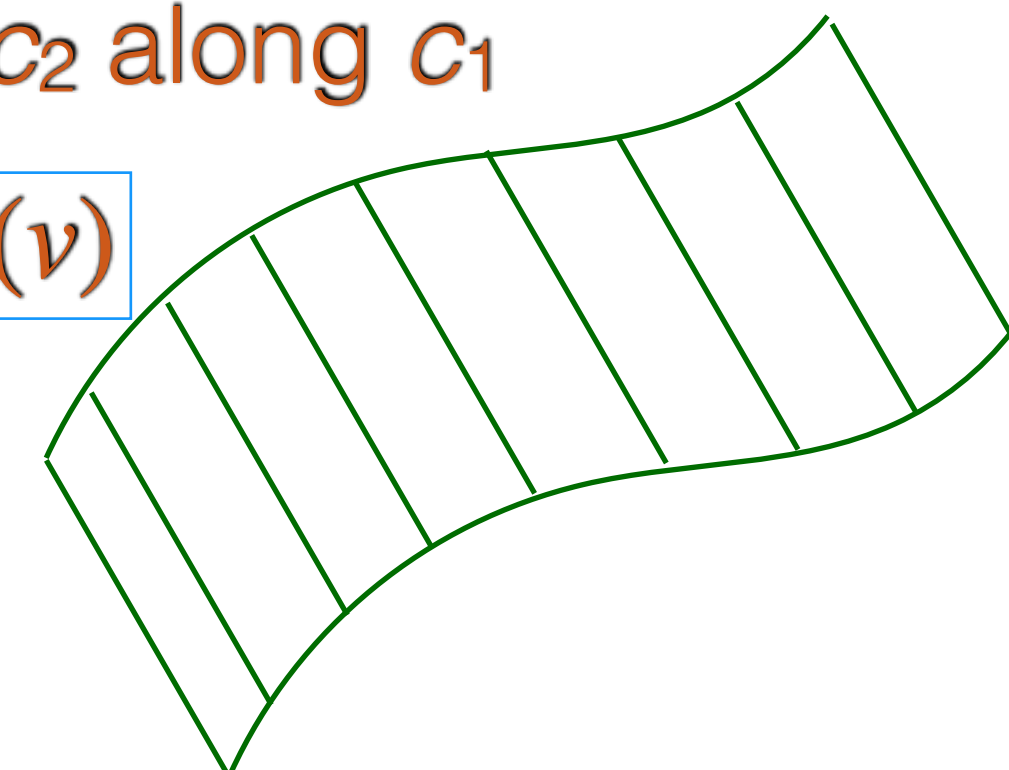
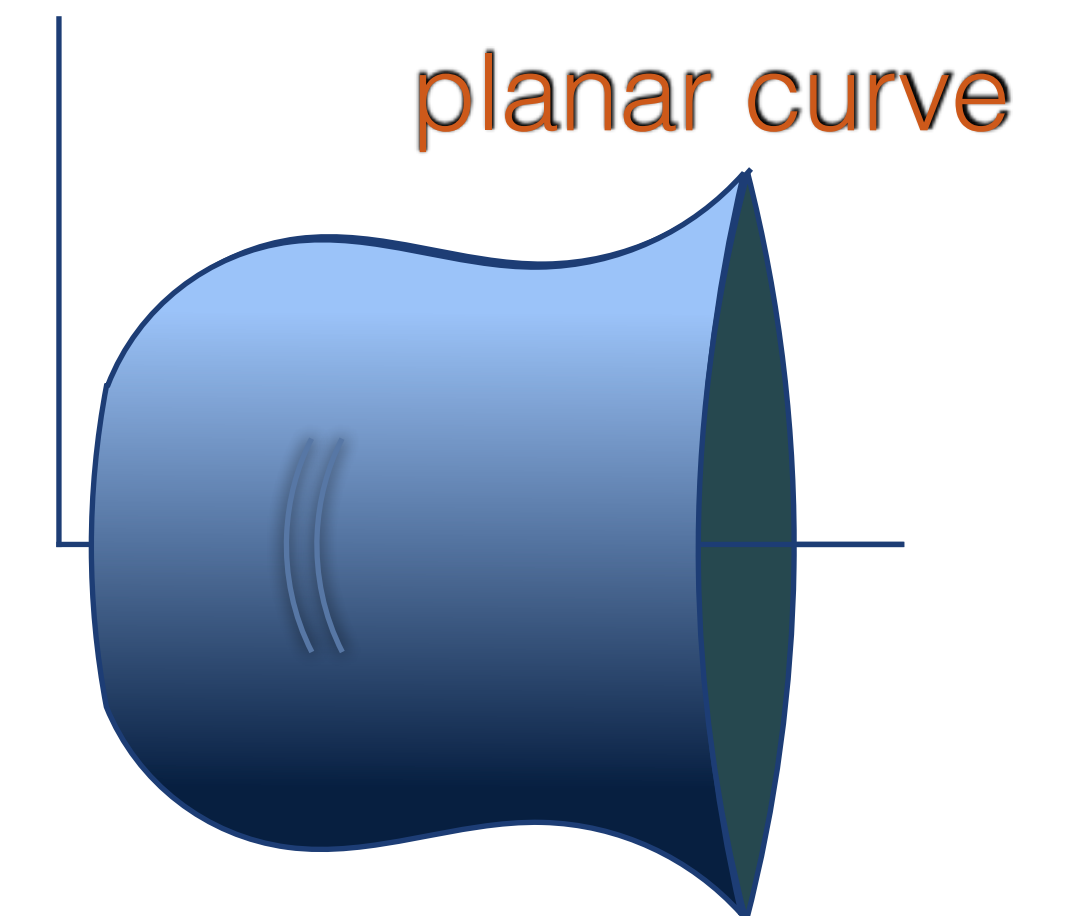
$$S(u, v) = (1 - v)c_1(u) + vc_2(u)$$

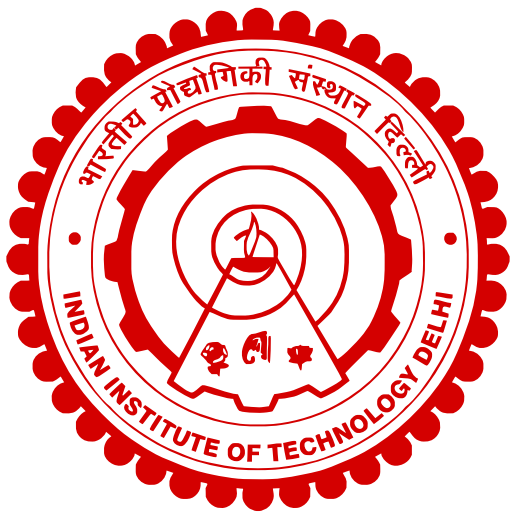
Given two curves $c_1(t)$ and $c_2(t)$, move ($\Rightarrow$ Translate/Rotate/Scale) c_2 along c_1

- Discretized structures

$$S(u, v) = c_1(u) + T(u)c_2(v)$$

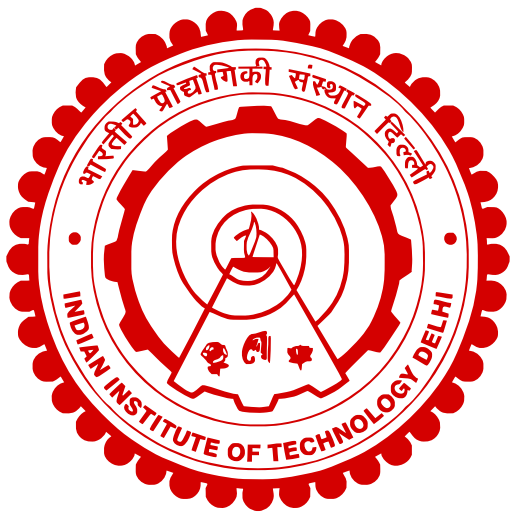
➔ Triangles, points, subdivision surfaces





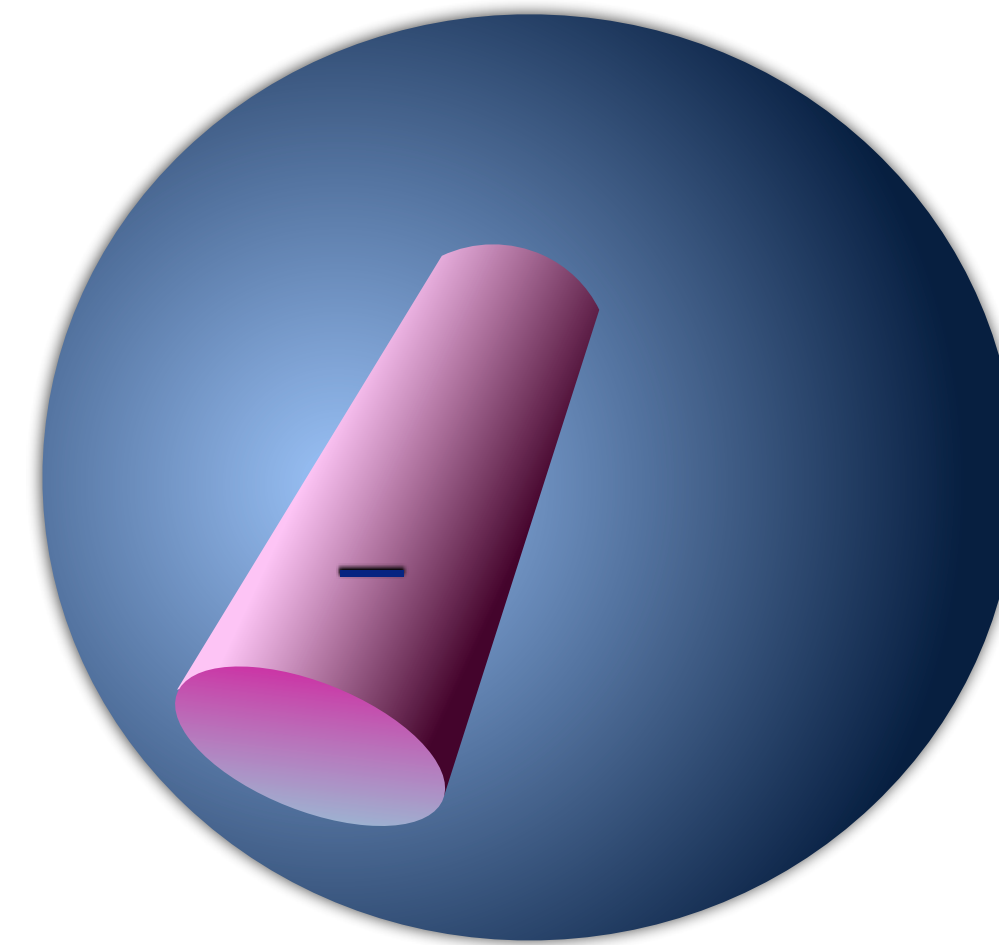
Design Operations

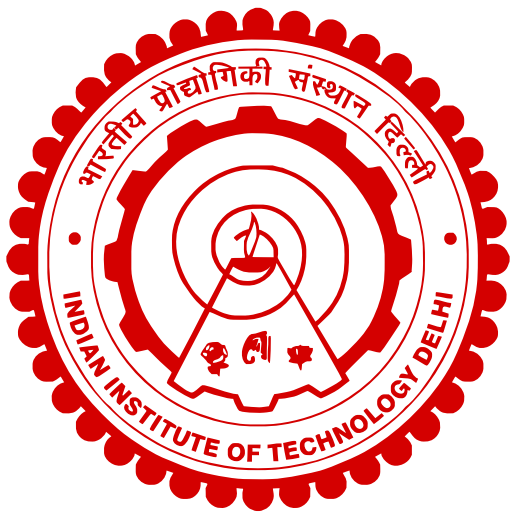
- Local
 - ➡ Cut, extrude, join etc.
 - ➡ Pre-defined operations: raise, dent ..
- Global
 - ➡ Transform
 - ➡ CSG (Regularized) etc.
 - ➡ Blend



