UiO : University of Oslo

INF3320: Computer Graphics and Discrete Geometry

Parametric Curves and Surfaces

Christopher Dyken, Michael S. Floater and Martin Reimers
Corrections and additions by André R. Brodtkorb

November 5, 2014

Parametric Curves and Surfaces

Akenine-Möller, Haines and Hoffmann:

- Parametric curves (Chapter 13.1)
- Bézier curves (Chapter 13.1.1)
- Parametric curved surfaces (Chapter 13.2)
- Bézier patches (Chapter 13.2.1)

Red Book

- Evaluators and NURBS (Chapter 12)

Curved geometry

Curved geometry

Not all things are flat, but polygons are - need many of them

Curved geometry

Not all things are flat, but polygons are - need many of them

Higher order descriptions of geometry:

- *curved* (non-flat) geometry
- described by *mathematical expressions*
- typically *few parameters*
- *compact format* (storage, transmission, transformation)
- *scalable* wrt resolution (vector vs raster images)
- easier to manipulate, animate, ...
- ... but must be *tesselated* for visualization

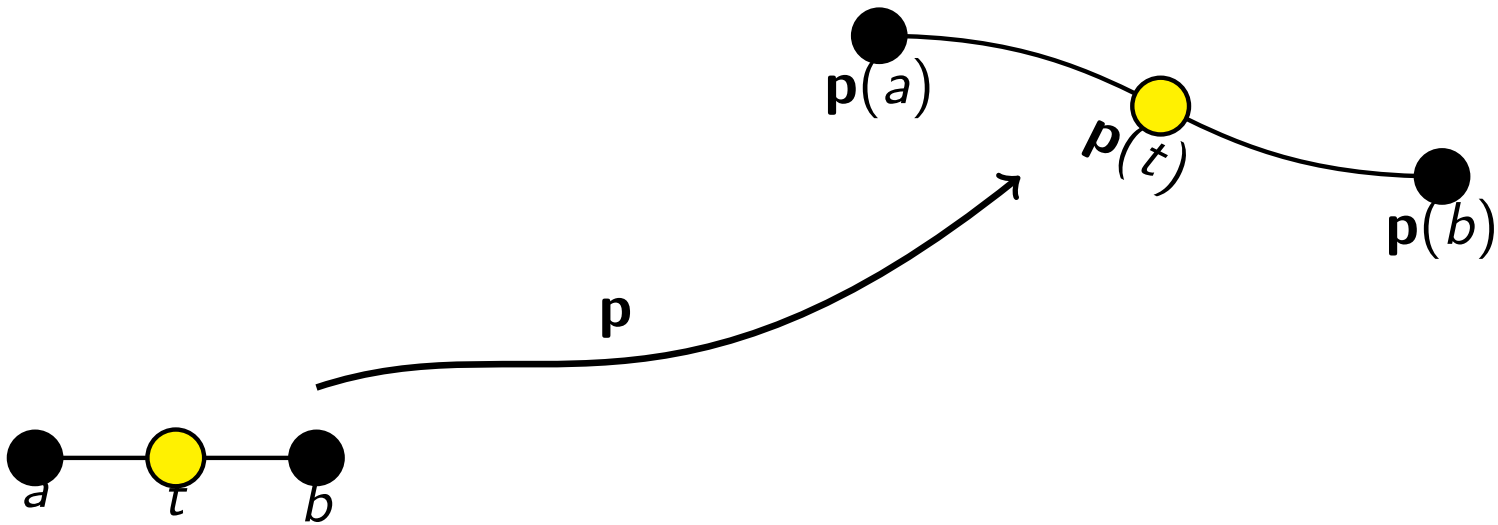
Parametric curves

Parametric Curves

A *parametric curve* is defined as a map

$$\mathbf{p} : [a, b] \to \mathbb{R}^n,$$

where usually $n = 2$ or $n = 3$.

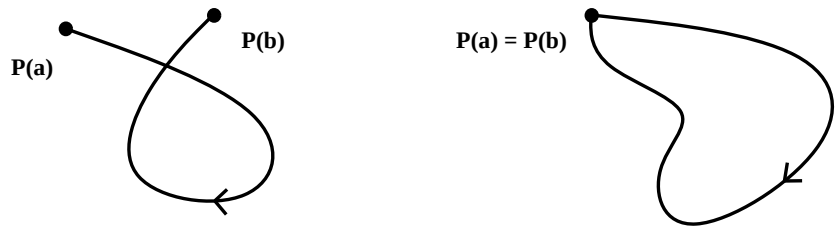


p maps each *parameter value* t between a and b to a corresponding point $\mathbf{p}(t) = (\mathbf{x}(t), \mathbf{y}(t))$.

The range of valid parameter values $[a, b]$ is called the *domain*.

A parametric curve has a *direction* corresponding to increasing t, beginning at $\mathbf{p}(a)$ and ending at $\mathbf{p}(b)$.

If $\mathbf{p}(t_1) = \mathbf{p}(t_2)$ for two distinct parameter values t_1 and t_2, we say that the curve has a *self-intersection*, otherwise it is *simple*



In the special case that $\mathbf{p}(a) = \mathbf{p}(b)$, i.e., the curve intersects itself at the end-points, the curve is *closed*.

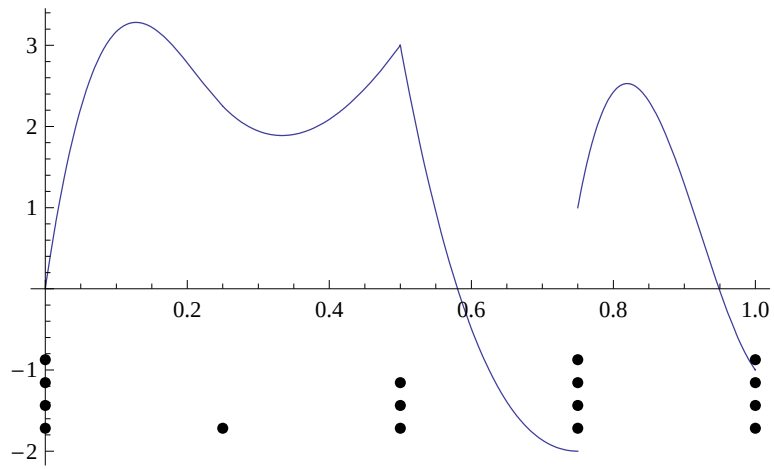
On the other hand, if $\mathbf{p}(a) \neq \mathbf{p}(b)$, we say that the curve is *open*.

Continuity

Normally, we assume that the curve P is at least *continuous*,

$$\lim_{\epsilon \to 0} ||\mathbf{p}(t+\epsilon) - \mathbf{p}(t)|| = 0,$$

for all t, i.e. $\mathbf{p}(t+\epsilon)$ approaches $\mathbf{p}(t)$ as ϵ approaches 0



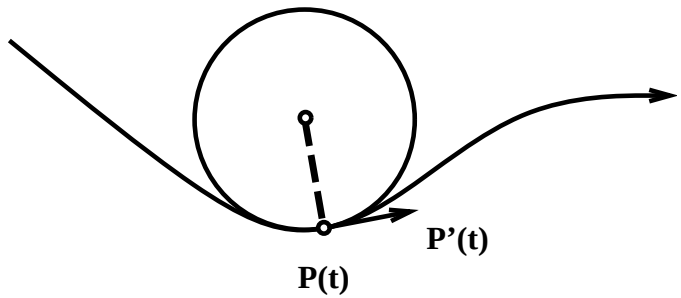
$\mathbf{p}$ is C^r *continuous/smooth* if $\frac{d^i \mathbf{p}}{dt^i}$ is continuous for $i = 0, ..., r$

Tangent and normal

If the curve $\mathbf{p} : [a, b] \to \mathbb{R}^n$ is differentiable at a point $\mathbf{p}(t)$, we write its (vector) derivative as

$$\frac{d}{dt}\mathbf{p}(t) \qquad \text{or} \qquad \mathbf{p}'(t).$$

The curve is *regular* if $\mathbf{p}'(t) \neq \mathbf{0}$ at every parameter value t. We call $T(t) = \mathbf{p}'(t)$ the *tangent* at $\mathbf{p}(t)$ (direction and speed)



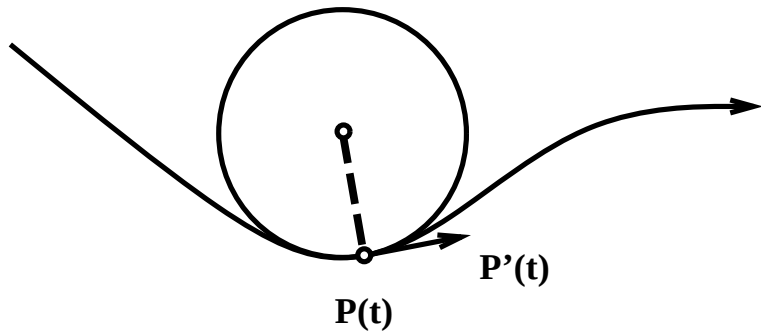
The *normal* $N(t)$ of a plane curve is orthogonal to $T(t)$, in the direction the curve is turning

Curvature

The *curvature* $\kappa(t)$ measures how fast the tangent (or normal) turns in $\mathbf{p}(t)$ for a C^2 curve

In a regular point $\mathbf{p}(t)$:

$$\kappa(t) = \frac{\|\mathbf{p}'(t) \times \mathbf{p}''(t)\|}{\|\mathbf{p}'(t)\|^3},$$



where $\times$ denotes the *cross product*,

$$\mathbf{u} \times \mathbf{v} = (u_y v_z - u_z v_y, u_z v_x - u_x v_z, u_x v_y - u_y v_x).$$

$\kappa(t) = 1/\rho(t)$, where $\rho(t)$ is the radius of the *osculating circle*.

Bézier curves

Polynomial curves

A polynomial *function* of degree d can be defined as

$$c(t) = \sum_{i=0}^{d} c_i t^i = c_0 + c_1 t + c_2 t^2 + \ldots + c_d t^d,$$

for real coefficients c_i. Called *power form* if using $t, t, t^2, \ldots$

A polynomial *curve* of degree d can be defined as

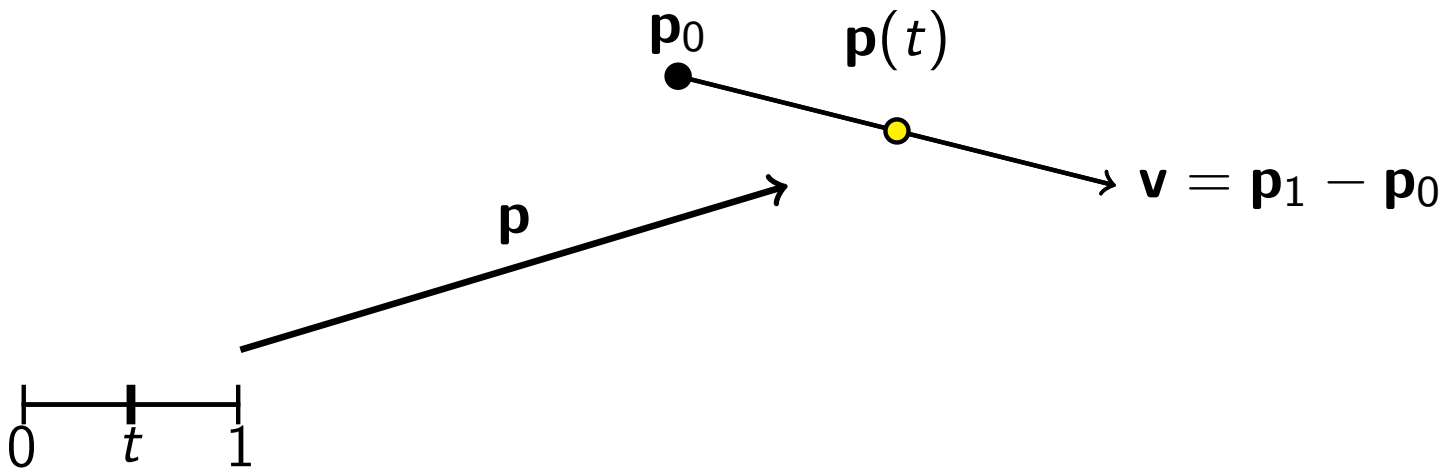
$$\mathbf{p}(t) = (\mathbf{x}(t), \mathbf{y}(t), \ldots) = \sum_{i=0}^{d} \mathbf{p}_i t^i = \mathbf{p}_0 + \mathbf{p}_1 t + \mathbf{p}_2 t^2 + \ldots + \mathbf{p}_d t^d,$$

for coefficients $\mathbf{p}_i \in \mathbb{R}^n$.