Design Operations

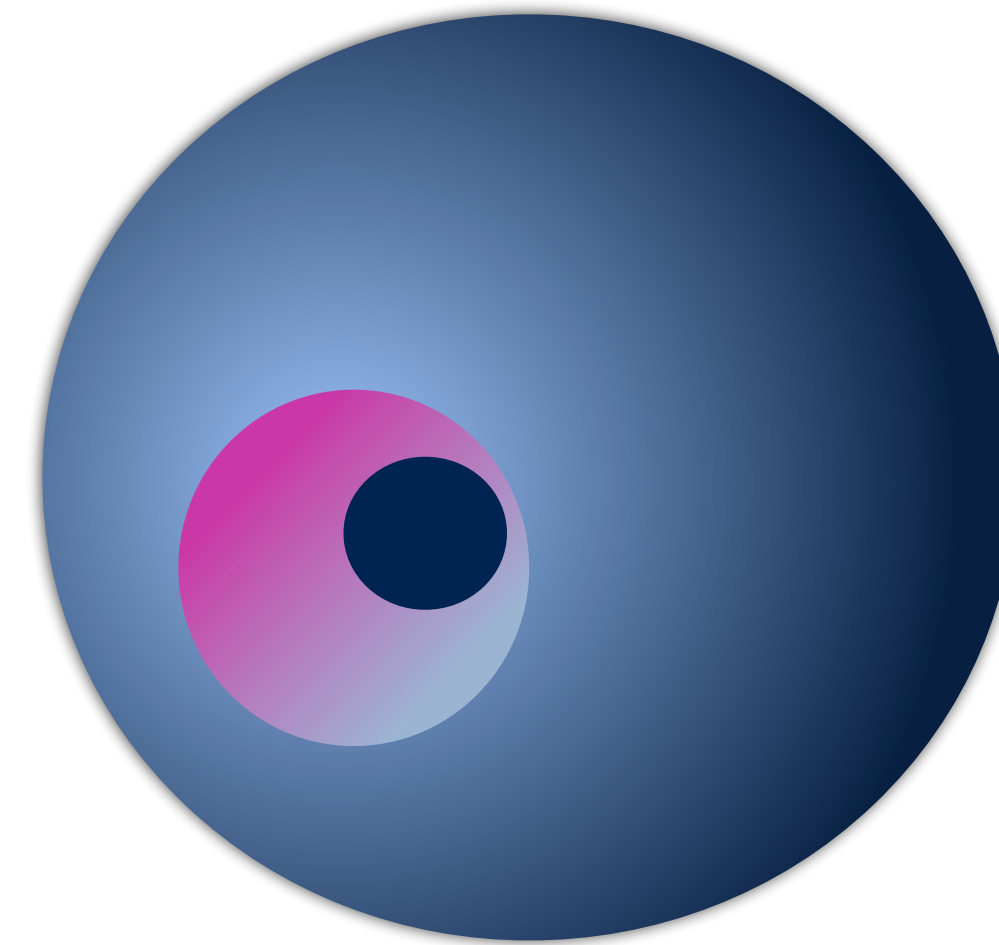
- Local
 - ➡ Cut, extrude, join etc.
 - ➡ Pre-defined operations: raise, dent ..
- Global
 - ➡ Transform
 - ➡ CSG (Regularized) etc.
 - ➡ Blend

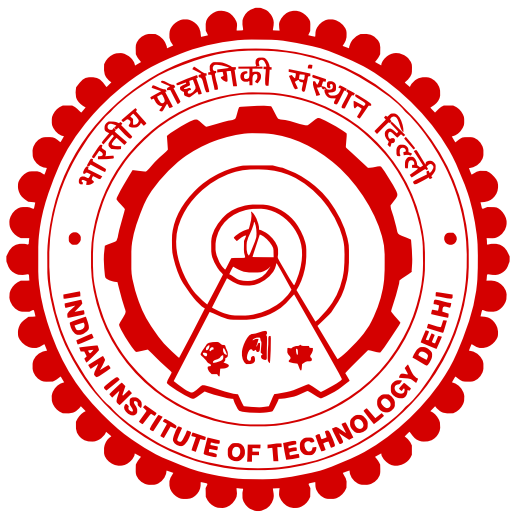




Design Operations

- Local
 - ➡ Cut, extrude, join etc.
 - ➡ Pre-defined operations: raise, dent ..
- Global
 - ➡ Transform
 - ➡ CSG (Regularized) etc.
 - ➡ Blend

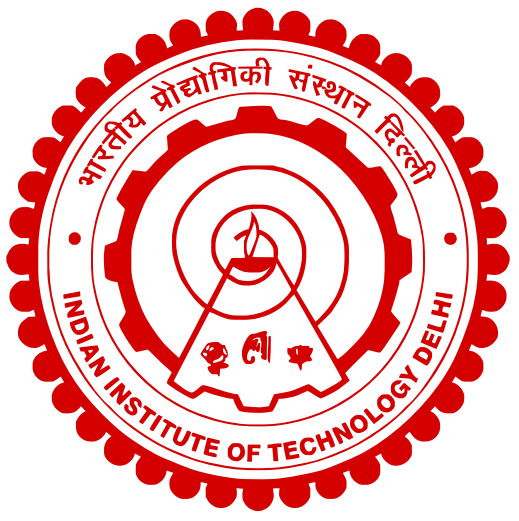




Design Operations

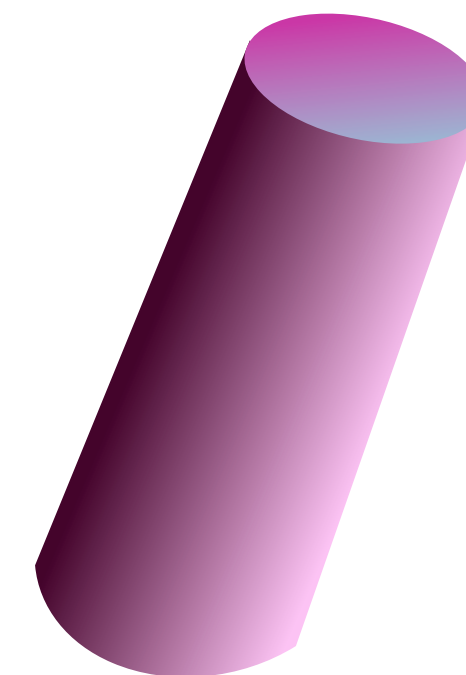
- Local
 - ➡ Cut, extrude, join etc.
 - ➡ Pre-defined operations: raise, dent ..
- Global
 - ➡ Transform
 - ➡ CSG (Regularized) etc.
 - ➡ Blend

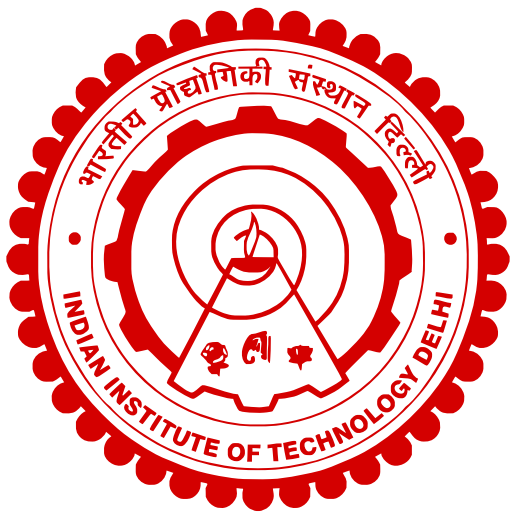




Design Operations

- Local
 - ➔ Cut, extrude, join etc.
 - ➔ Pre-defined operations: raise, dent ..
- Global
 - ➔ Transform
 - ➔ CSG (Regularized) etc.
 - ➔ Blend





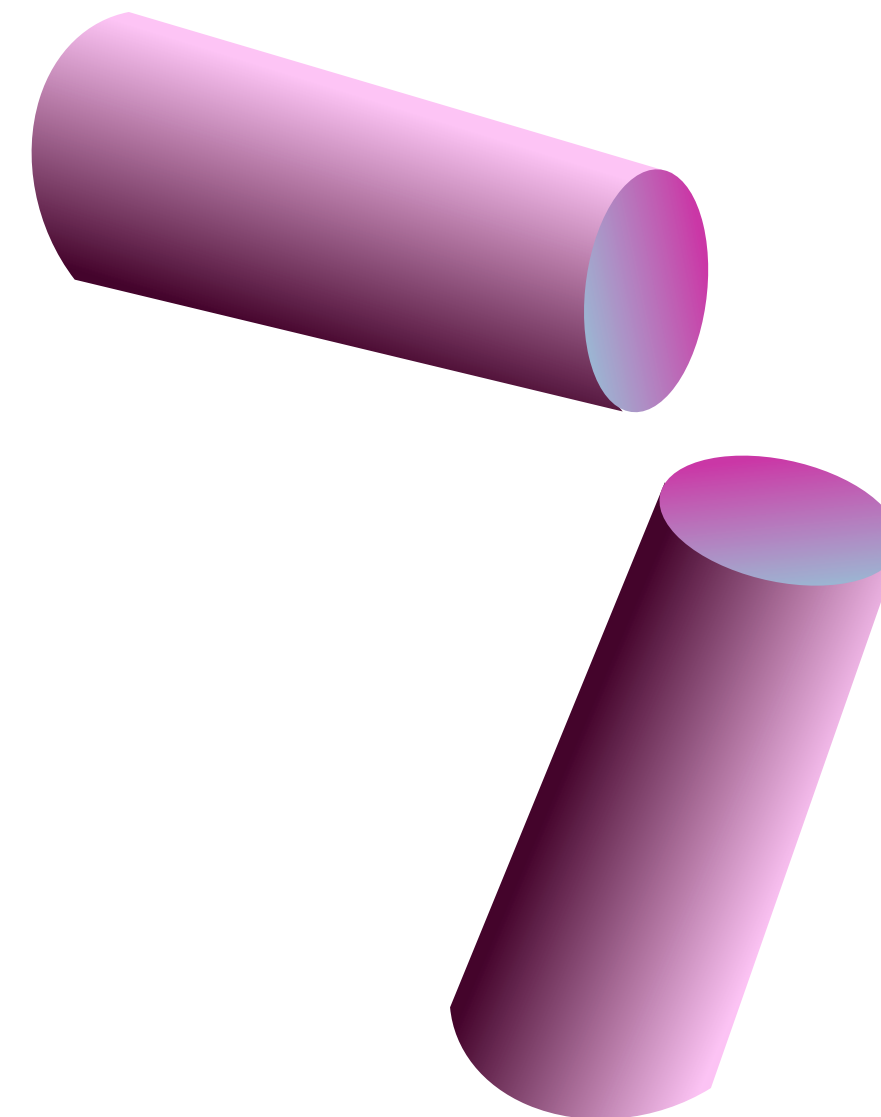
Design Operations

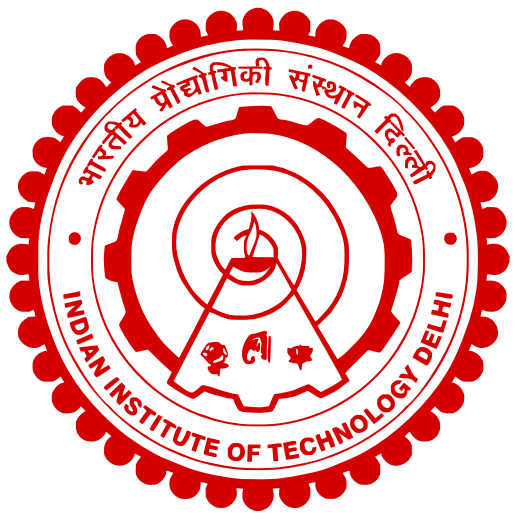
- Local

- ➔ Cut, extrude, join etc.
- ➔ Pre-defined operations: raise, dent ..

- Global

- ➔ Transform
- ➔ CSG (Regularized) etc.
- ➔ Blend





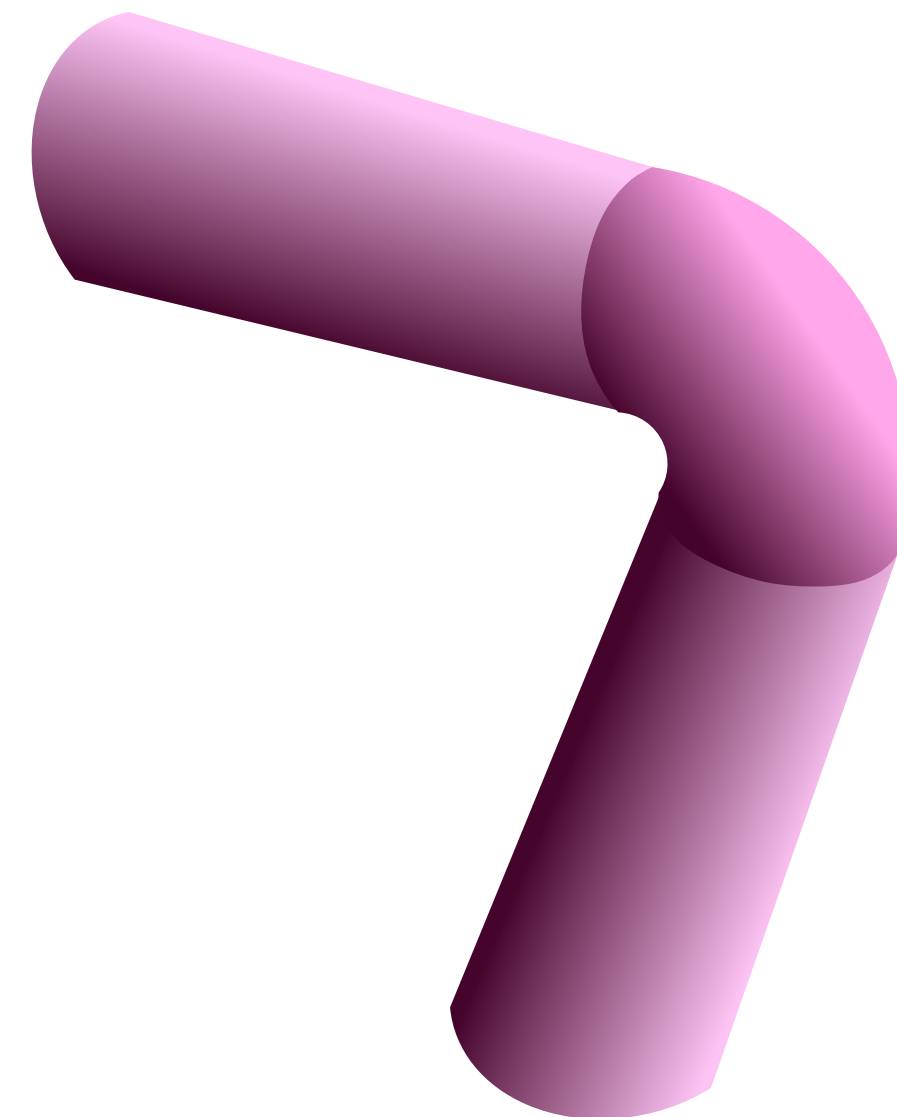
Design Operations

- Local

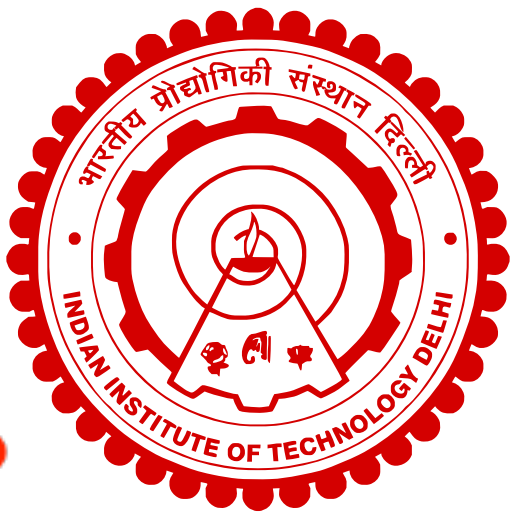
- ➔ Cut, extrude, join etc.
- ➔ Pre-defined operations: raise, dent ..

- Global

- ➔ Transform
- ➔ CSG (Regularized) etc.
- ➔ Blend



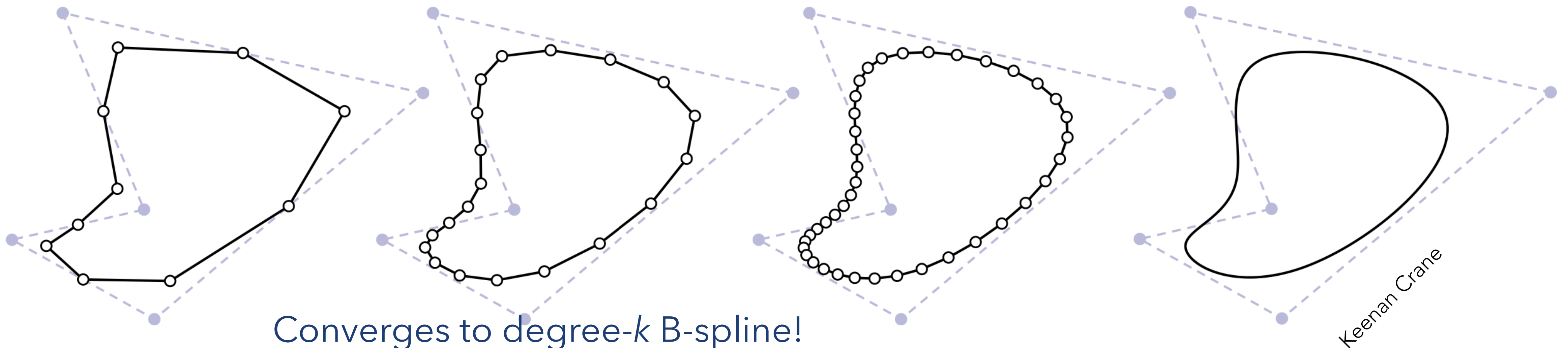
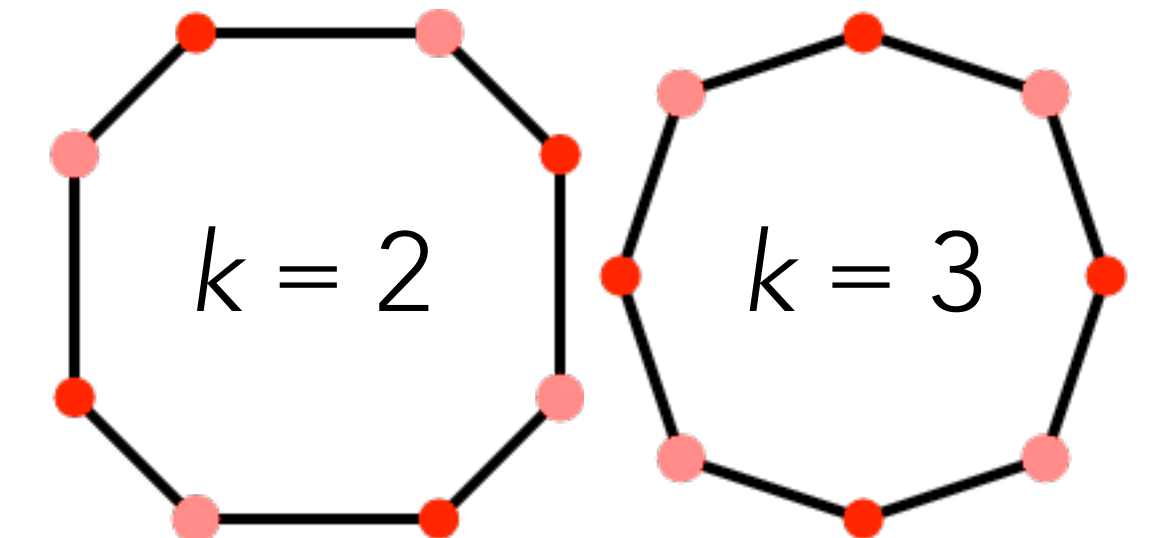
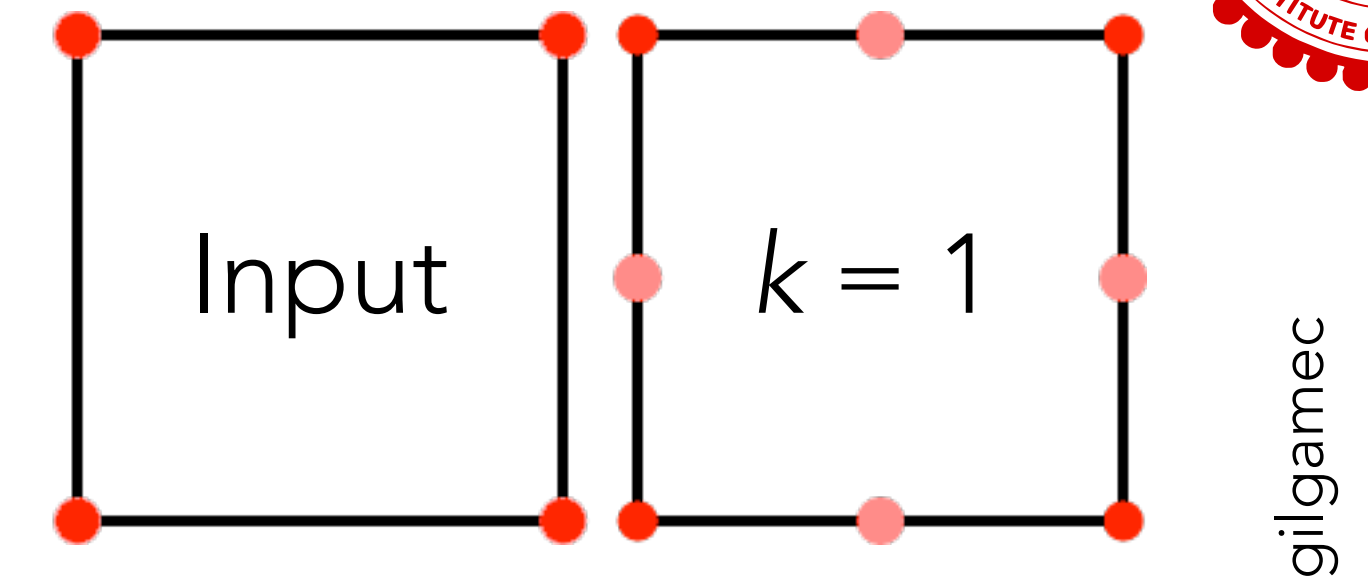
Subdivision curves



Lane-Riesenfeld curve subdivision rule:

- Add new vertex at edges
- Do k times: Average adjacent vertices

This gives a new polygon with $2n$ vertices. Repeat

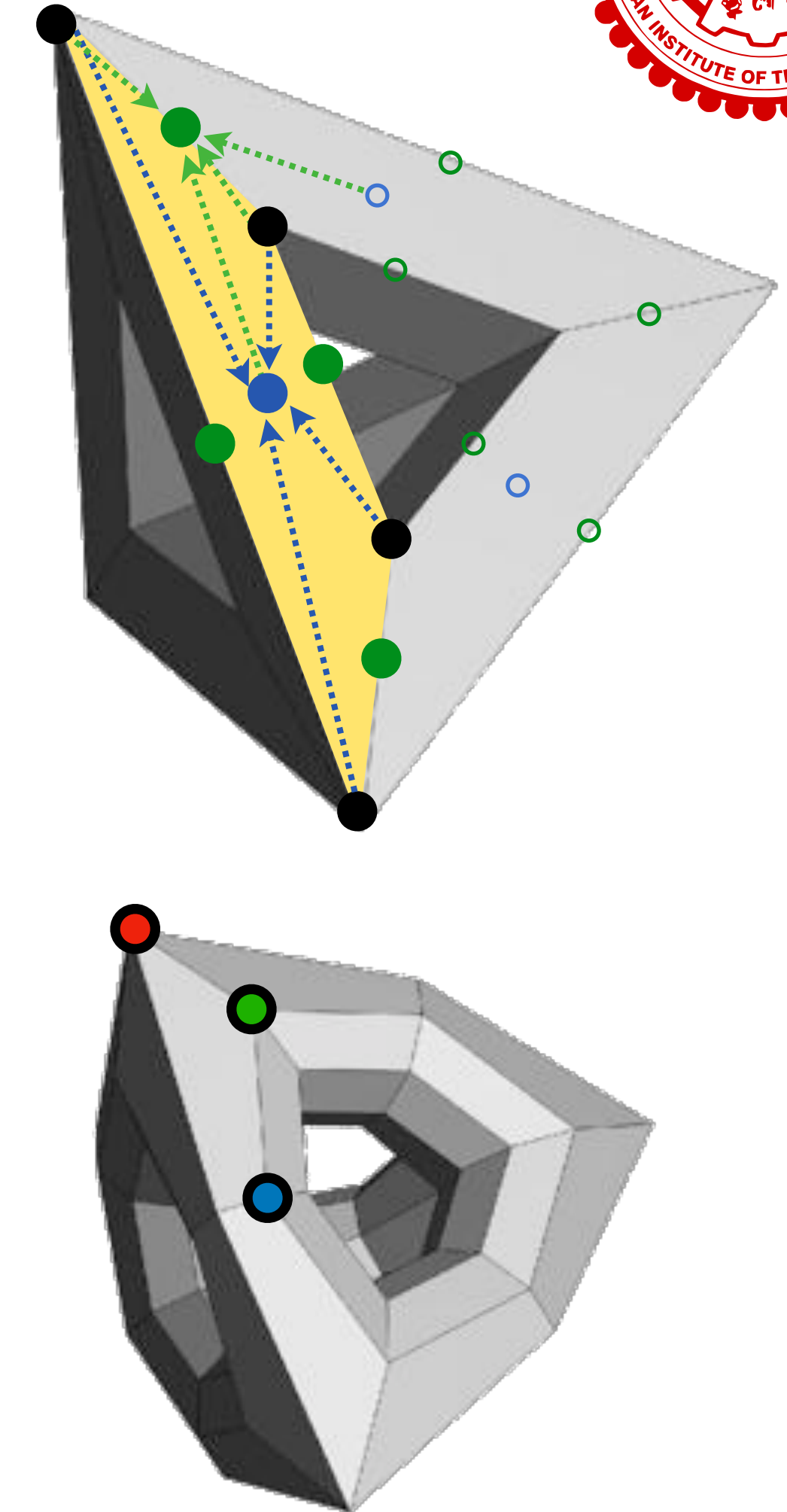


Keenan Crane

Catmull-Clark Subdivision Surface



- Split each n -sided face into n quads
- Update vertex positions by averaging
 - ➔ New face vertices = average of old face vertices
 - ➔ **New edge vertices** = average of 2 old vertices and 2 new face points
 - ➔ **Update old vertices** = $\frac{1}{n} (F + 2E + (n-3)S)$
where F = average of n new face vertices,
 E = average of n new edge vertices
 S = old vertex