Example: line segment in power form

Example: A line segment $[\mathbf{p}_0, \mathbf{p}_1]$ can be written

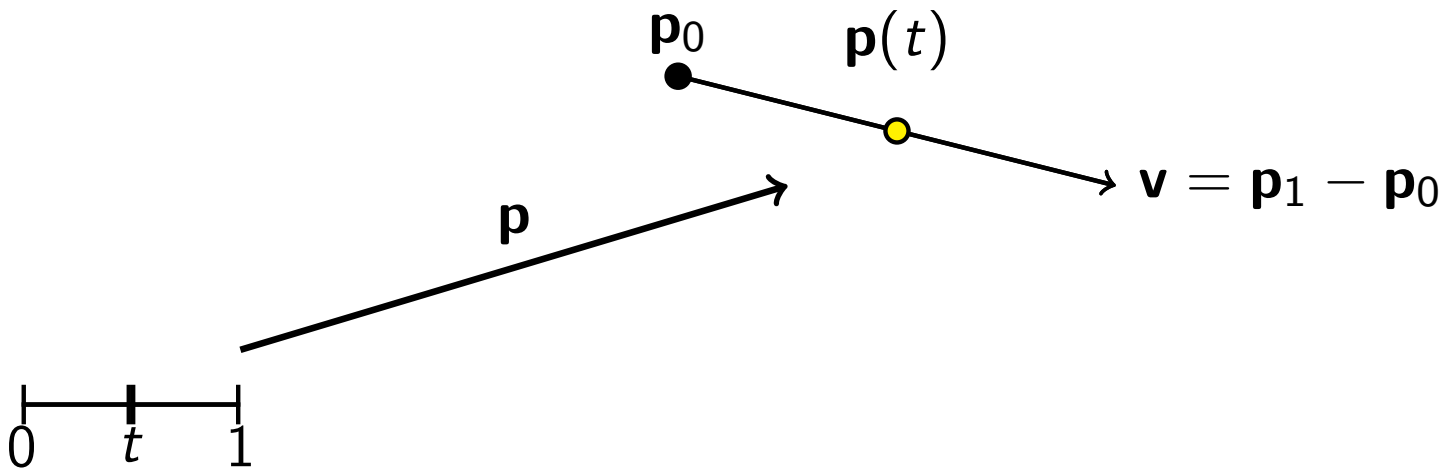
$$\mathbf{p}(t) = \mathbf{p}_0 + (\mathbf{p}_1 - \mathbf{p}_0)t \quad \text{for } t \in [0, 1]$$



Example: line segment in power form

Example: A line segment $[\mathbf{p}_0, \mathbf{p}_1]$ can be written

$$\mathbf{p}(t) = \mathbf{p}_0 + (\mathbf{p}_1 - \mathbf{p}_0)t \quad \text{for } t \in [0, 1]$$



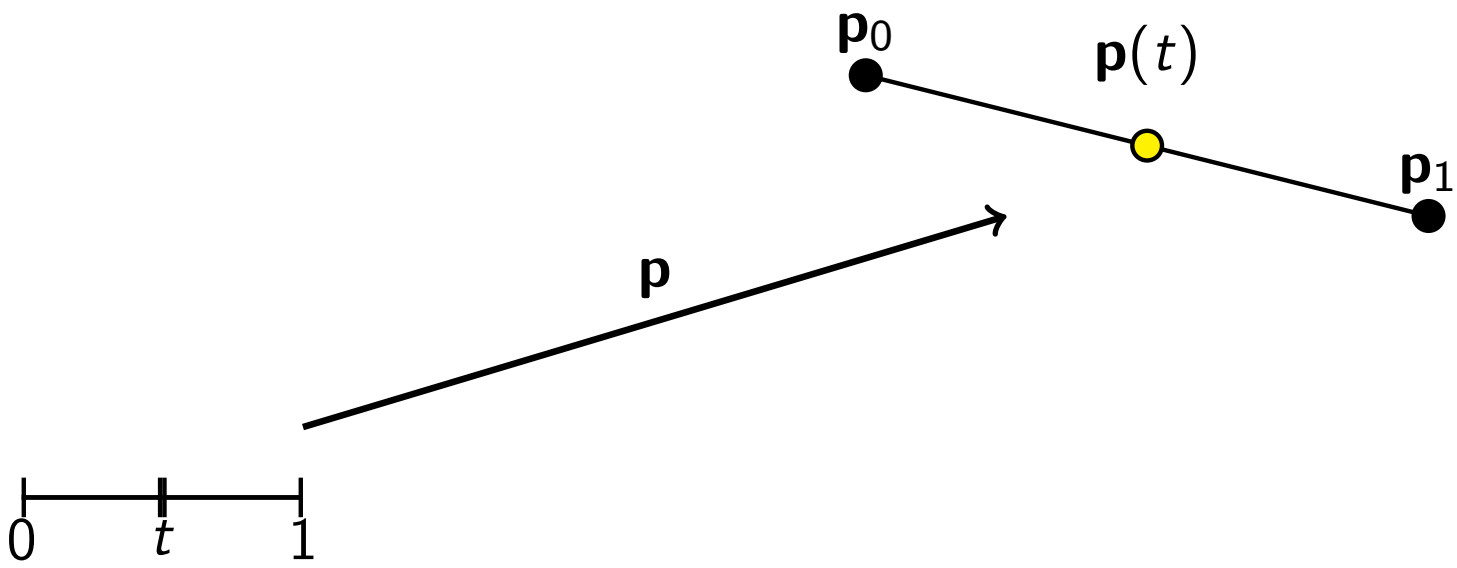
There are better ways to represent polynomial curves...

$$\mathbf{p}(t) = \mathbf{p}_0 + (\mathbf{p}_1 - \mathbf{p}_0)t = (1 - t)\mathbf{p}_0 + t\mathbf{p}_1$$

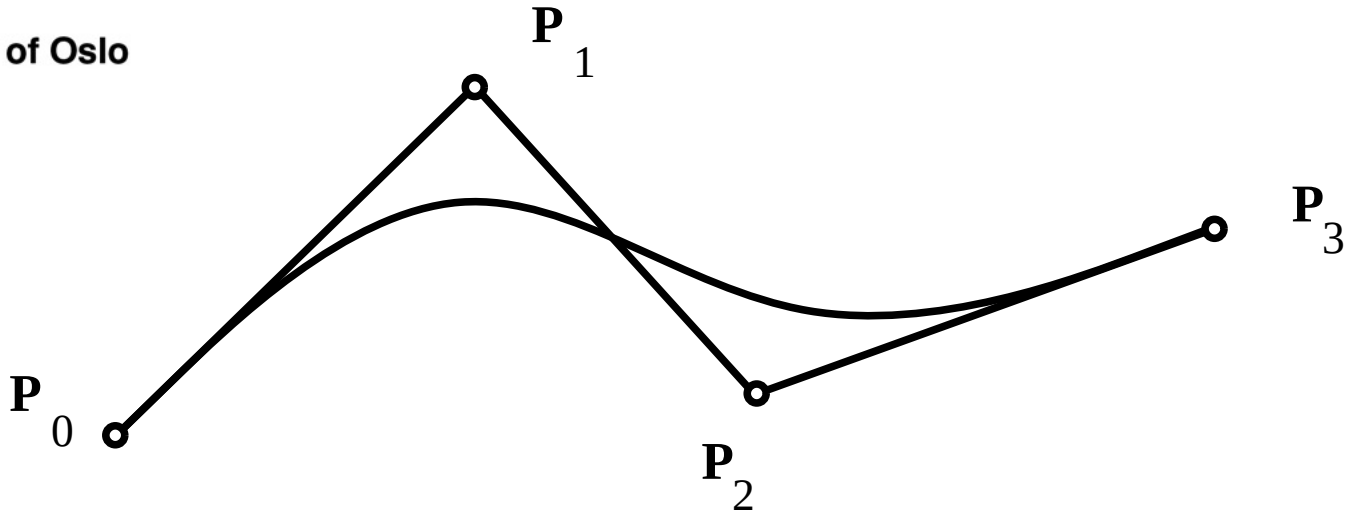
Example: line segment on Bezier form

Example: A line segment $[\mathbf{p}_0, \mathbf{p}_1]$ can be written in *Bézier form*

$$\mathbf{p}(t) = (1-t)\mathbf{p}_0 + t\mathbf{p}_1 \quad \text{for } t \in [0,1]$$



Bezier segment: $\mathbf{p}(t)$ is a *convex combination* of the endpoints



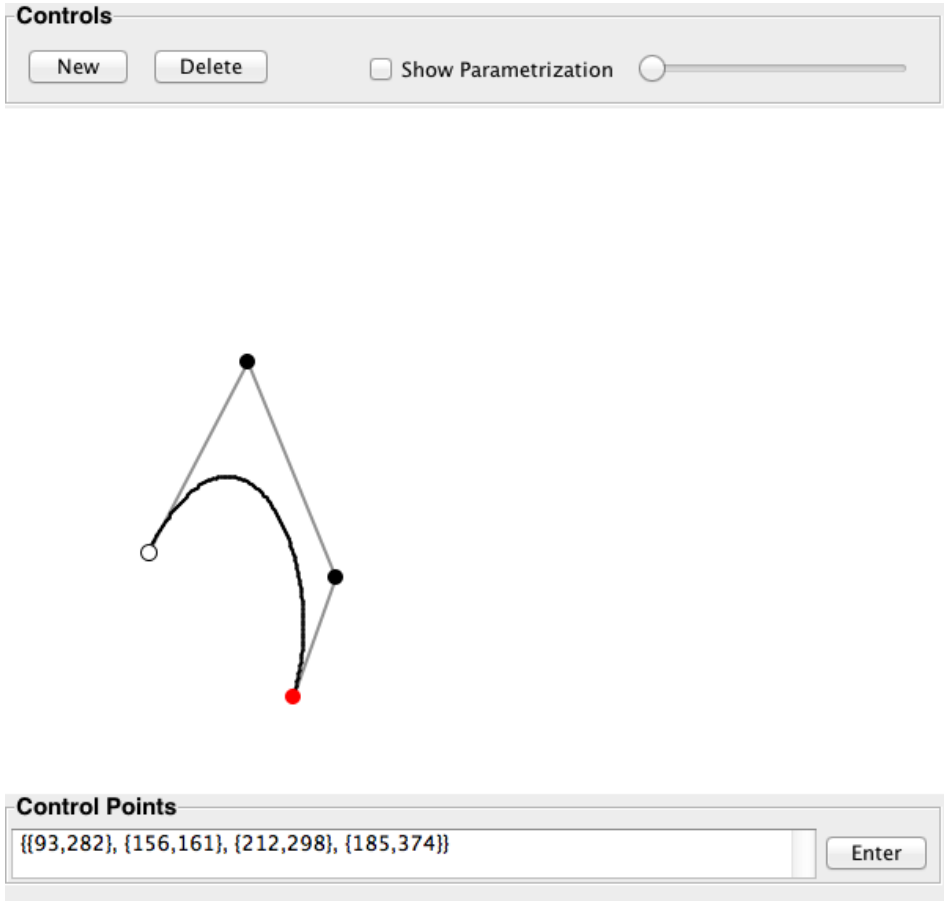
In general, a Bézier curve of degree d is written

$$\mathbf{p}(t) = (1-t)^d \mathbf{p}_0 + dt(1-t)^{d-1}\mathbf{p}_1 + \ldots + t^d \mathbf{p}_d$$
$$= \sum_{i=0}^{d} \underbrace{\binom{d}{i} t^i (1-t)^{d-i}}_{B_{i,d}(t)} \mathbf{p}_i = \sum_{i=0}^{d} \mathbf{p}_i B_{i,d}(t).$$

where $\binom{d}{i} = \frac{d!}{i!(d-i)!}$ are the *Binomial coefficients* and

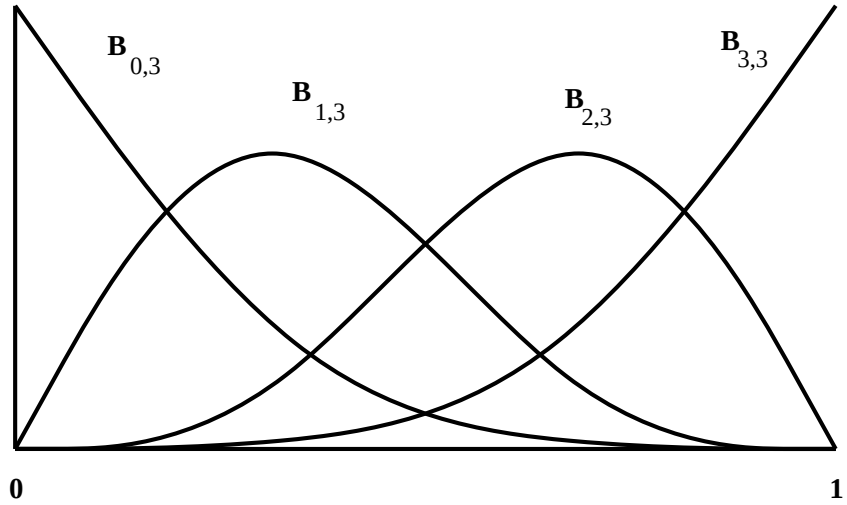
- $B_{i,d}$ are the *Bernstein polynomials* of degree d
- The vertices $(\mathbf{p}_i)$ are *control points* of $\mathbf{p}$
- The polygon $[\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, ...]$ is the *control polygon* of $\mathbf{p}$

Example: Bezier curve applet



http://www.math.psu.edu/dlittle/java/parametricequations/beziercurves/index.html

The Bernstein polynomials

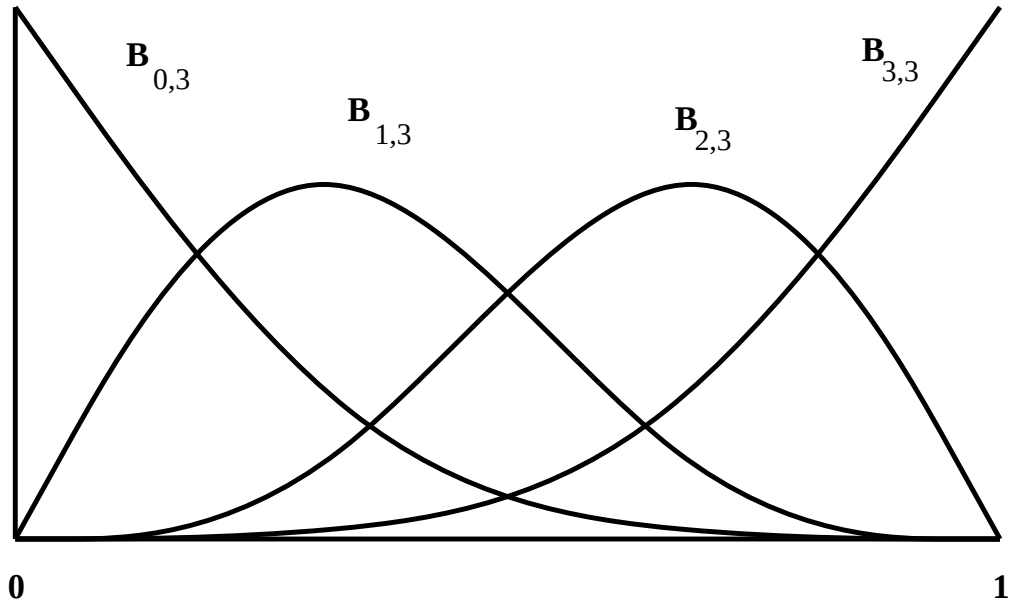


$$(1-t)^3, \qquad 3t^1(1-t)^2, \qquad 3t^2(1-t)^1, \qquad t^3$$

The $(B_{i,d})_i$ form a basis for the polynomials of degree $\leq d$.

- Alternative to power basis $(t^i) = 1, t, t^2, \ldots$
- $\sum_i B_{i,d}(t) = 1$
- $B_{i,d}(t) \geq 0$ for $t \in [0,1]$
- Simple to generalize to $[a,b]$
- $\min c_i \leq \sum_i c_i B_{i,d}(t) \leq \max c_i$ (Convex Hull Property)

Evaluation



There are two recursive approaches to evaluating Bézier curves.

$$\mathbf{p}(t) = \sum \mathbf{p}_i B_{i,d}(t)$$

1. Recursion on basis functions
2. Recursion on control points

Recursion on basis functions

Idea:

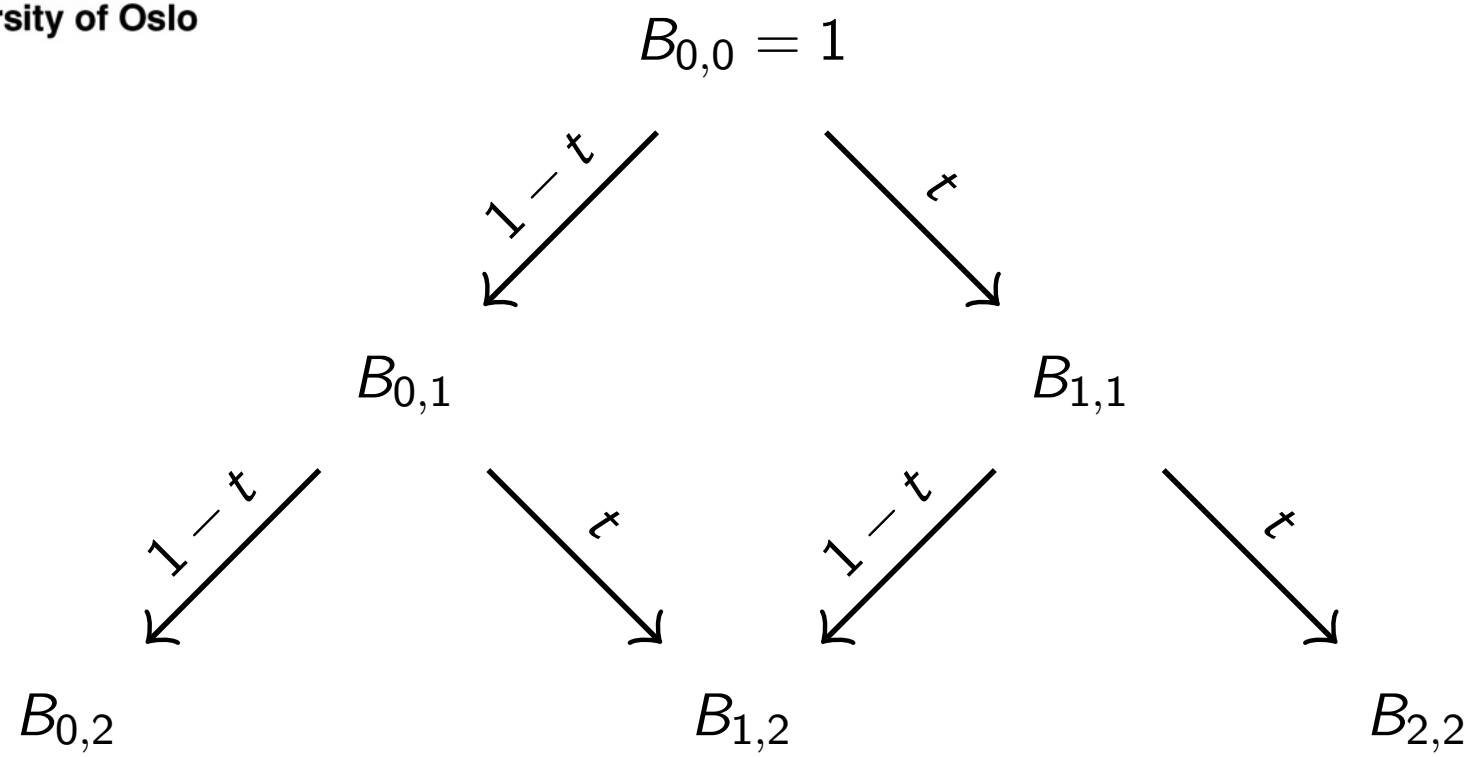
1. Evaluate each basis function, i.e. $B_{i,d}(t)$
2. Compute $\mathbf{p}(t) = \sum \mathbf{p}_i B_{i,d}(t)$

- We can evaluate the Bernstein functions recursively using

$$B_{i,d}(t) = (1-t)B_{i,d-1}(t) + tB_{i-1,d-1},$$

where we define $B_{0,0} = 1$ and $B_{i,d} = 0$ for $i < 0$ and $i > d$.

$\Longrightarrow$ Thus, a d-degree basis function can be written as a convex combination of two $d-1$ degree basis functions.



- First set $B_{0,0} = 1$,
- then $B_{0,1}$ and $B_{1,1}$ from multiplication with $B_{0,0}$,
- then $B_{0,2}, B_{1,2}, B_{2,2}$ from mult. with $B_{0,1}$ and $B_{1,1}$ ...