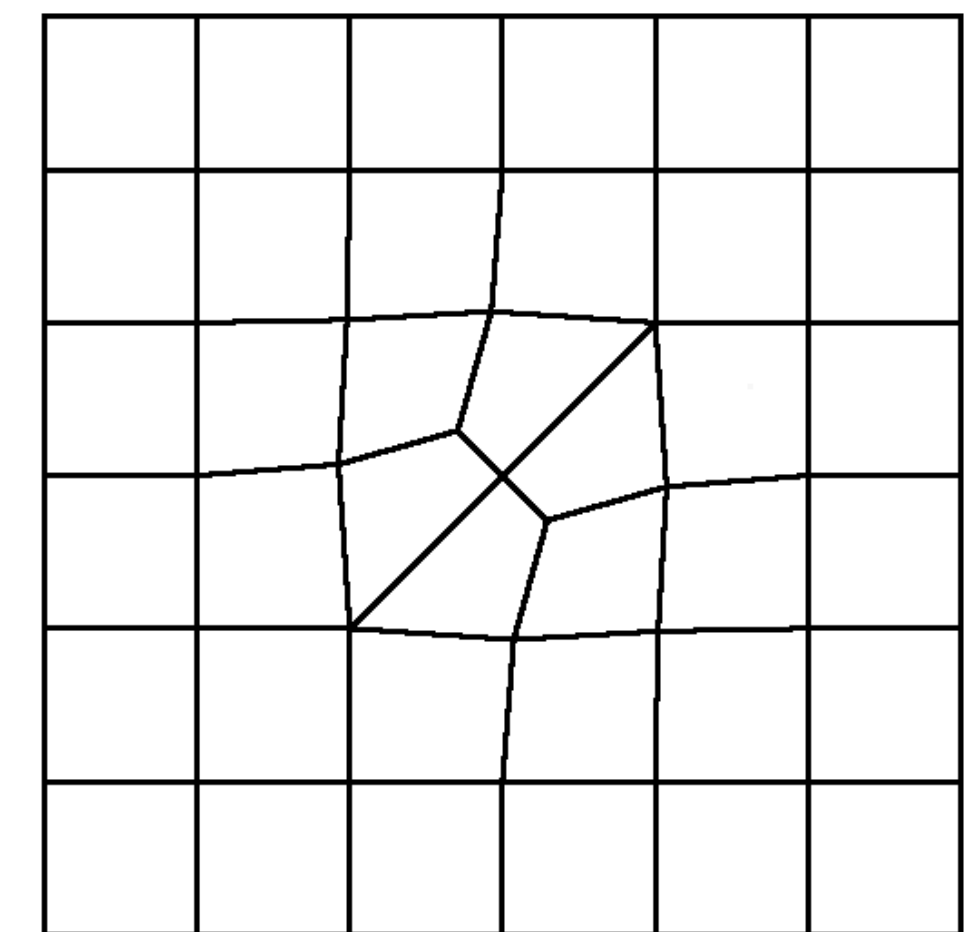
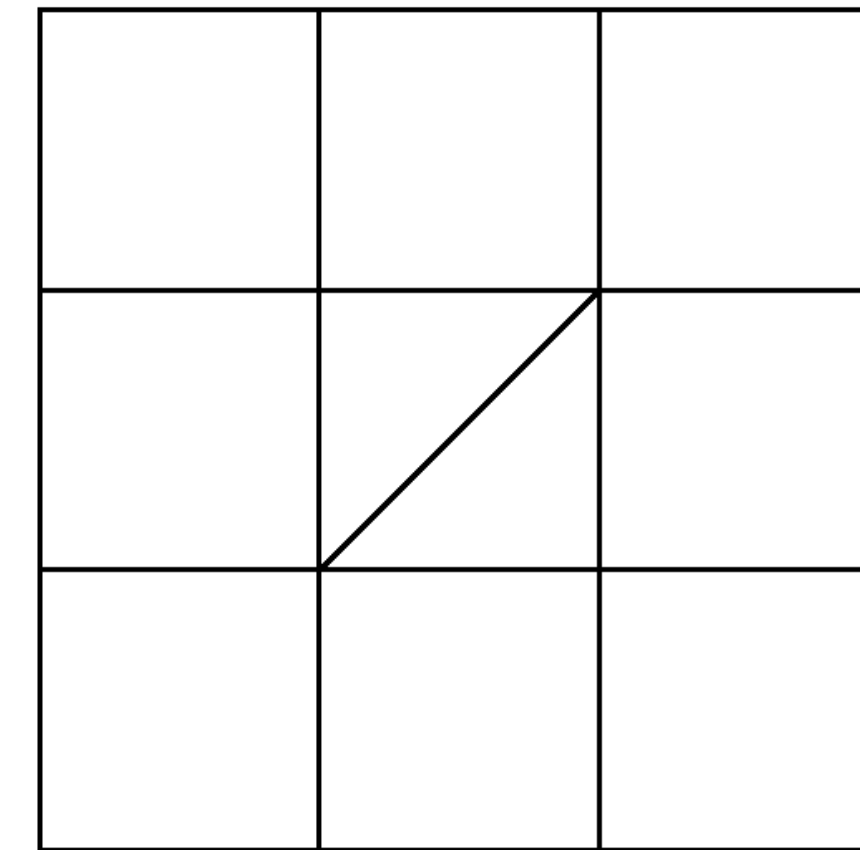
Catmull-Clark subdivision