$\Longrightarrow$ This gives us a triangular scheme for $(B_{i,d})$

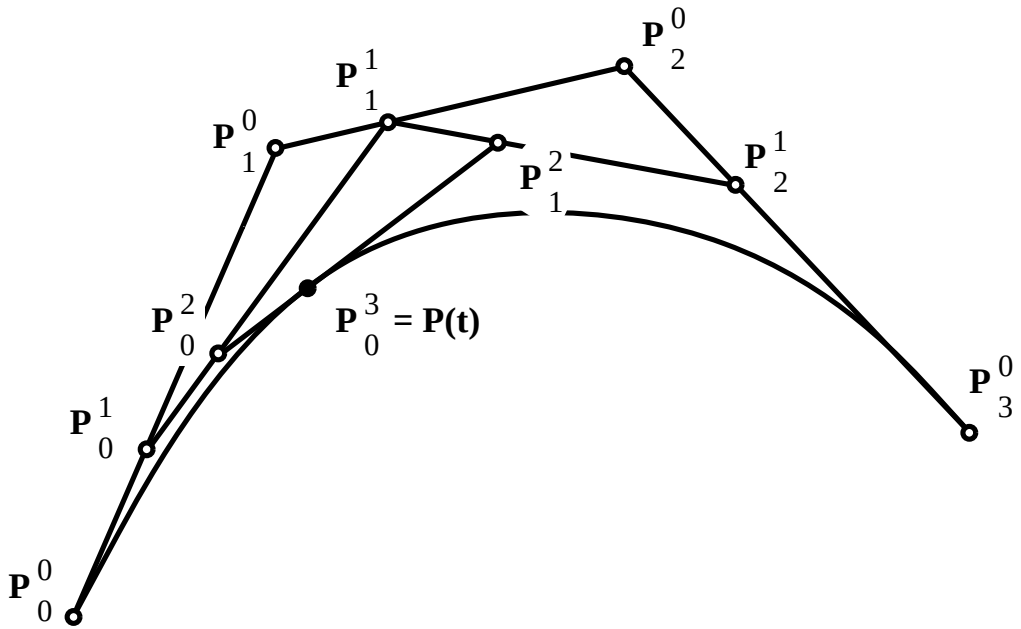
Recursion on the control points

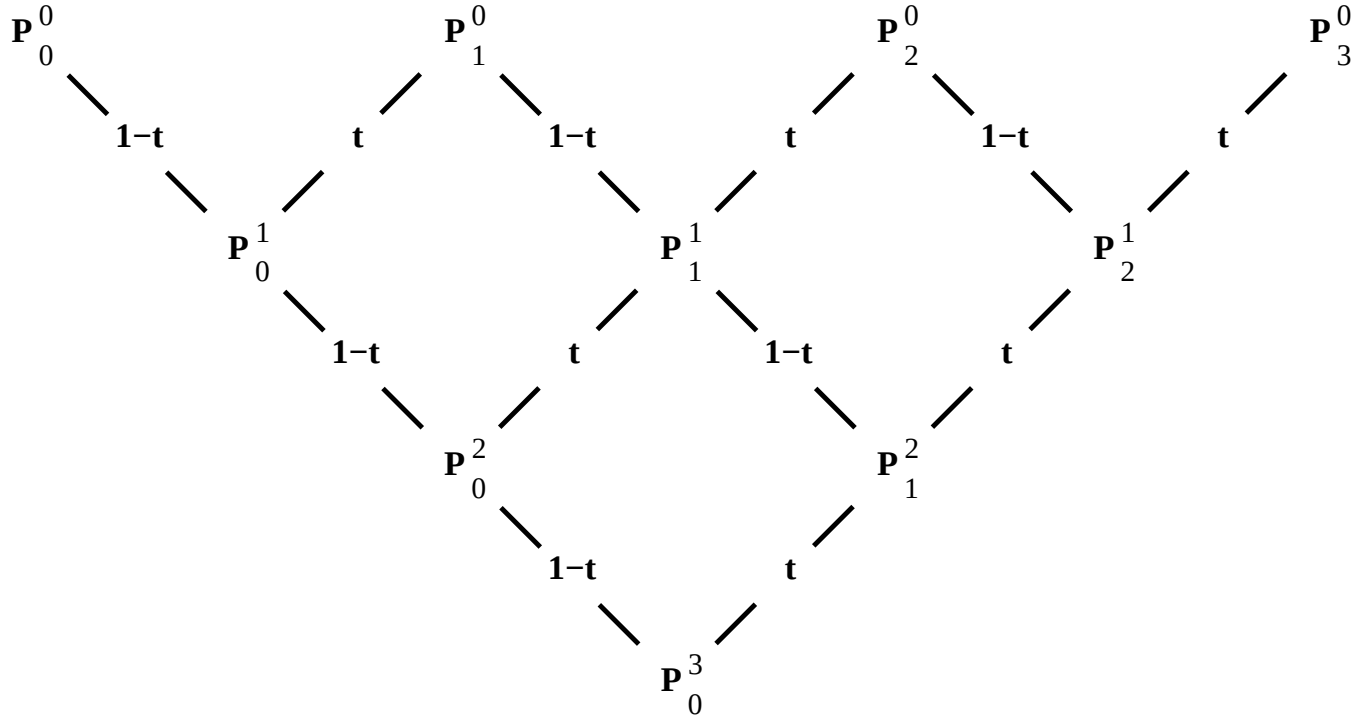
de Casteljau algorithm:

1. Relabel the control points, with $\mathbf{p}_i^0 = \mathbf{p}_i$
2. Apply repeated linear interpolation for $r = 1, ..., d$

$$\mathbf{p}_i^r = (1-t)\mathbf{p}_i^{r-1} + t\mathbf{p}_{i+1}^{r-1}, \qquad i = 0, ..., d-r$$

3. Then $\mathbf{p}(t) = \sum_{i=0}^{d} \mathbf{p}_i B_{i,d}(t) = \mathbf{p}_0^d$





Yields a triangular scheme for a point $\mathbf{p}(t)$ on the Bézier curve,

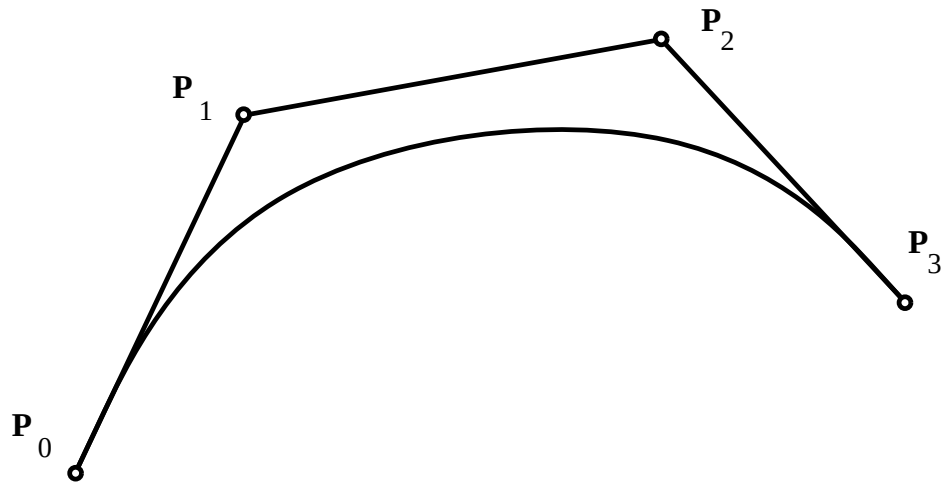
$$\mathbf{p}_0^d = \mathbf{p}(t) = \sum_{i=0}^{d} \mathbf{p}_i B_{i,d}(t)$$

Applet (enable show parametrization):
http://www.math.psu.edu/dlittle/java/parametricequations/beziercurves/index.html

Shape preservation

The shape of the Bézier curve $\mathbf{p}(t) = \sum_{i=0}^{d} \mathbf{p}_i B_{i,d}(t)$ mimics the shape of the control polygon.

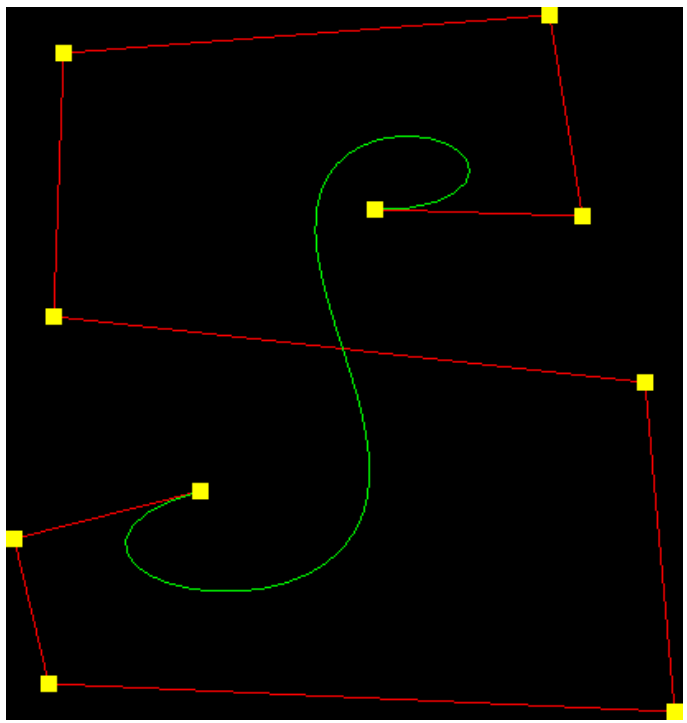
Bézier curves have the *convex hull* property: the curve is contained in the convex hull of its control points.



Bézier curves have the *variation diminishing property*:

> *The curve has no more intersections with a plane (or line in $\mathbb{R}^2$) than its control polygon.*

Thus a curve has *less shape variations* than the control polygon.



I.e, if the control polygon is convex then so is the curve itself.

Derivatives

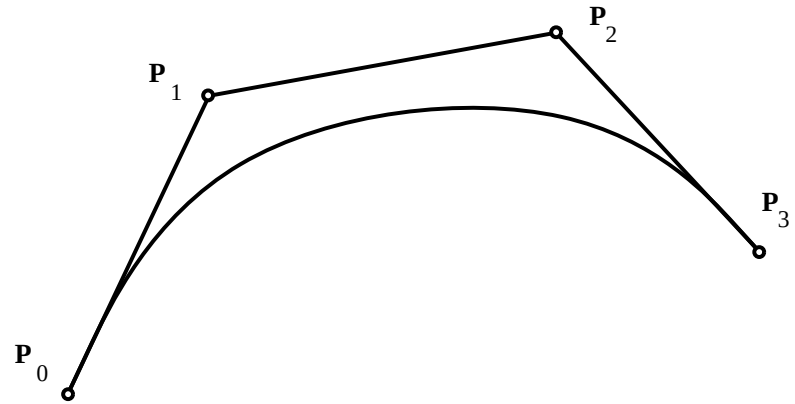
The derivative of the Bernstein polynomial $B_{i,d}$ is

$$B'_{i,d}(t) = d(B_{i-1,d-1}(t) - B_{i,d-1}(t)),$$

and it follows that $\mathbf{p}'(t)$ is a new "tangent" Bézier curve:

$$\begin{aligned}\mathbf{p}'(t) &= d\sum_{i=0}^{d}\mathbf{p}_i(B_{i-1,d-1}(t) - B_{i,d-1}(t)) \\ &= d\sum_{i=0}^{d-1}(\mathbf{p}_{i+1} - \mathbf{p}_i)B_{i,d-1}(t) = \sum_{i=0}^{d-1}\mathbf{p}'_i B_{i,d-1}(t).\end{aligned}$$

Control pts. of $\mathbf{p}'$ equals tangents of the control polygon $\mathbf{p}_0, \mathbf{p}_1, \ldots$



For higher derivatives we define the r-th forward difference by

$$\Delta \mathbf{p}_i = \mathbf{p}_{i+1} - \mathbf{p}_i$$

and

$$\Delta^r \mathbf{p}_i = \Delta^{r-1} \mathbf{p}_{i+1} - \Delta^{r-1} \mathbf{p}_i$$

so that

$$\Delta^2 \mathbf{p}_i = \mathbf{p}_{i+2} - 2\mathbf{p}_{i+1} + \mathbf{p}_i$$

and

$$\Delta^3 \mathbf{p}_i = \mathbf{p}_{i+3} - 3\mathbf{p}_{i+2} + 3\mathbf{p}_{i+1} - \mathbf{p}_i,$$

and so on.

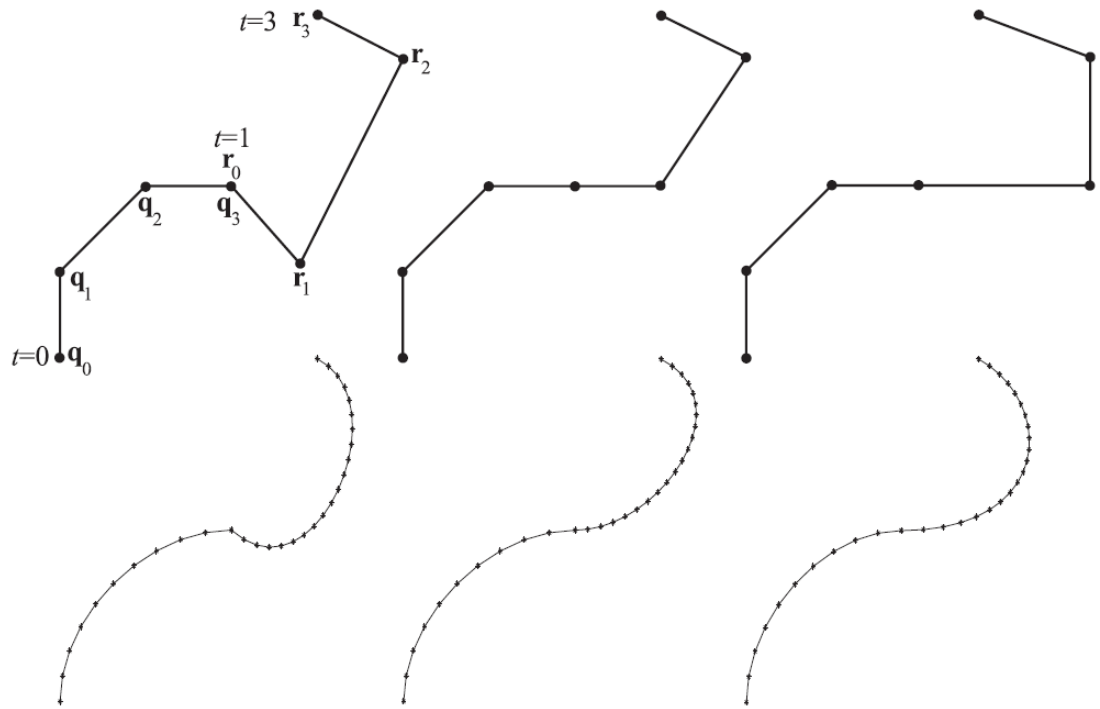
The r-th derivative of a Bézier curve is then

$$\mathbf{p}^{(r)}(t) = \frac{d!}{(d-r)!} \sum_{i=0}^{d-r} \Delta^r \mathbf{p}_i B_{i,d-r}(t).$$