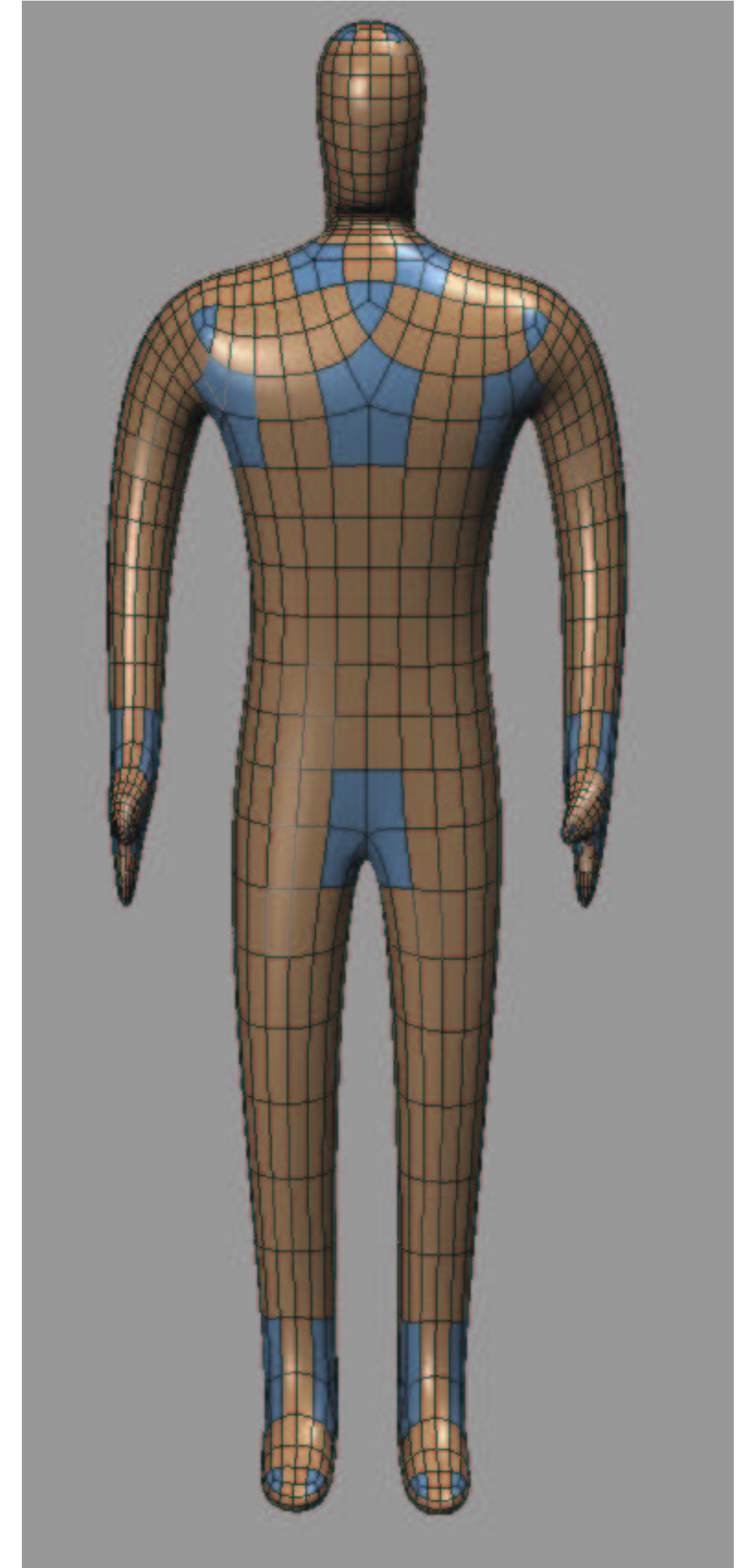
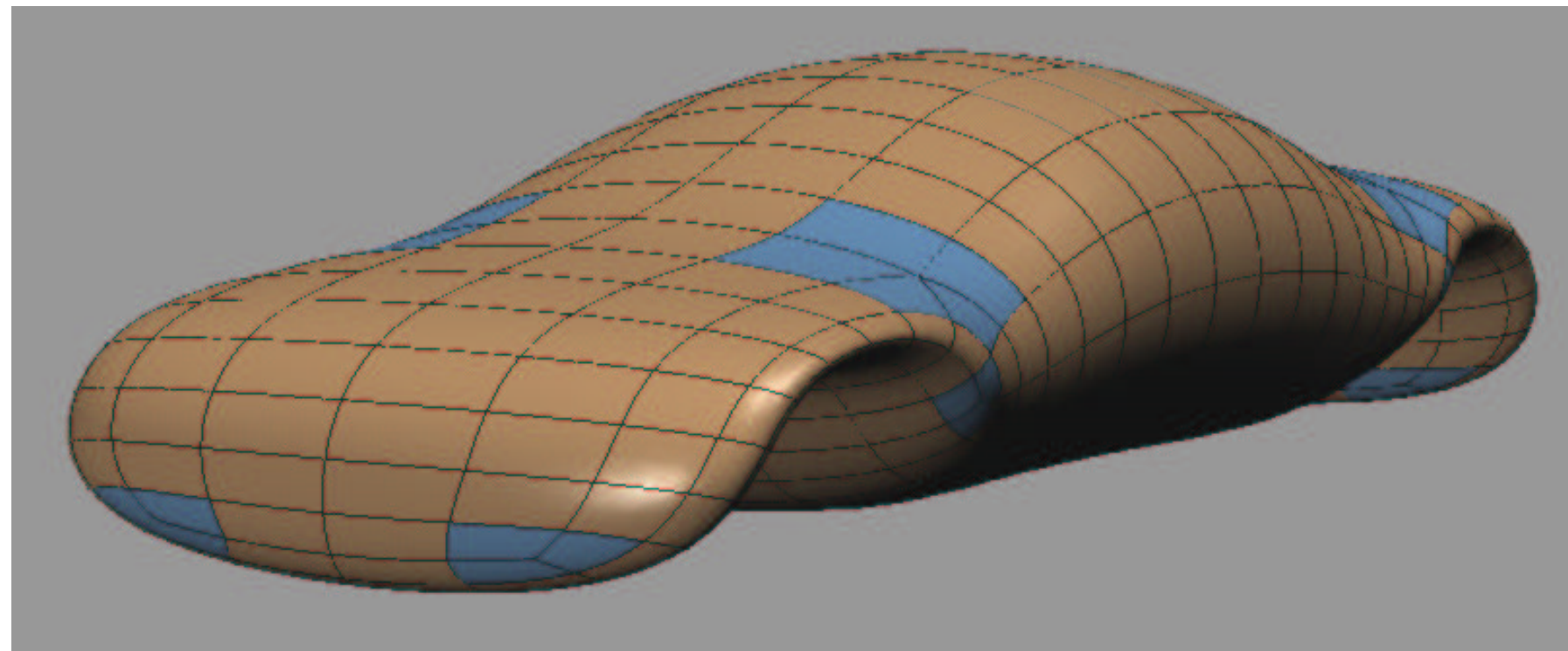
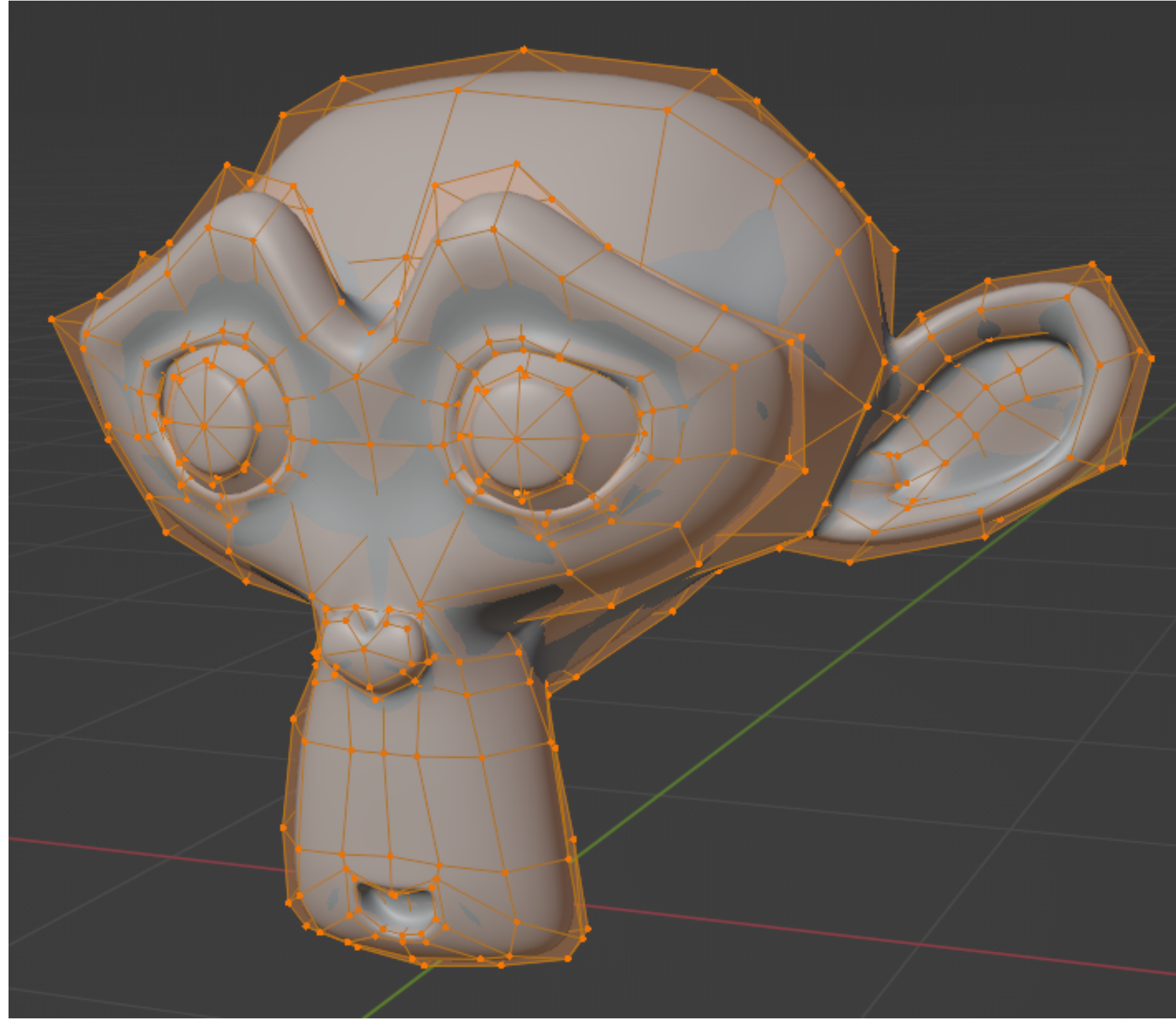
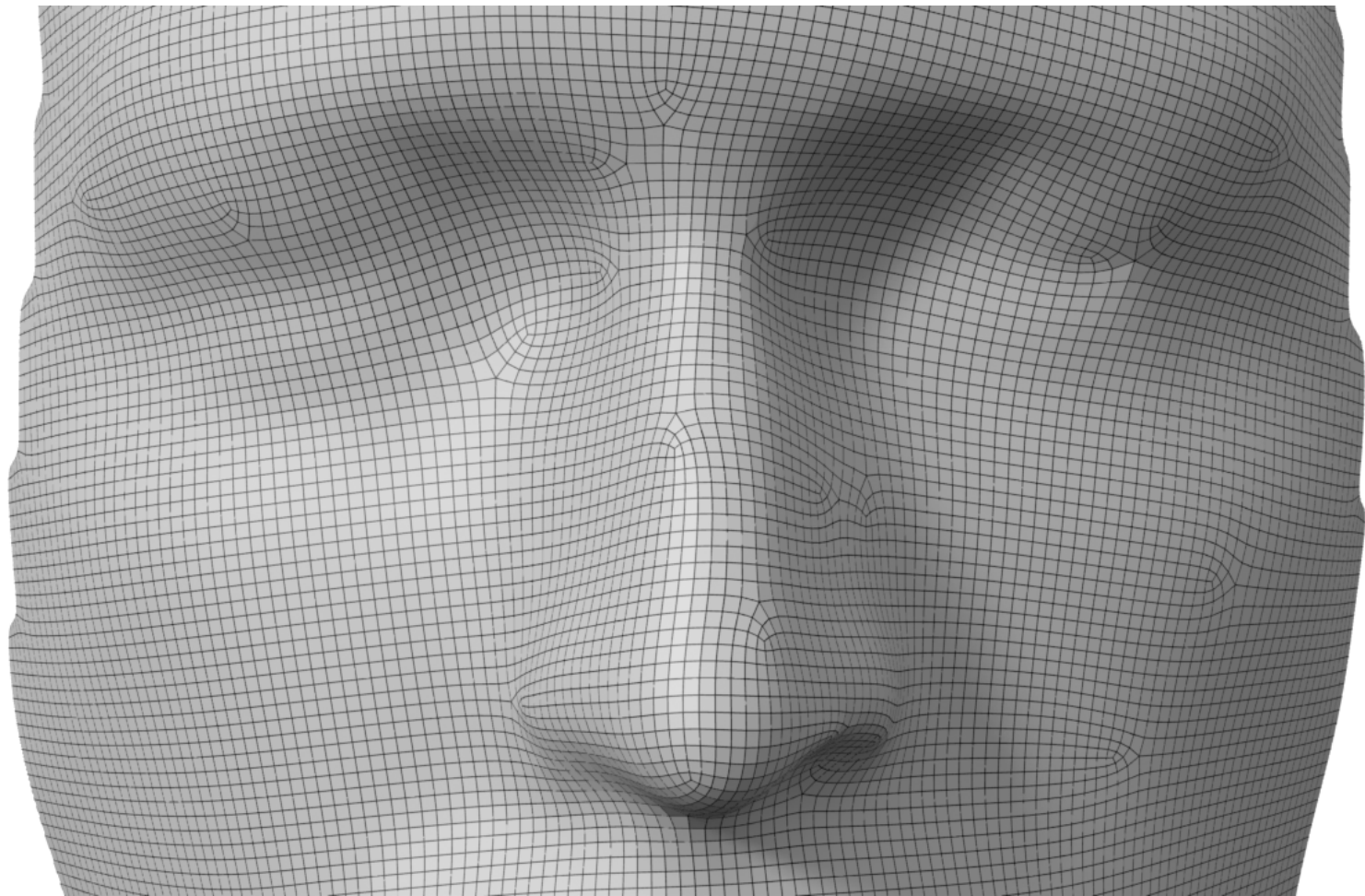
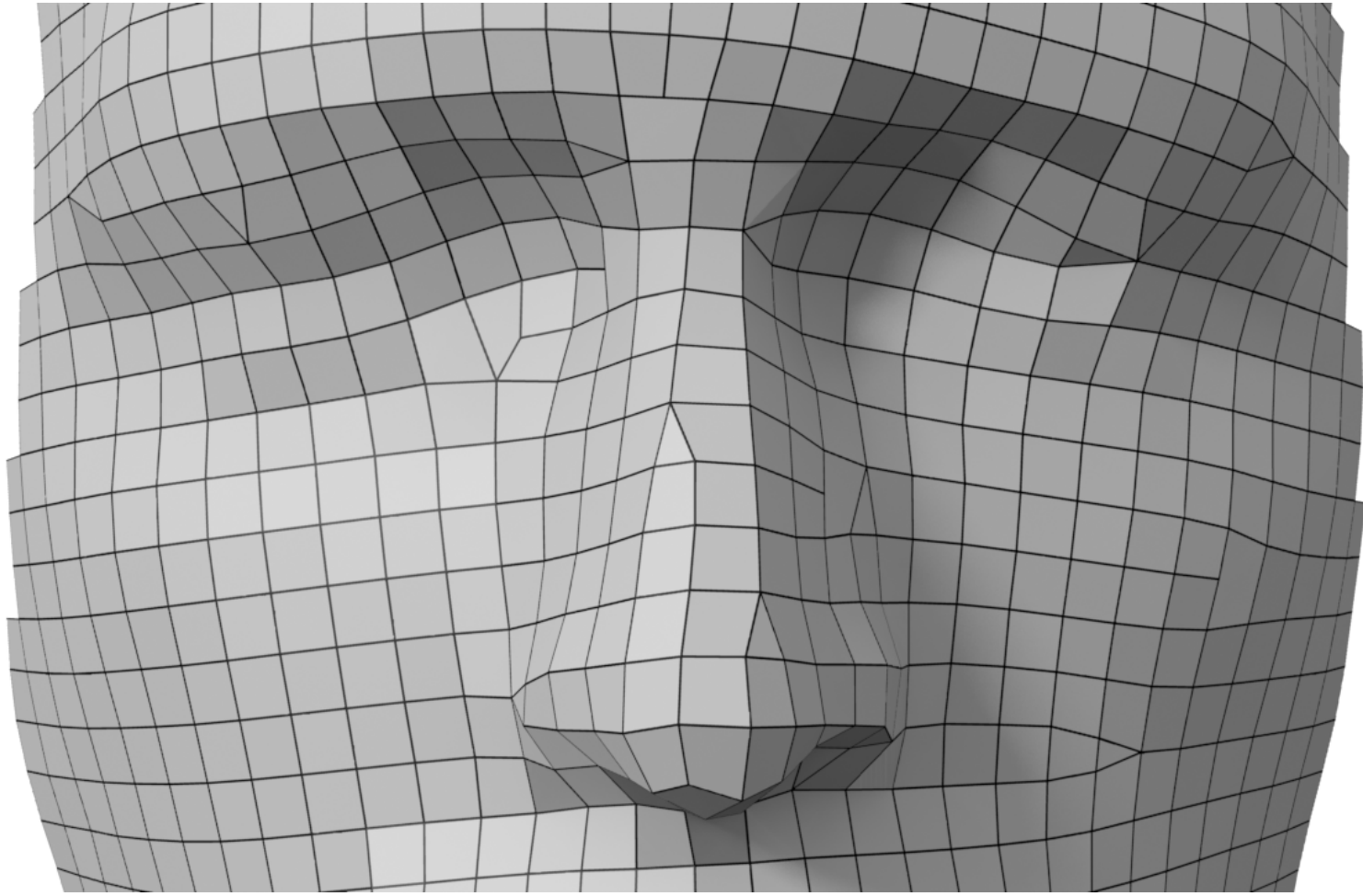
After 1 iteration: All faces are quads

After 2 iterations: All **new** vertices are degree-4

Limit surface is C^2 , except at "extraordinary vertices" (degree $\neq 4$), where it is C^1

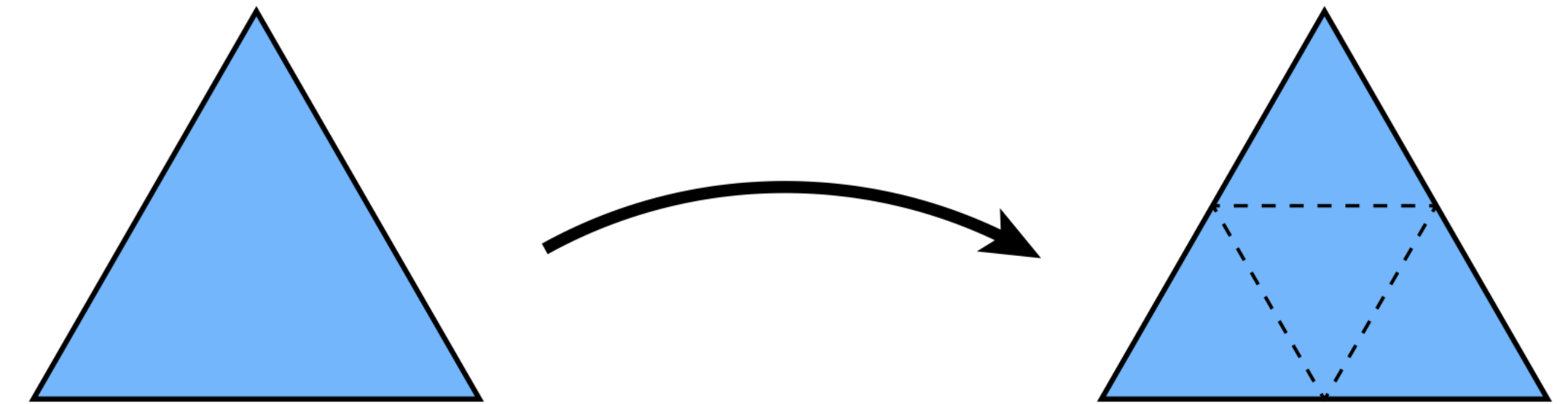


Examples

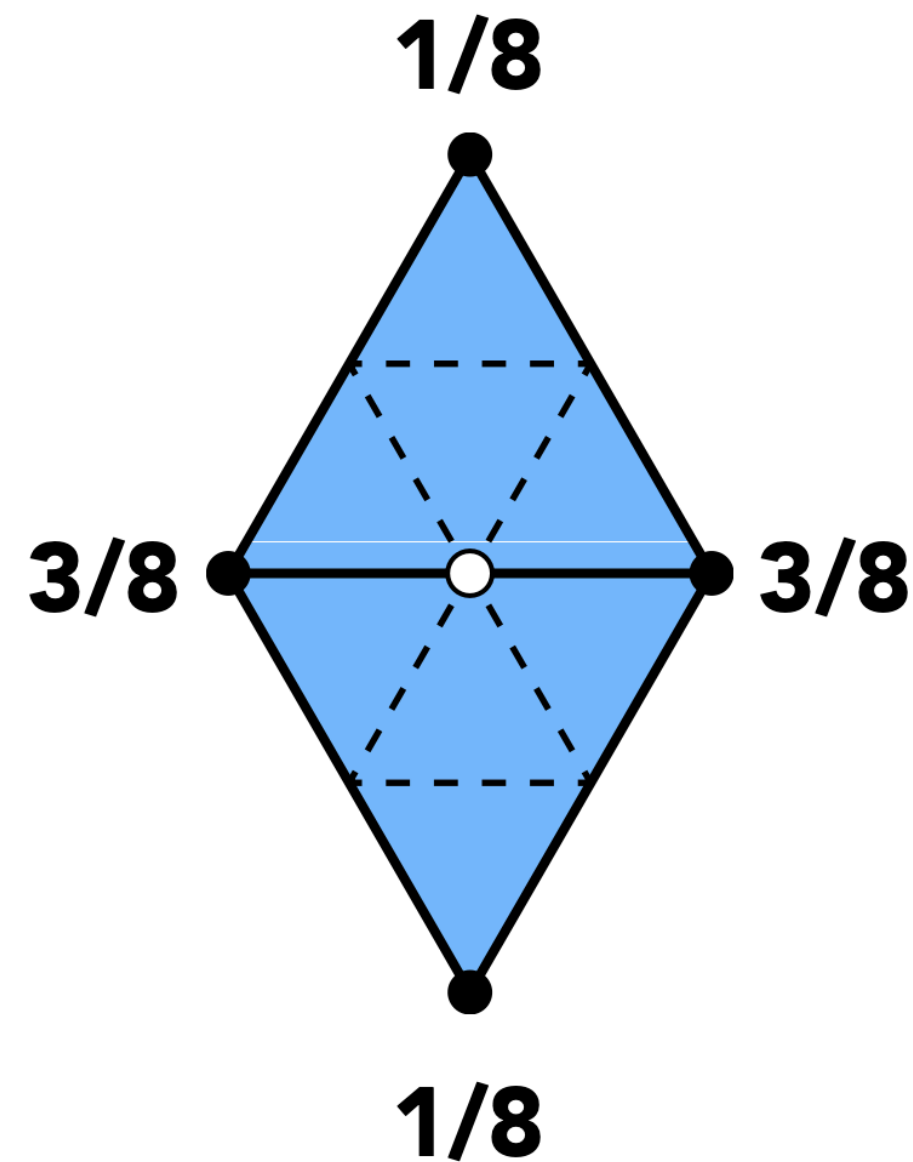


Loop Subdivision

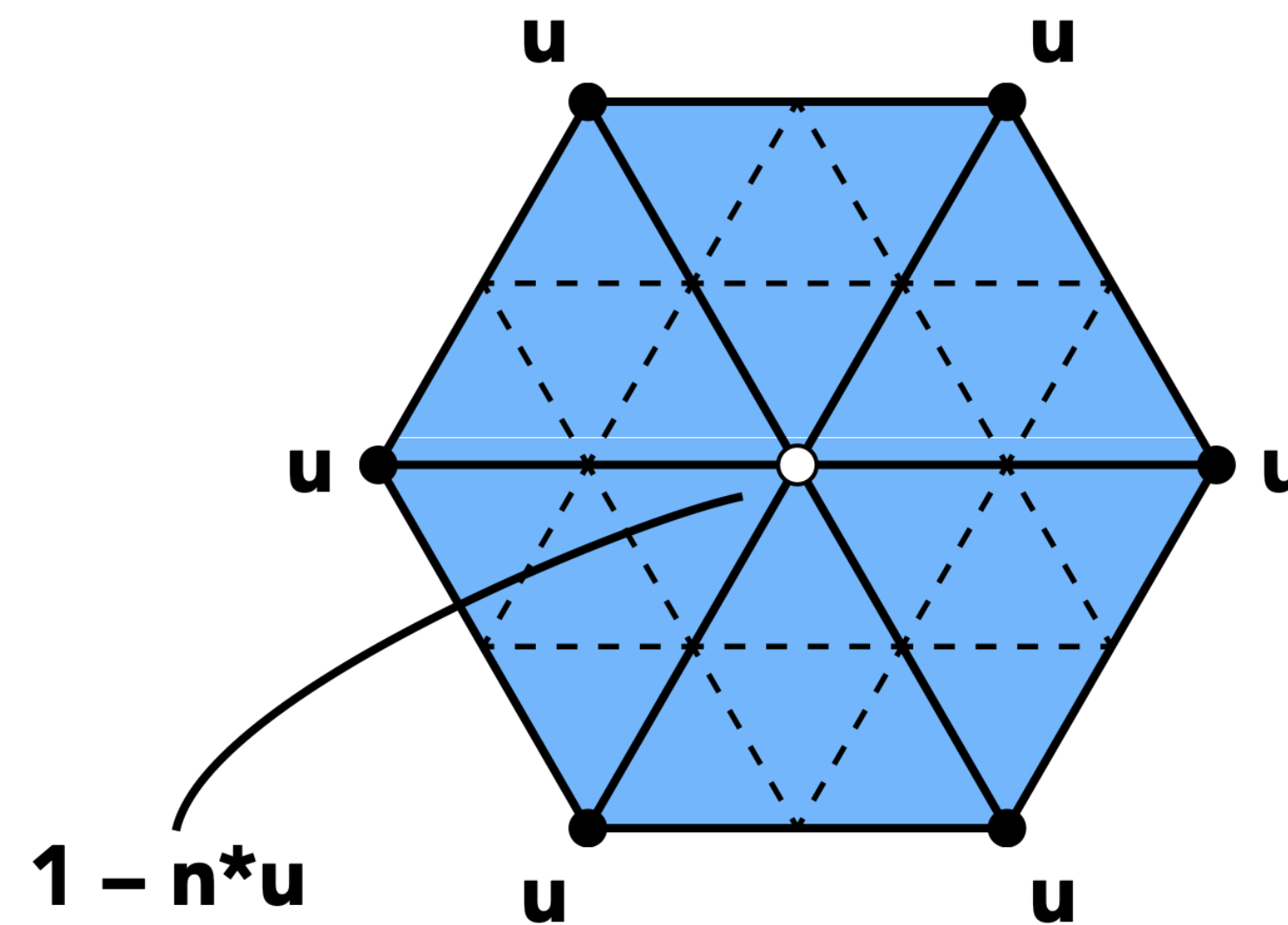
- Subdivide each triangle into 4
 ➔ by introducing edge vertices