Composite curves, Continuity

Polynomial curves are C^∞ smooth, but for complex shapes one must *join curves* with some *continuity*:

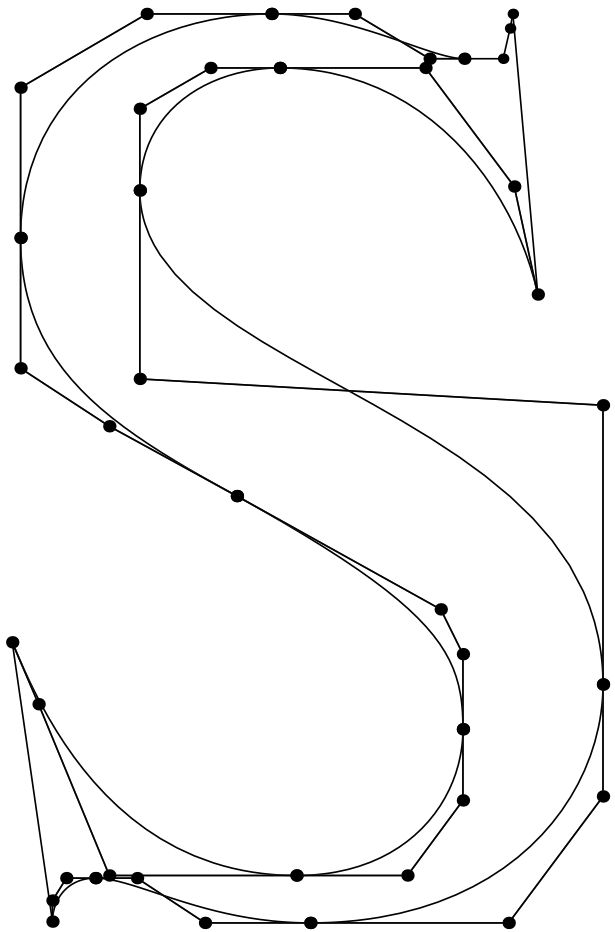
- Geometric (G^r) continuity (tangent/normal continuous)
- Parametric (C^r) continuity (derivatives continuous)



Composite curves require *continuity conditions* to be satisfied

Composite polynomial curves - Fonts

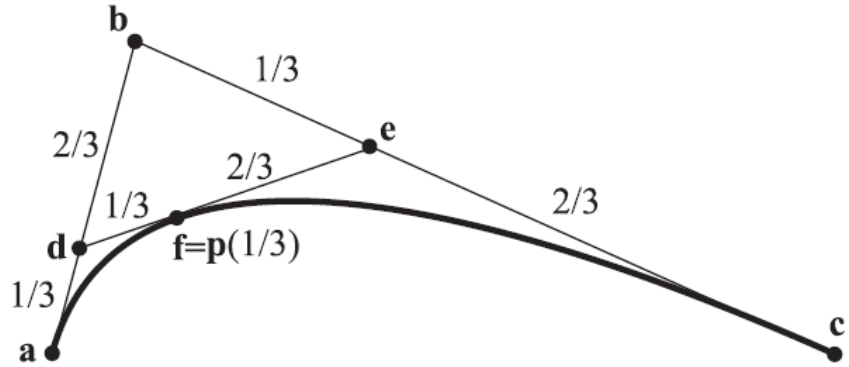
Many fonts are constructed using composite Bézier curves, eg. True type (degree 2) and Postscript Times Roman (degree 3)



Subdivision curves

Subdivision of Bézier Curves

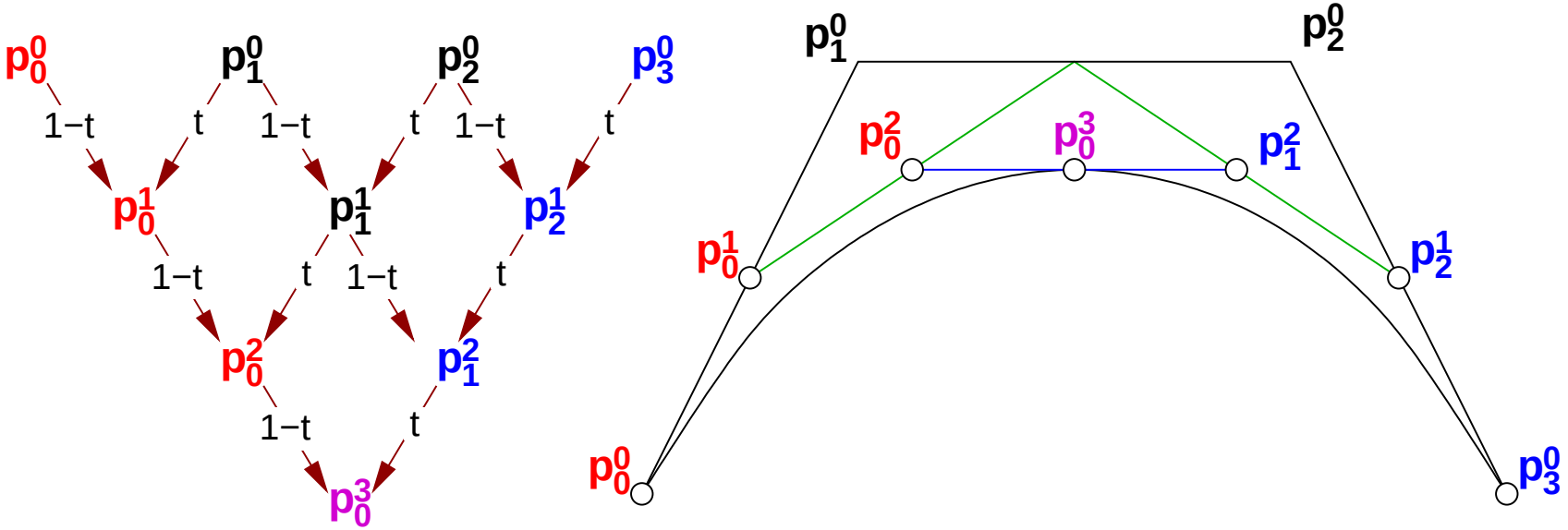
- We have seen that the de Casteljau algorithm evaluates a Bézier curve from its control polygon.



- The intermediate points of the de Casteljau algorithm define the control points of two Bézier segments representing the original.

$\Longrightarrow$ We can divide one Bézier segment into two Bézier segments!

- On the left is the triangular structure of the intermediate points of the de Casteljau evaluation.
- On the right is a figure of the points.



- The original segment on $[0, 1]$ has control points $\mathbf{p}_0^0, \mathbf{p}_1^0, \mathbf{p}_2^0, \mathbf{p}_3^0$.
- The segment on $[0, t]$ has control points $\mathbf{p}_0^0, \mathbf{p}_0^1, \mathbf{p}_0^2, \mathbf{p}_0^3$.
- The segment on $[t, 1]$ has control points $\mathbf{p}_0^3, \mathbf{p}_1^2, \mathbf{p}_2^1, \mathbf{p}_3^0$.

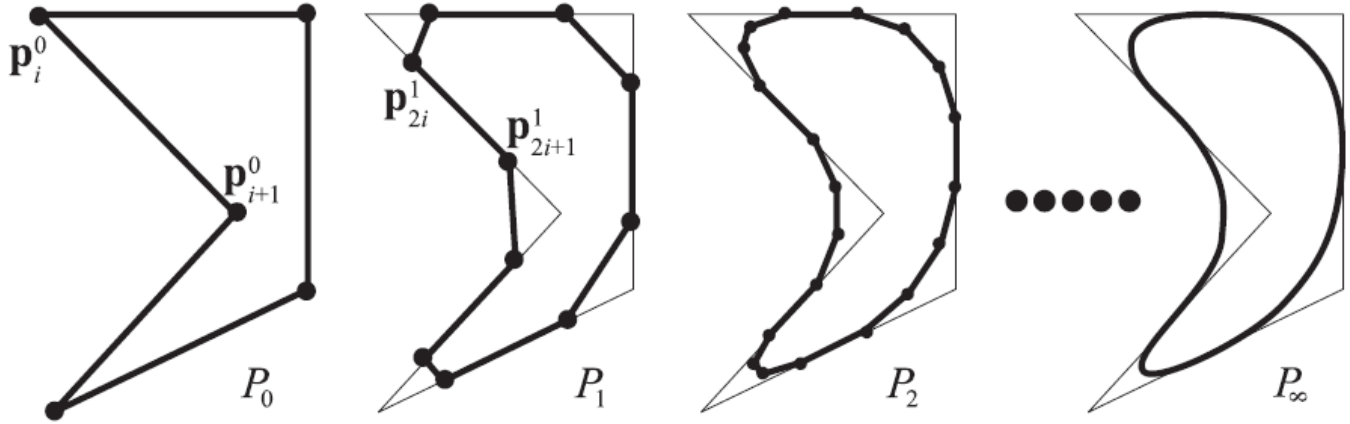
- If we divide at $t = \frac{1}{2}$, we get

$$\begin{aligned}
\mathbf{p}_0^0 &= \mathbf{p}_0^0 & \mathbf{p}_0^3 &= \tfrac{1}{8}(\mathbf{p}_0^0 + 3\mathbf{p}_1^0 + 3\mathbf{p}_2^0 + \mathbf{p}_3^0),\\
\mathbf{p}_0^1 &= \tfrac{1}{2}(\mathbf{p}_0^0 + \mathbf{p}_1^0), & \mathbf{p}_1^2 &= \tfrac{1}{4}(\mathbf{p}_1^0 + 2\mathbf{p}_2^0 + \mathbf{p}_3^0),\\
\mathbf{p}_0^2 &= \tfrac{1}{4}(\mathbf{p}_0^0 + 2\mathbf{p}_1^0 + \mathbf{p}_2^0), & \mathbf{p}_2^1 &= \tfrac{1}{2}(\mathbf{p}_2^0 + \mathbf{p}_3^0),\\
\mathbf{p}_0^3 &= \tfrac{1}{8}(\mathbf{p}_0^0 + 3\mathbf{p}_1^0 + 3\mathbf{p}_2^0 + \mathbf{p}_3^0), & \mathbf{p}_3^0 &= \mathbf{p}_3^0,
\end{aligned}$$

- This is a *refinement rule* for cubic Bézier curves.
- Under repeated division, called *subdivision*, the control polygon *converges to the curve*.
- In practice: render the control polygon after a few subdivisions
- There are similar rules for (uniform) splines
- *Subdivision curves* are defined by such refinement rules.

Subdivision curves

- Curves generated by iterative refinement of a given *control polygon* (vertices=*control points*).



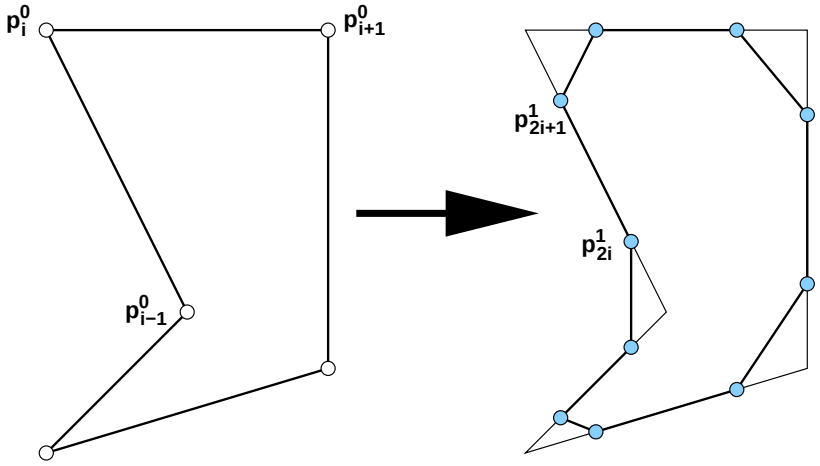
- The result after infinitely many steps is called the *limit curve*.

- Both Bézier curves and spline curves are subdivision curves. (I.e. there exists subdivision schemes to generate them)

- Many such schemes (subdivision zoo).