Edge Vertex position



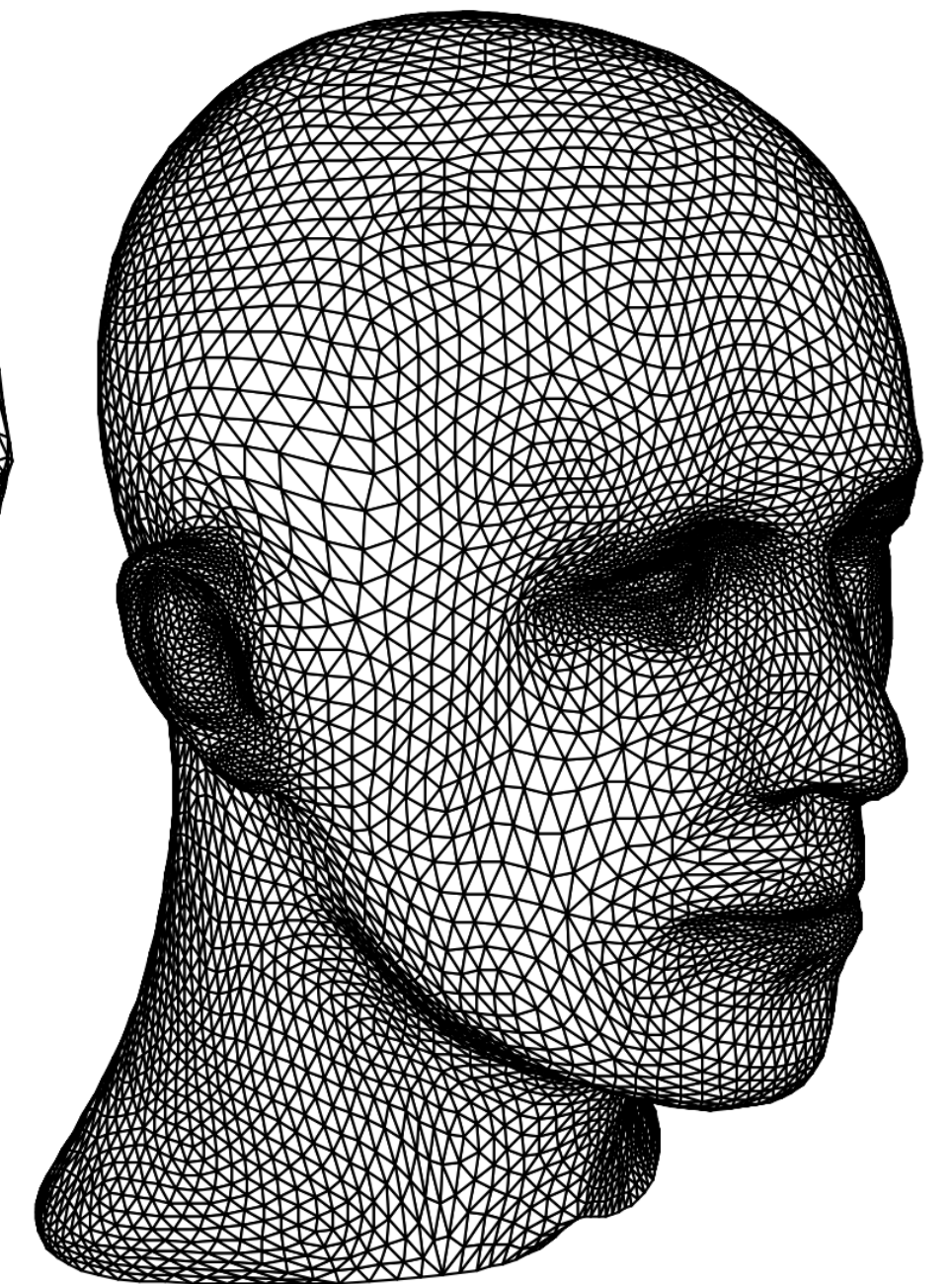
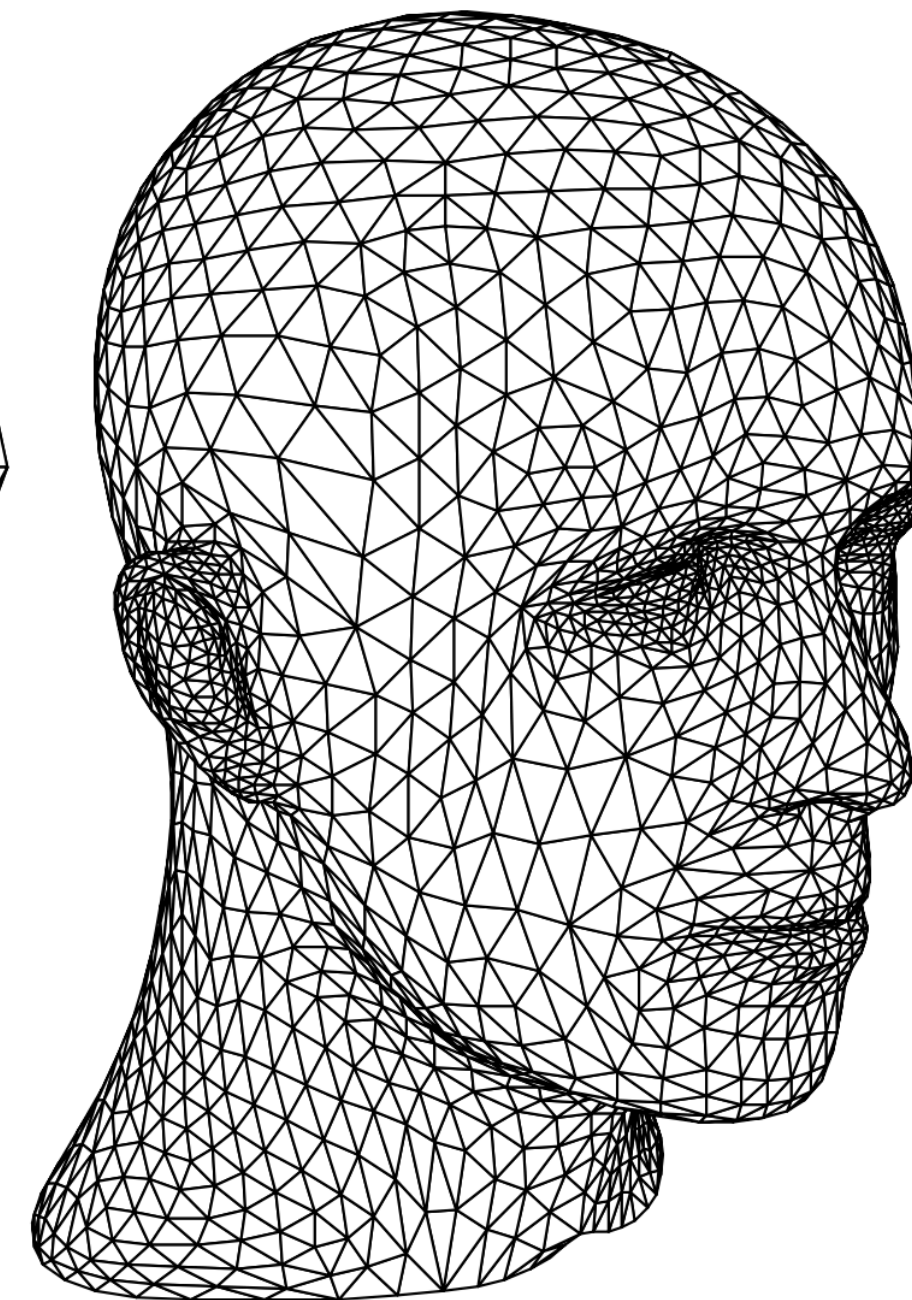
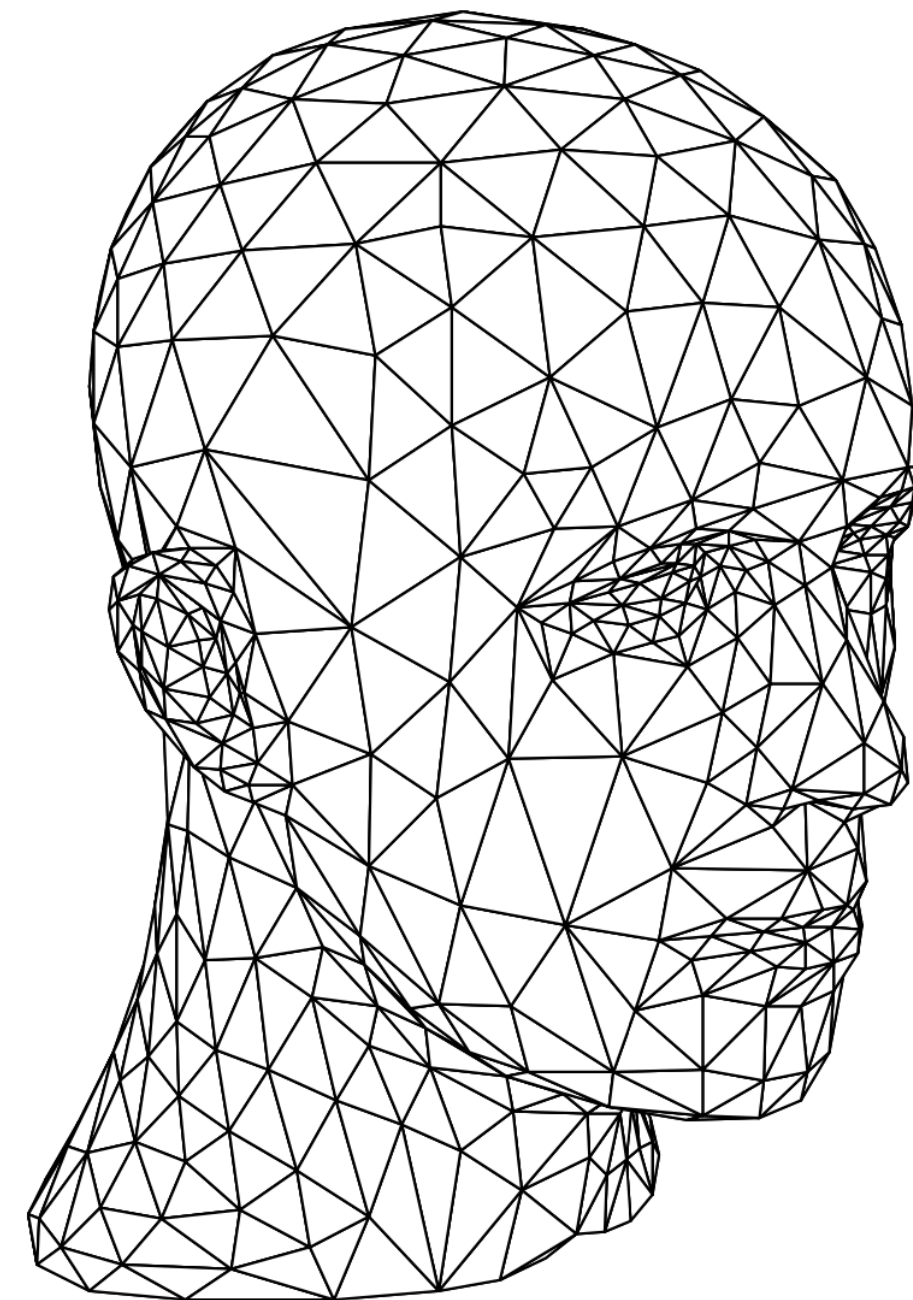
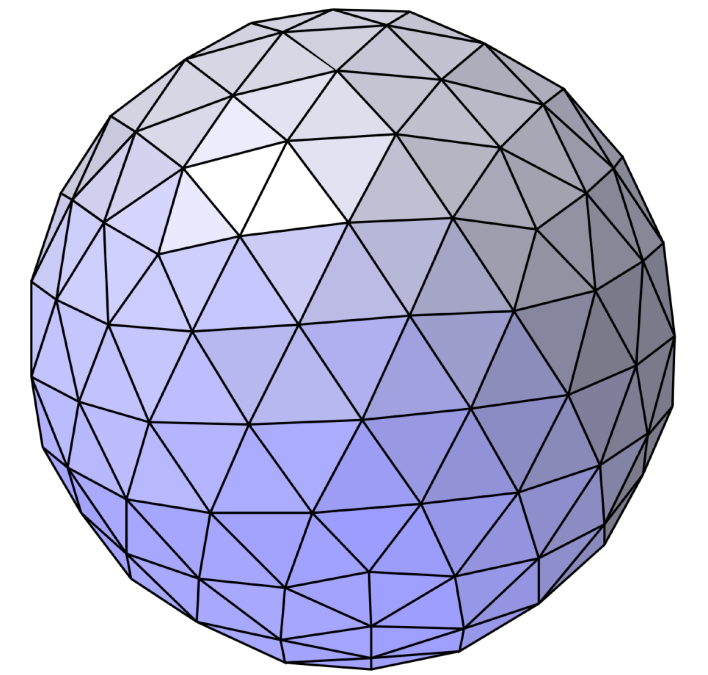
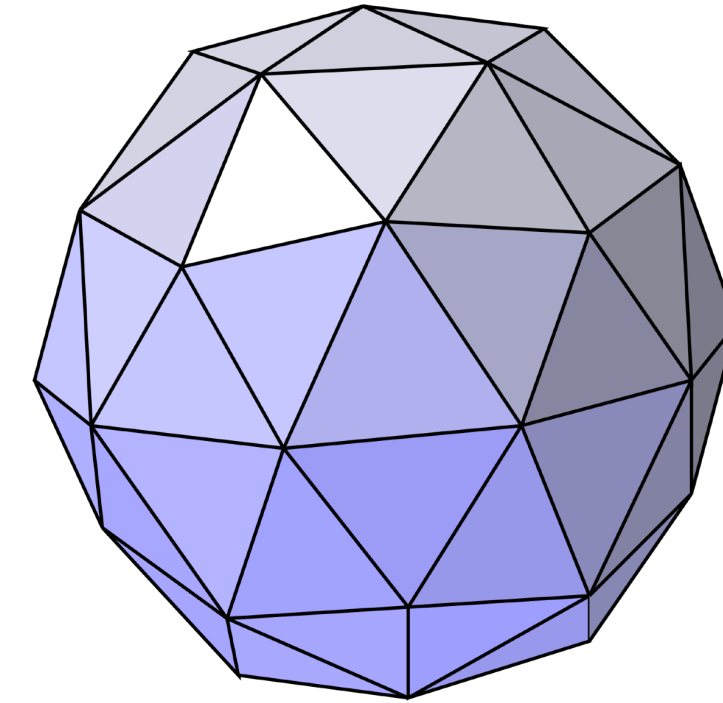
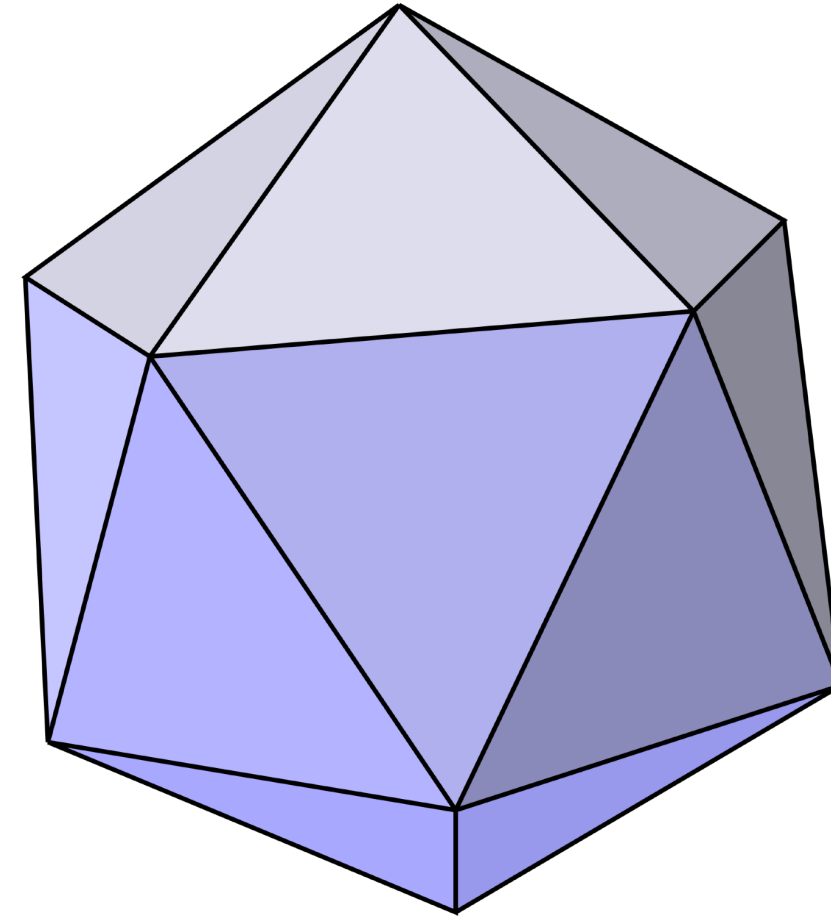
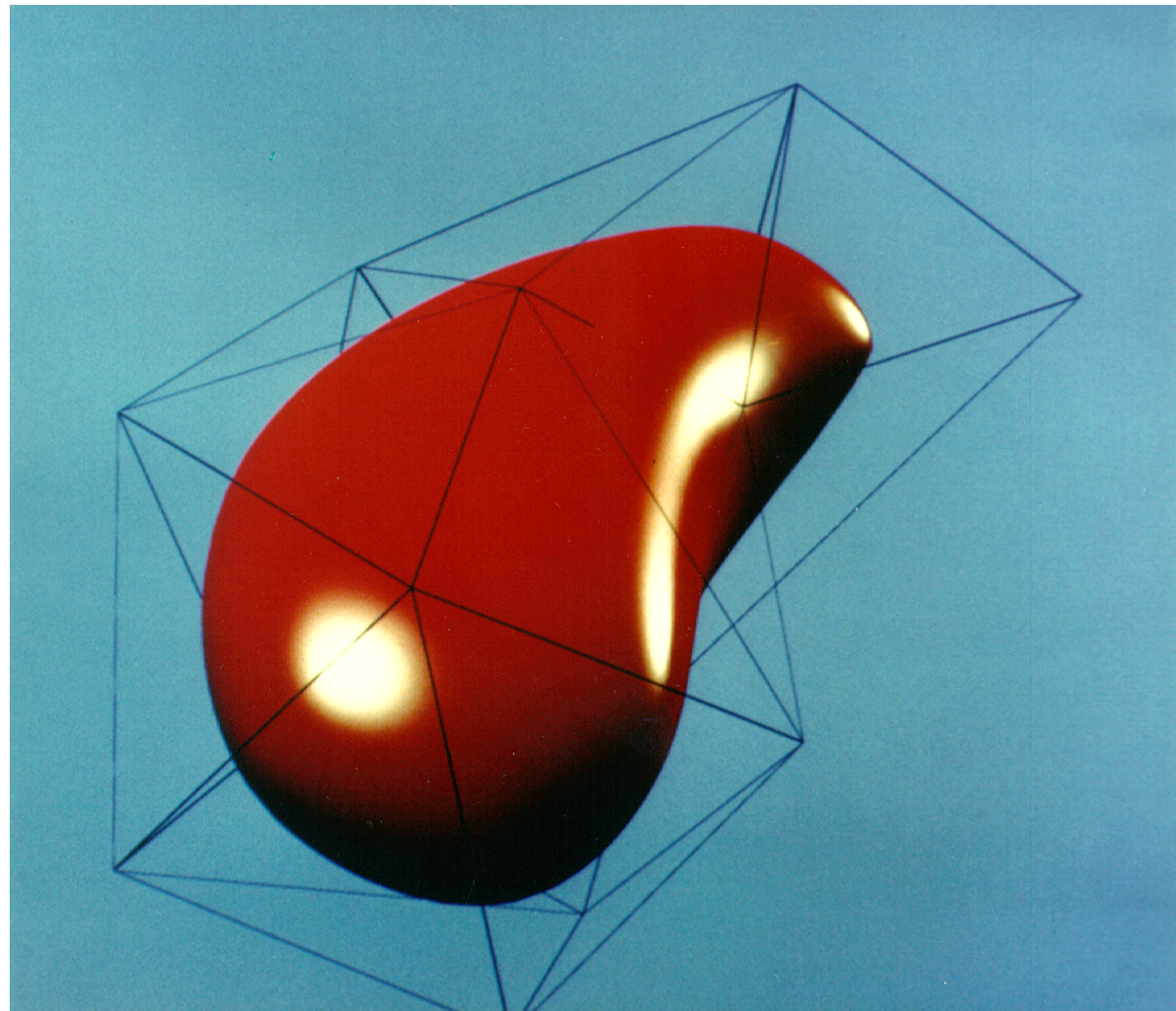
Update old vertex position



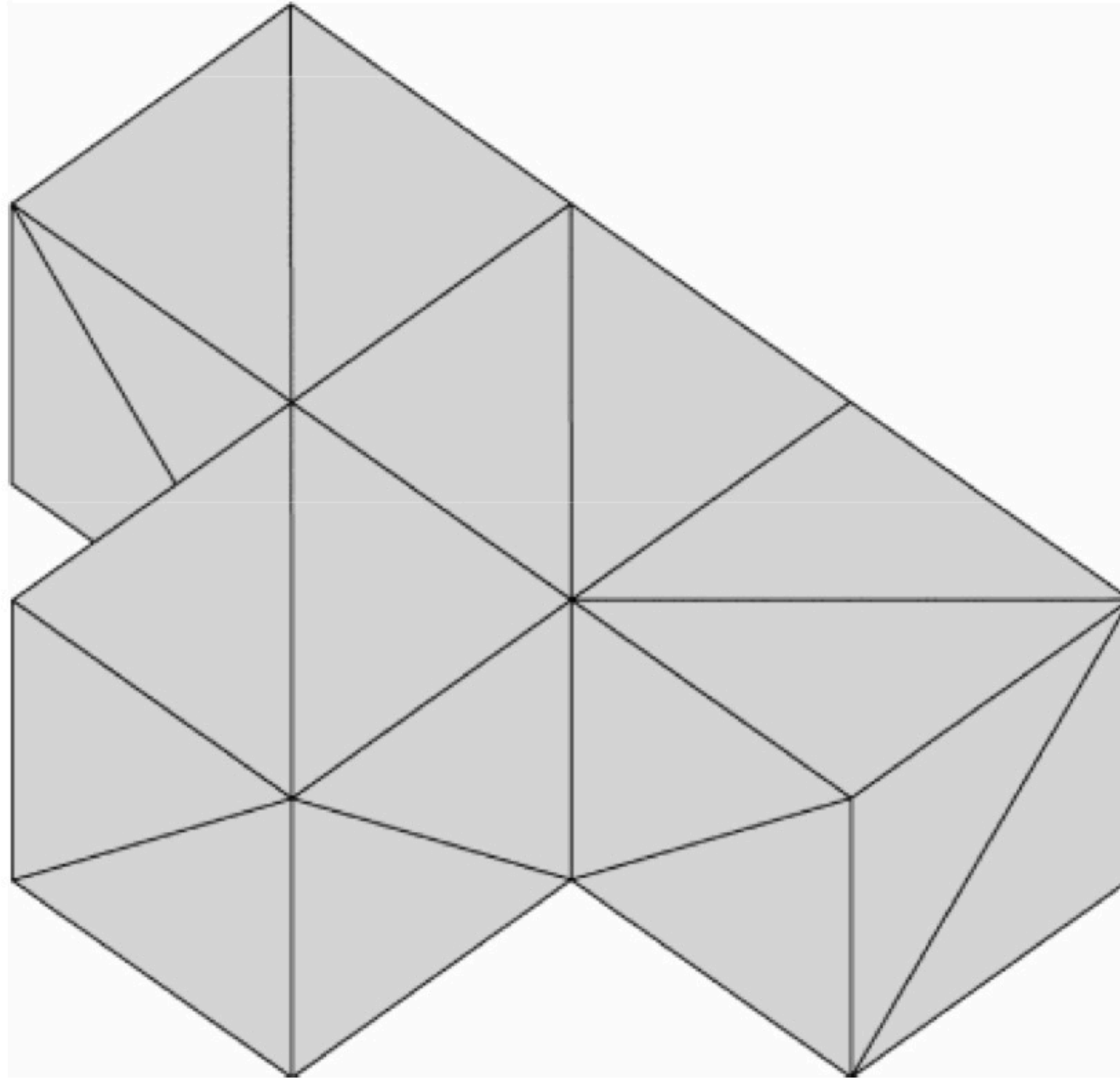
$$\text{where } u = \begin{cases} 1/16 & \text{if } n = 3, \\ 3/(8n) & \text{otherwise} \end{cases}$$

Limit surface is $\mathcal{C}^2$, except at
 "extraordinary vertices"
 (degree $\neq 6$), where it is $\mathcal{C}^1$

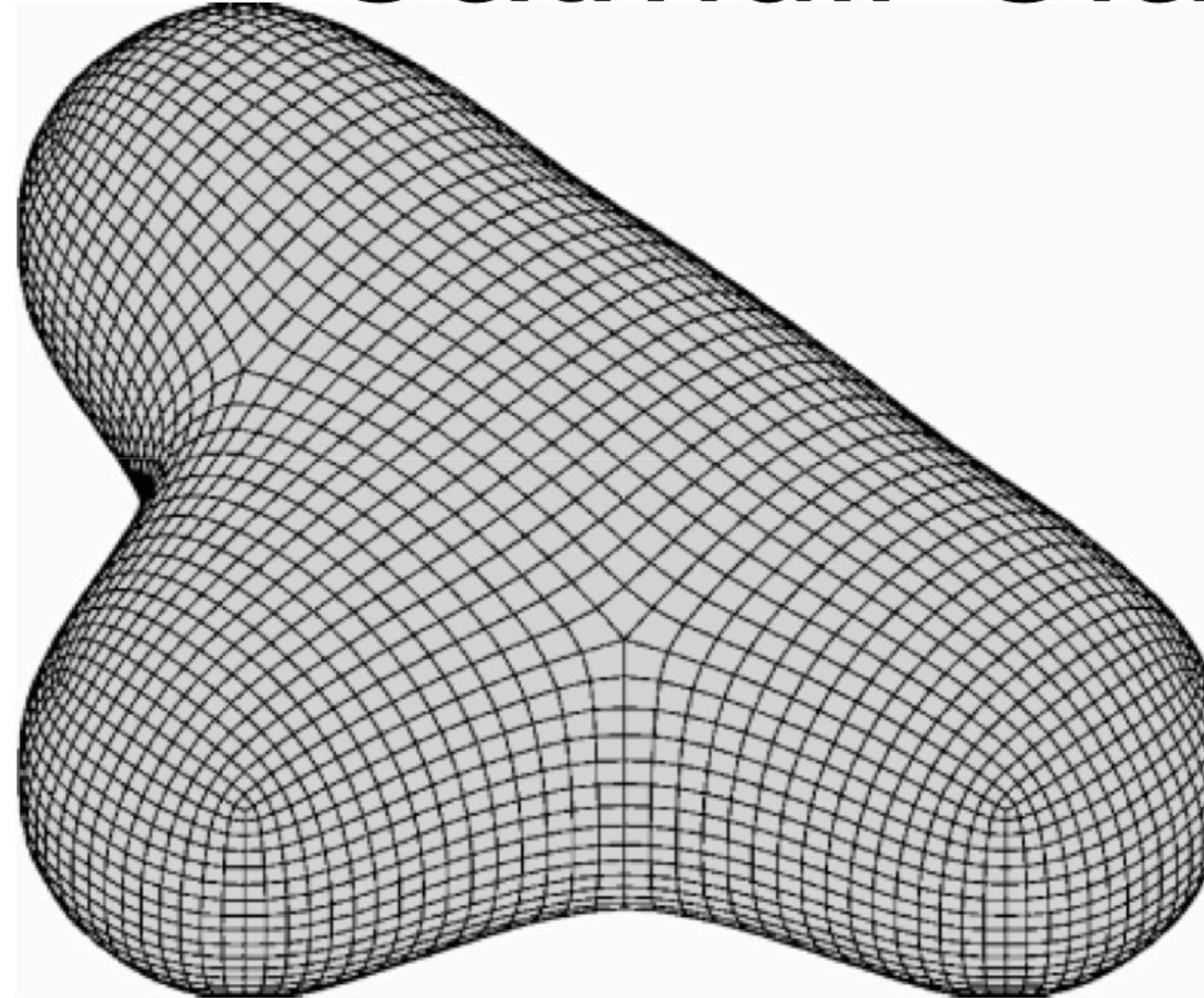
Examples



Comparison



Catmull-Clark



Loop