Chaikin's scheme

- From control points $\ldots, \mathbf{p}_{i-1}, \mathbf{p}_i, \mathbf{p}_{i+1}, \ldots,$



we refine with the rule

$$\mathbf{p}_{2i}^{k+1} = \frac{1}{4}\mathbf{p}_i^k + \frac{3}{4}\mathbf{p}_{i-1}^k$$

$$\mathbf{p}_{2i+1}^{k+1} = \frac{3}{4}\mathbf{p}_i^k + \frac{1}{4}\mathbf{p}_{i-1}^k$$

Important: The indices differs slightly from the book.

- Repeat the refinement ad infinitum.

! Observation: the number of points doubles at each iteration.

- Converges to a quadratic uniform spline curve (C^1) .

Masks and stencils

Collect coefficients into even stencil and odd stencils (rules)

$$\mathbf{a}^{\mathrm{e}} = \{\tfrac{1}{4}, \tfrac{3}{4}\} \quad \text{and} \quad \mathbf{a}^{\mathrm{o}} = \{\tfrac{3}{4}, \tfrac{1}{4}\},$$

and get

$$\mathbf{p}_{2i}^{k+1} = \frac{1}{4}\mathbf{p}_i^k + \frac{3}{4}\mathbf{p}_{i-1}^k = \sum_{j\in\mathbb{Z}} a_j^{\mathrm{e}}\mathbf{p}_{i-j}^k$$

$$\mathbf{p}_{2i+1}^{k+1} = \frac{3}{4}\mathbf{p}_i^k + \frac{1}{4}\mathbf{p}_{i-1}^k = \sum_{j\in\mathbb{Z}} a_j^{\mathrm{o}}\mathbf{p}_{i-j}^k,$$

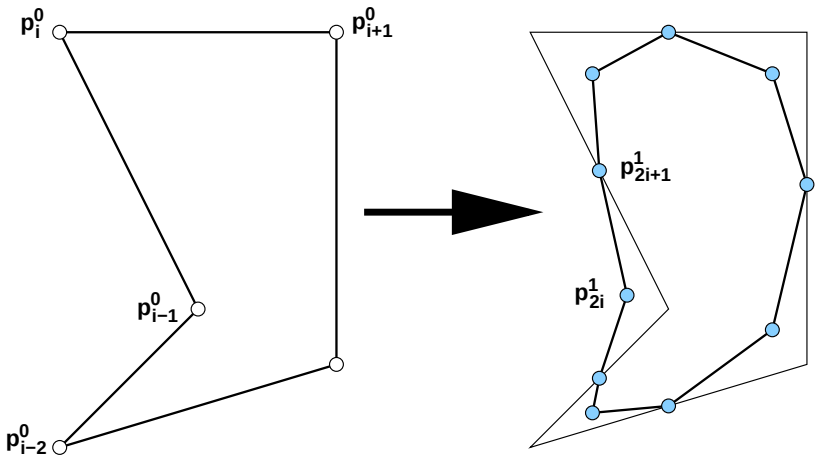
combined, the even and odd stencils form the *mask* of the scheme,

$$\mathbf{a} = \{\tfrac{1}{4}, \tfrac{3}{4}, \tfrac{3}{4}, \tfrac{1}{4}\}$$

which yields $\mathbf{p}_i^{k+1} = \sum_{j\in\mathbb{Z}} a_{i-2j}\mathbf{p}_j^k$ for $i \in \mathbb{Z}$
The mask express the influence of an old vertex on the new verts.

C^2 cubic spline subdivision

Another example is a C^2 cubic spline curve.



The subdivision rule is

$$\mathbf{p}_{2i}^{k+1} = \frac{1}{8}\mathbf{p}_i^k + \frac{6}{8}\mathbf{p}_{i-1}^k + \frac{1}{8}\mathbf{p}_{i-2}^k$$

$$\mathbf{p}_{2i+1}^{k+1} = \frac{1}{2}\mathbf{p}_i^k + \frac{1}{2}\mathbf{p}_{i-1}^k$$

i.e. even (vertex) and odd (edge) stencils

$$\mathbf{a}^{\mathrm{e}} = \{\tfrac{1}{8}, \tfrac{6}{8}, \tfrac{1}{8}\} \quad \text{and} \quad \mathbf{a}^{\mathrm{o}} = \{\tfrac{1}{2}, \tfrac{1}{2}\},$$

and the mask

$$\mathbf{a} = \{\tfrac{1}{8}, \tfrac{1}{2}, \tfrac{6}{8}, \tfrac{1}{2}, \tfrac{1}{8}\}.$$

Note: stencils are convex combinations of control points

General C^{d-1} uniform spline subdivision

Mask for C^{d-1} uniform spline curve of degree d is given by

$$\mathbf{a} = \{a_0, a_1, \ldots, a_{d+1}\} = \frac{1}{2^d}\left\{\binom{d+1}{0}, \binom{d+1}{1}, \ldots, \binom{d+1}{d+1}\right\}$$

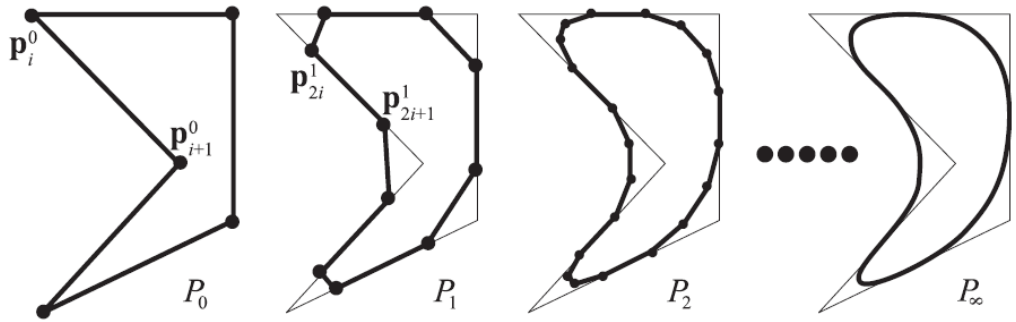
Examples:

$\mathbf{a} = \{\frac{1}{2}, 1, \frac{1}{2}\}$	Linear spline
$\mathbf{a} = \{\frac{1}{4}, \frac{3}{4}, \frac{3}{4}, \frac{1}{4}\}$	Quadratic spline (Chaikin)
$\mathbf{a} = \{\frac{1}{8}, \frac{1}{2}, \frac{6}{8}, \frac{1}{2}, \frac{1}{8}\}$	Cubic spline
$\mathbf{a} = \{\frac{1}{16}, \frac{5}{16}, \frac{10}{16}, \frac{10}{16}, \frac{5}{16}, \frac{1}{16}\}$	Quartic spline

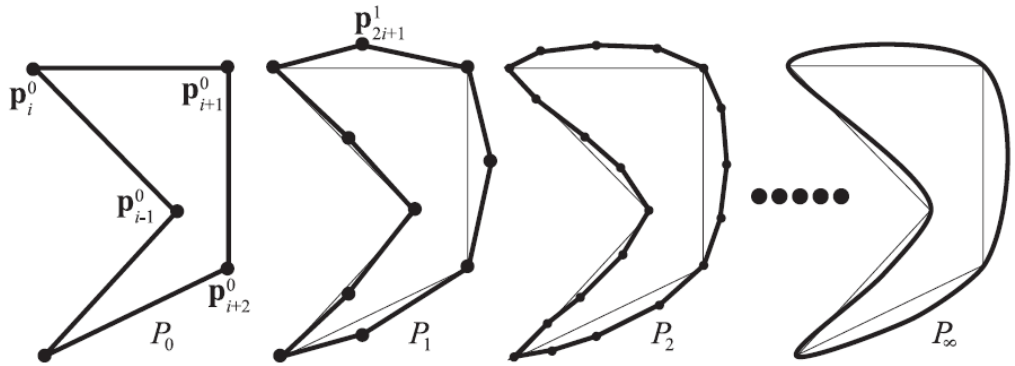
Note: level d mask can be obtained by averaging level $d-1$ masks

Interpolatory subdivision

Quadratic spline scheme: convex comb. yields convex hull-property

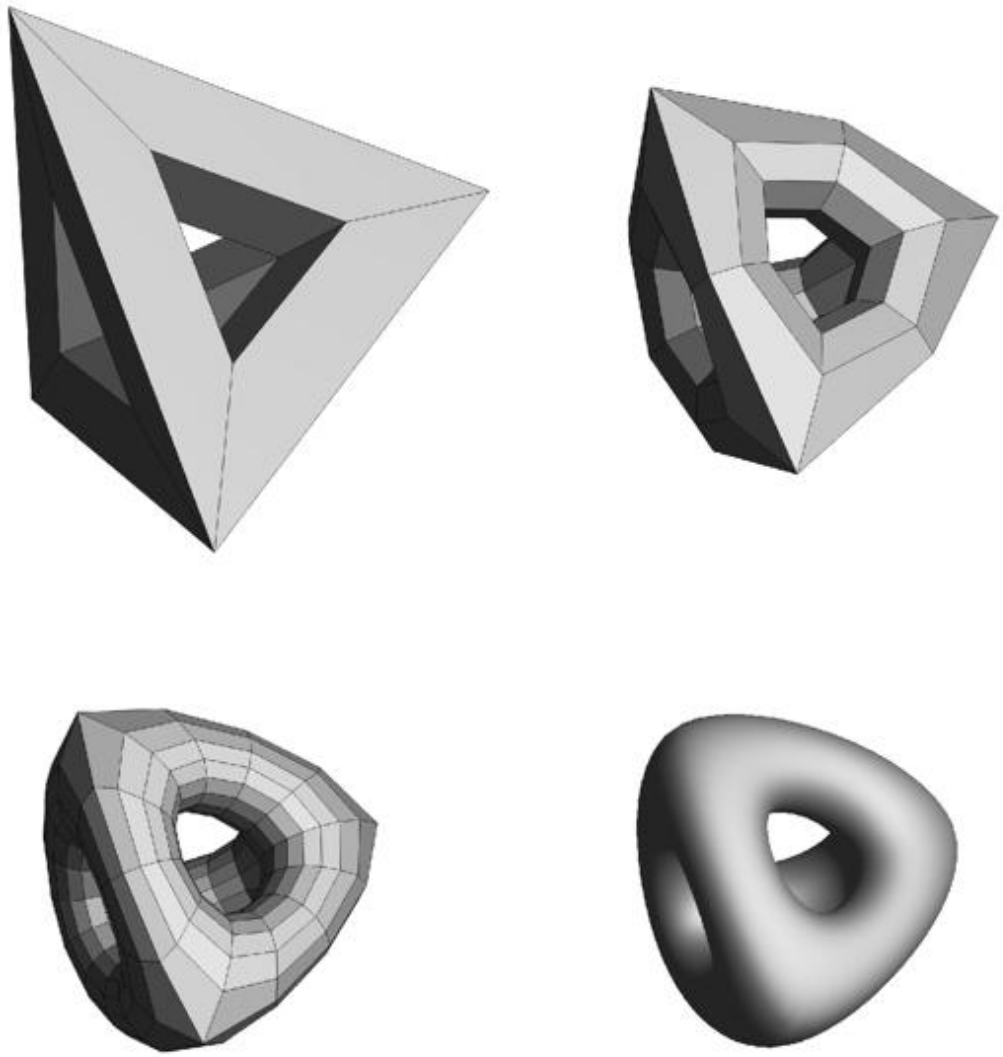


The so-called *four-point scheme* interpolates control points



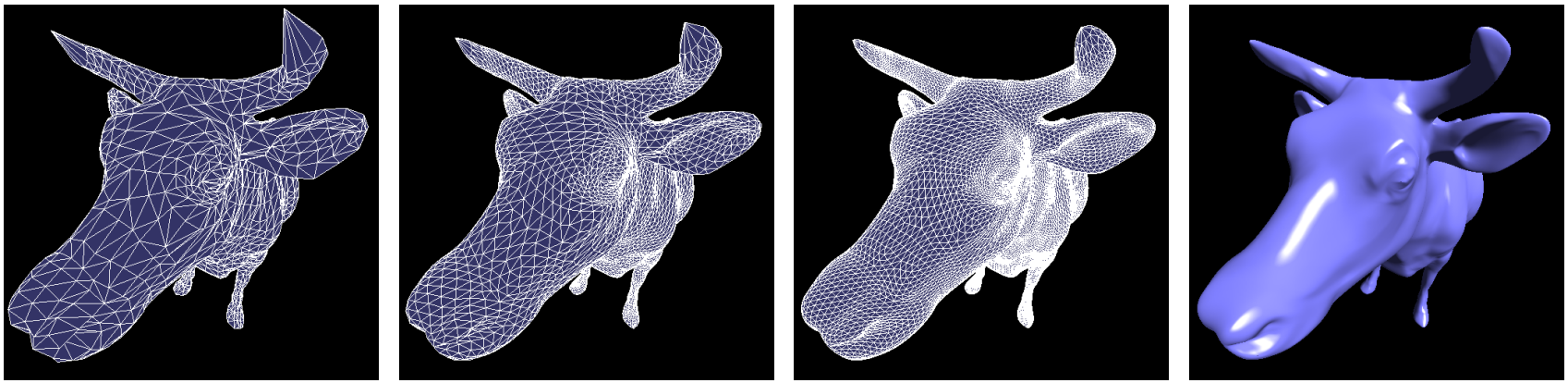
Subdivision surfaces

Subdivision surfaces



Subdivision surfaces

Generated by iterative refinement of a polygonal mesh.



Usually, the mesh has

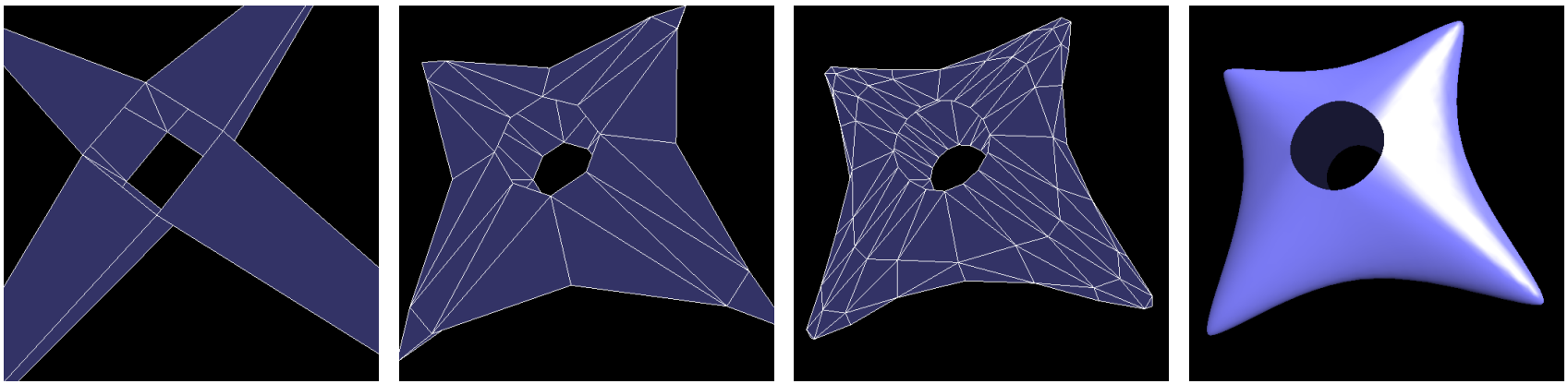
- four-sided faces (quadrilateral meshes) or
- three sided faces (triangle meshes).

Scheme given by refinement rule, typically generalize 1D rules

Can repeat to infinity in theory, interested in *limit surface*

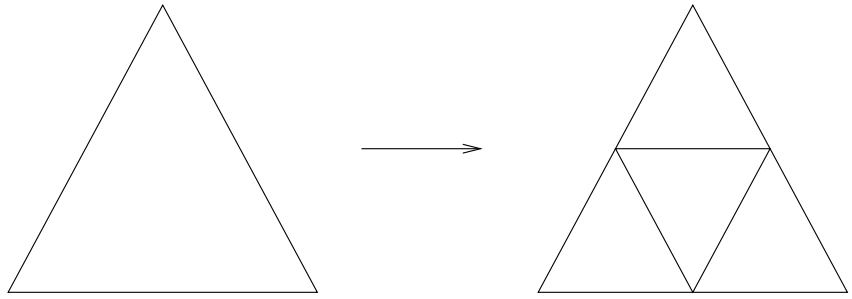
Loop subdivision

A subdivision scheme for arbitrary triangle meshes.



The Loop scheme is based on so-called C^2 quartic 'box-splines'

The Loop subdivision scheme uses the 1-4 split:



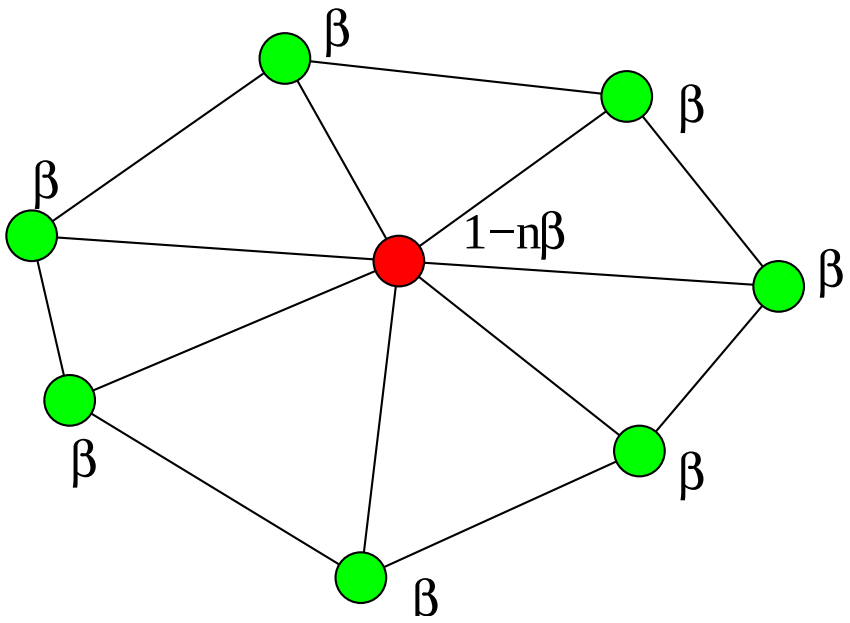
After one subdivision step each former triangle is replaced by four:

$$
\begin{aligned}
|\mathbb{V}^{i+1}| &= |\mathbb{V}^i| + |\mathbb{E}^i|, \\
|\mathbb{E}^{i+1}| &= 2|\mathbb{E}^i| + 3|\mathbb{T}^i|, \\
|\mathbb{T}^{i+1}| &= 4|\mathbb{T}^i|.
\end{aligned}
$$

There are only two stencils in the Loop scheme:

1. positions for vertices corresponding to the old vertices
2. positions for vertices corresponding to edge points.

The *vertex rule* is a combination of the position of the old vertex and its immediate neighbours:



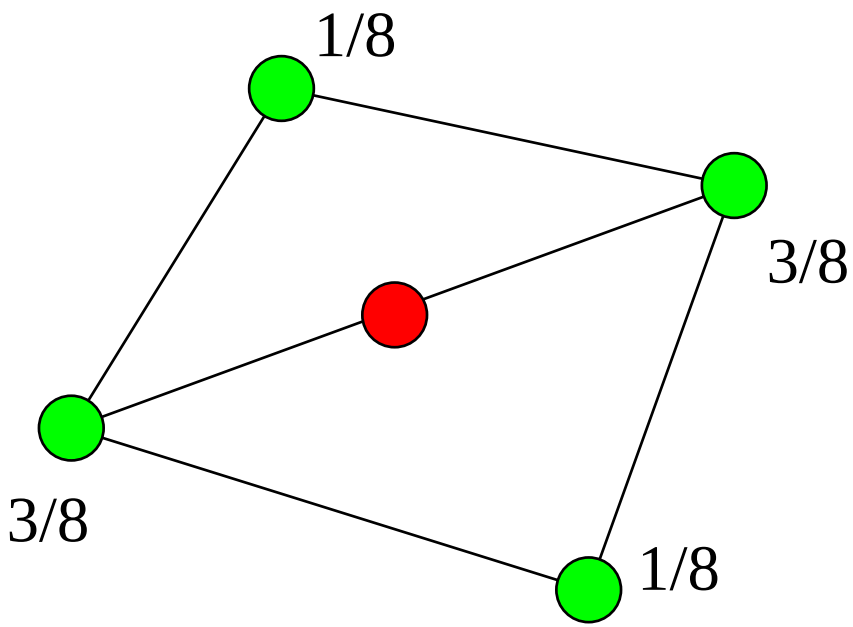
where

$$\beta = \frac{1}{n}\left(\frac{5}{8} - \frac{(3+2\cos(2\pi/n))^2}{64}\right).$$

The 'canonical' case is $n = 6$ in which case the scheme reduces to 'box-spline' subdivision, yielding a C^2 surface.

The limit surface is C^1 at extraordinary points, and C^2 elsewhere.

The *edge rule* is a combination of the vertices of the two triangles adjacent to that edge:



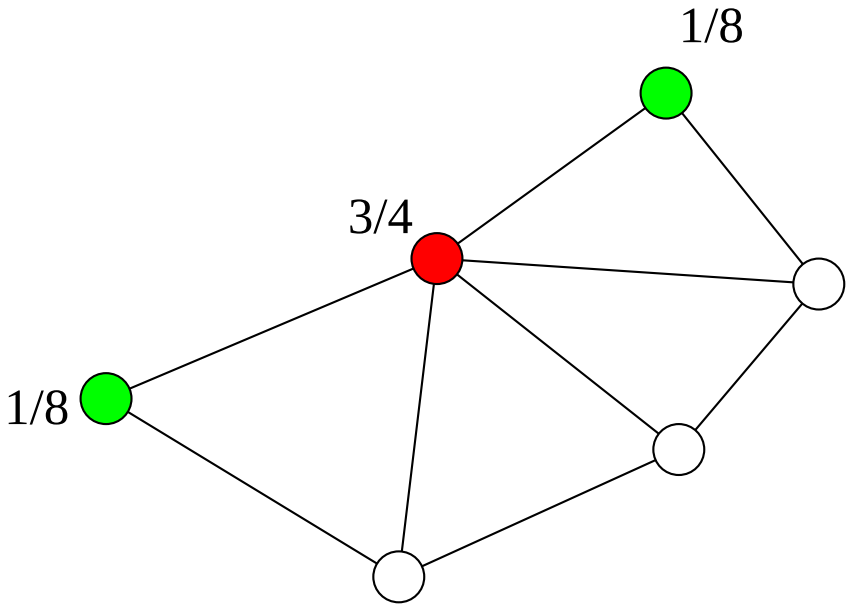
Then, what should we do along the boundary?

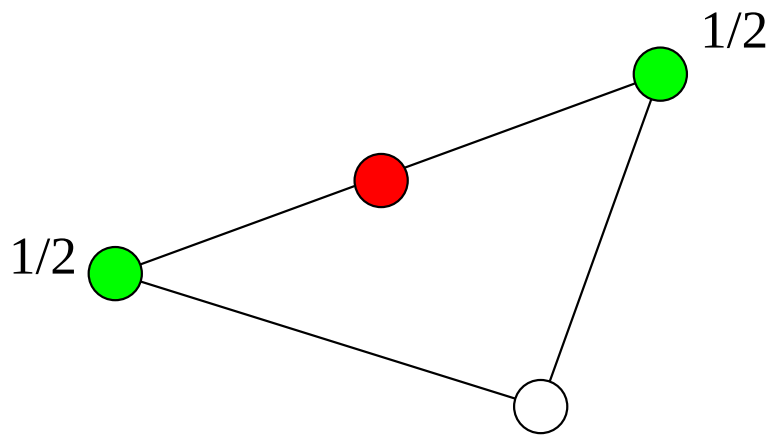
A common technique for dealing with the boundary in the Loop scheme is to let the boundary curve be a cubic spline.

Recall that the mask of cubic spline curve subdivision is

$$\mathbf{a} = \{\tfrac{1}{8}, \tfrac{1}{2}, \tfrac{3}{4}, \tfrac{1}{2}, \tfrac{1}{8}\}$$

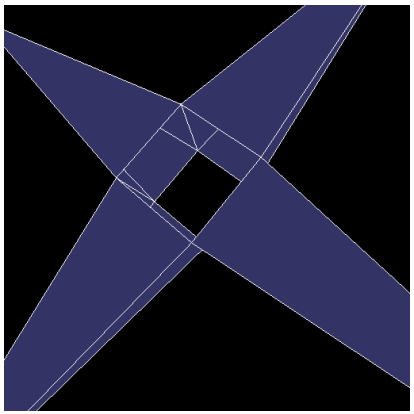
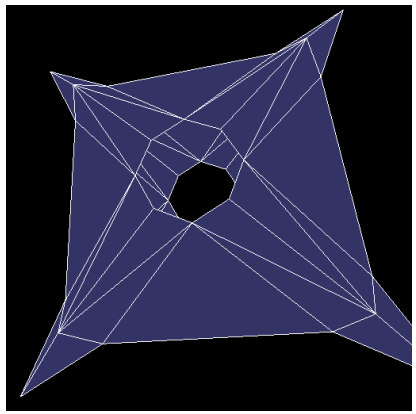
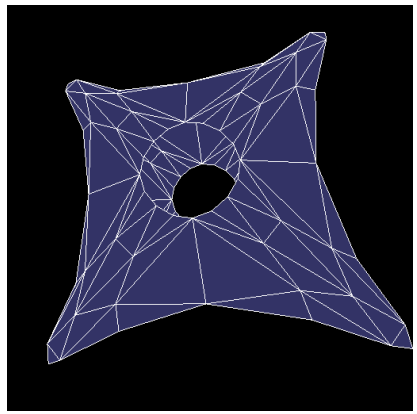
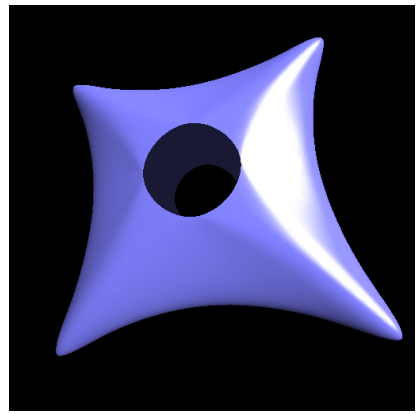
and let boundary edge points be determined by the odd stencil and the boundary vertices be determined by the even stencil, i.e.,





$\sqrt{3}$ subdivision

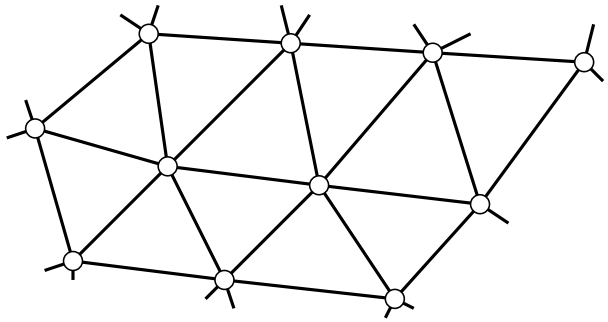
A subdivision scheme for arbitrary triangle meshes.

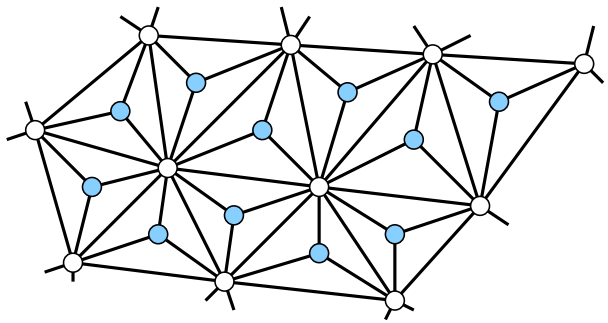
The $\sqrt{3}$ subdivision scheme of Leif Kobbelt creates only 3 new triangles per triangle in each subdivision step (Loop creates 4).

C^1 continuous in extraordinary vertices and C^2 elsewhere

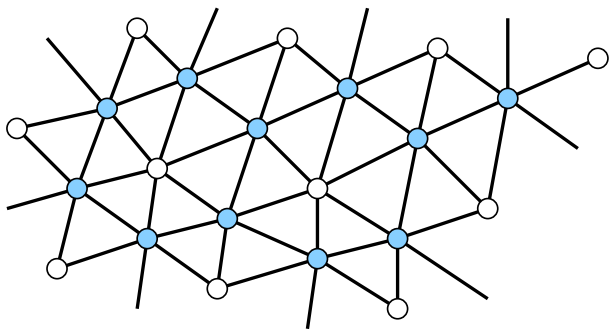
The splitting scheme of $\sqrt{3}$ subdivision is:



Given a triangle mesh, compute the new positions for the vertices as well as the triangle midpoints from the old vertices,

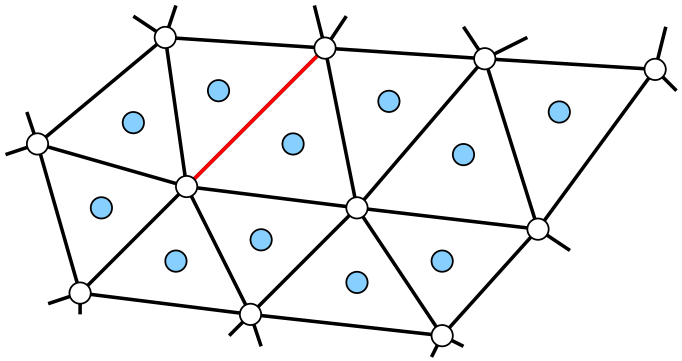


split each triangle into three new triangles,

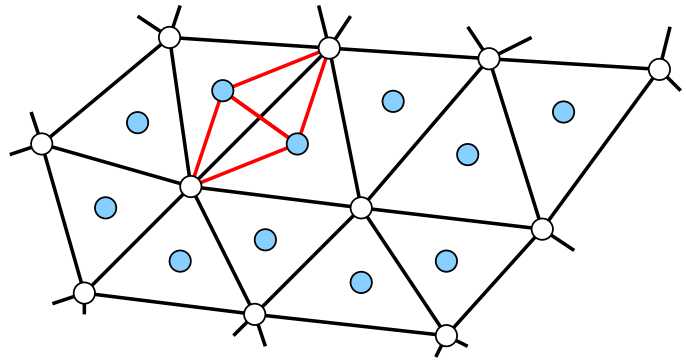


then swap the diagonal of all pairs of adjacent triangles where the common edge is an old edge.

We can create the new triangles directly:



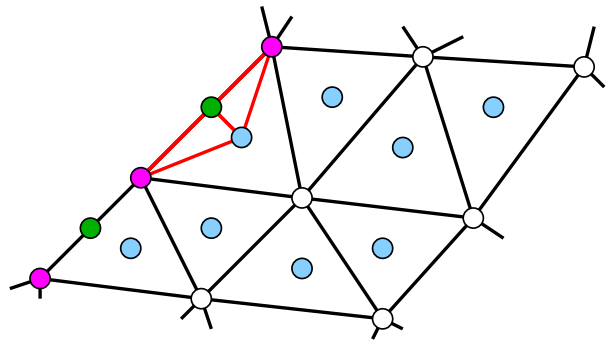
Each edge in the old mesh



yields two new triangles.

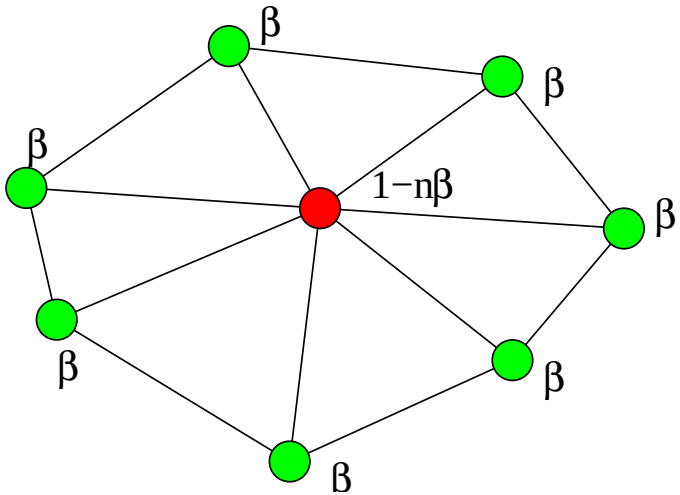
To handle boundaries, we use the same rules as Loop (Slide 48) to calculate boundary edge points and boundary vertices.

Each boundary edge yields two new triangles connecting the end points of the edge, the adjacent triangle midpoint, and the edge point.



Note: Only boundary edges get an edge point!

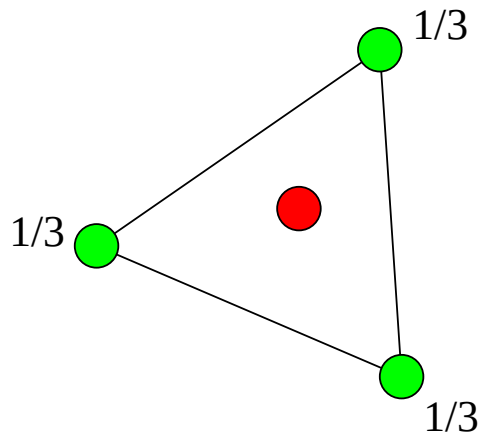
$\sqrt{3}$ subdivision has a vertex rule and a triangle rule:



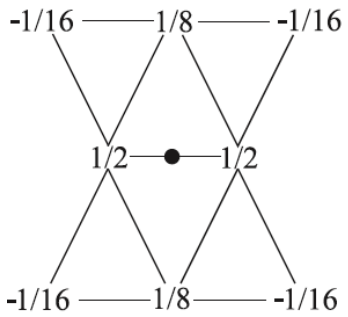
The vertex rule is a blend of the old position and the surrounding vertices, where

$$\beta = \frac{4 - 2\cos(2\pi/n)}{9n}.$$

The triangle midpoint rule is simple, the position is the barycentre of the triangle.



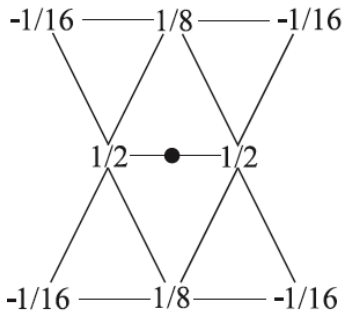
Interpolatory subdivision schemes



The butterfly scheme: vertex points preserved, edge points by affine combination

Yields C^1 smooth surfaces that interpolates control mesh

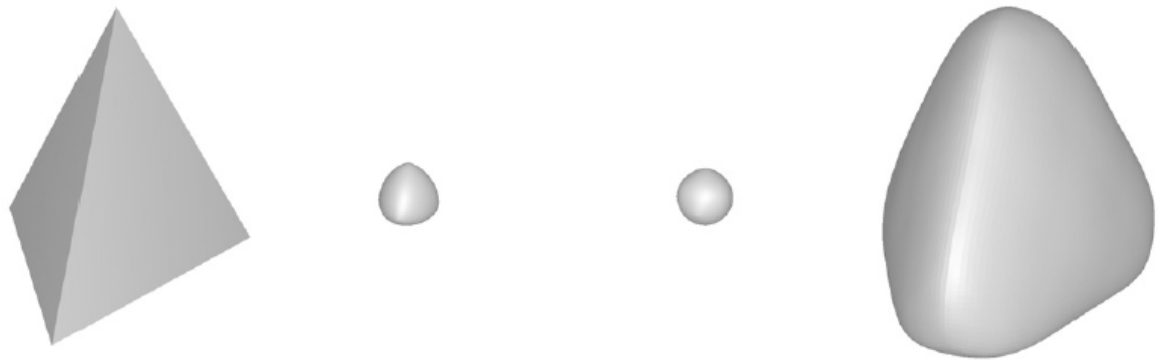
Interpolatory subdivision schemes



The butterfly scheme: vertex points preserved, edge points by affine combination

Yields C^1 smooth surfaces that interpolates control mesh

Approximating (spline-based) and interpolatory schemes produce very different limit surfaces



The last slide

- Litterature:
  - G. Farin, “”Curves And Surfaces For CAGD: A Practical Guide“”
- Courses:
  - MAT-INF4170 - Spline methods
  - MAT-INF4160 - Topics in Geometric modelling

Next time: more smooth surfaces