



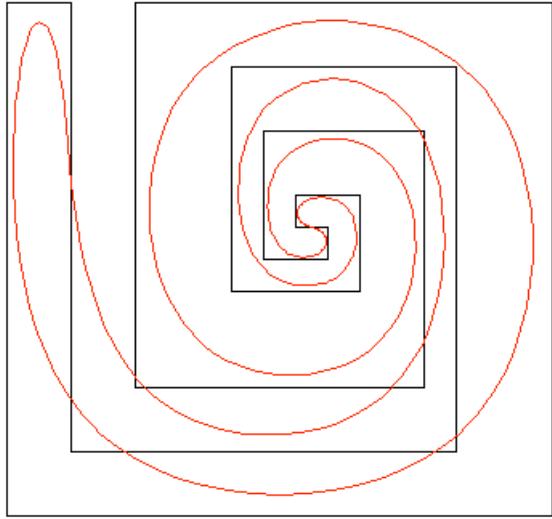
# Curve subdivision

- Smoothing polyloops
- B-spline, Bezier and 4-points subdivisions
- The Jarek subdivision schemes for polyloops
- Surfaces

Updated August 25, 2011

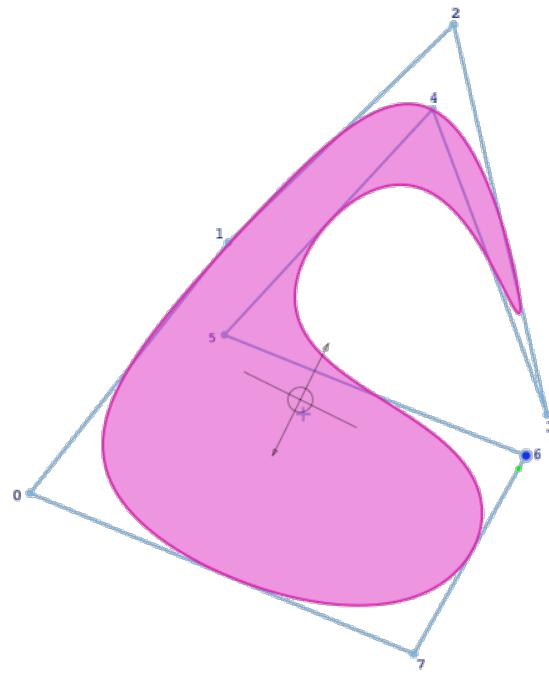
# Motivation

---

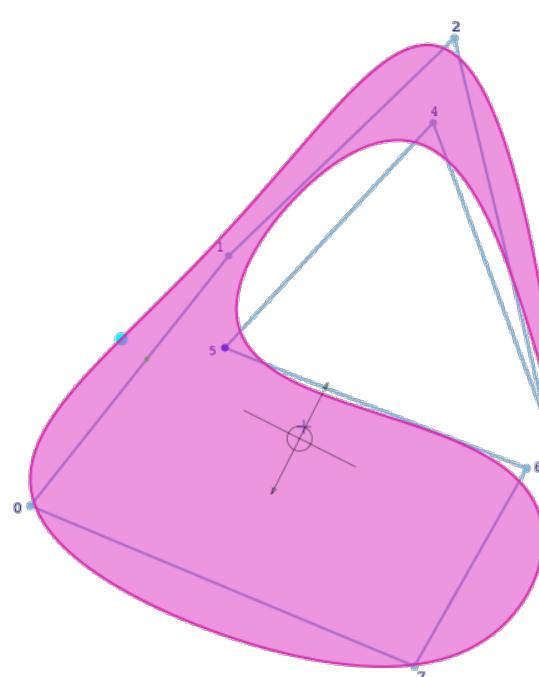


- **Smooth** curves and surfaces are used for aesthetic, manufacturing, and analysis applications where discontinuities due to piecewise linear approximations would create misleading artifacts.
- Designers like to specify such curves by adjusting a **control** network with only a few vertices
- How to go from a crude 2D control polyloop (black) to a smooth curve (red)?
- **Suggestions?**

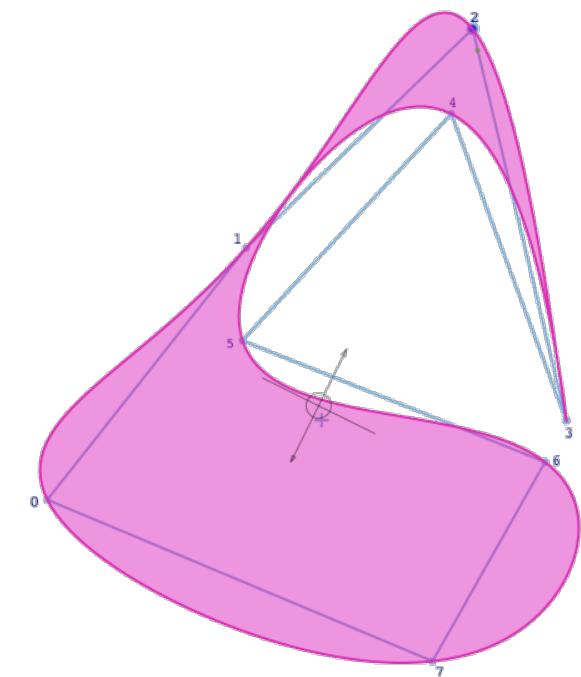
# Different requirements



smooth

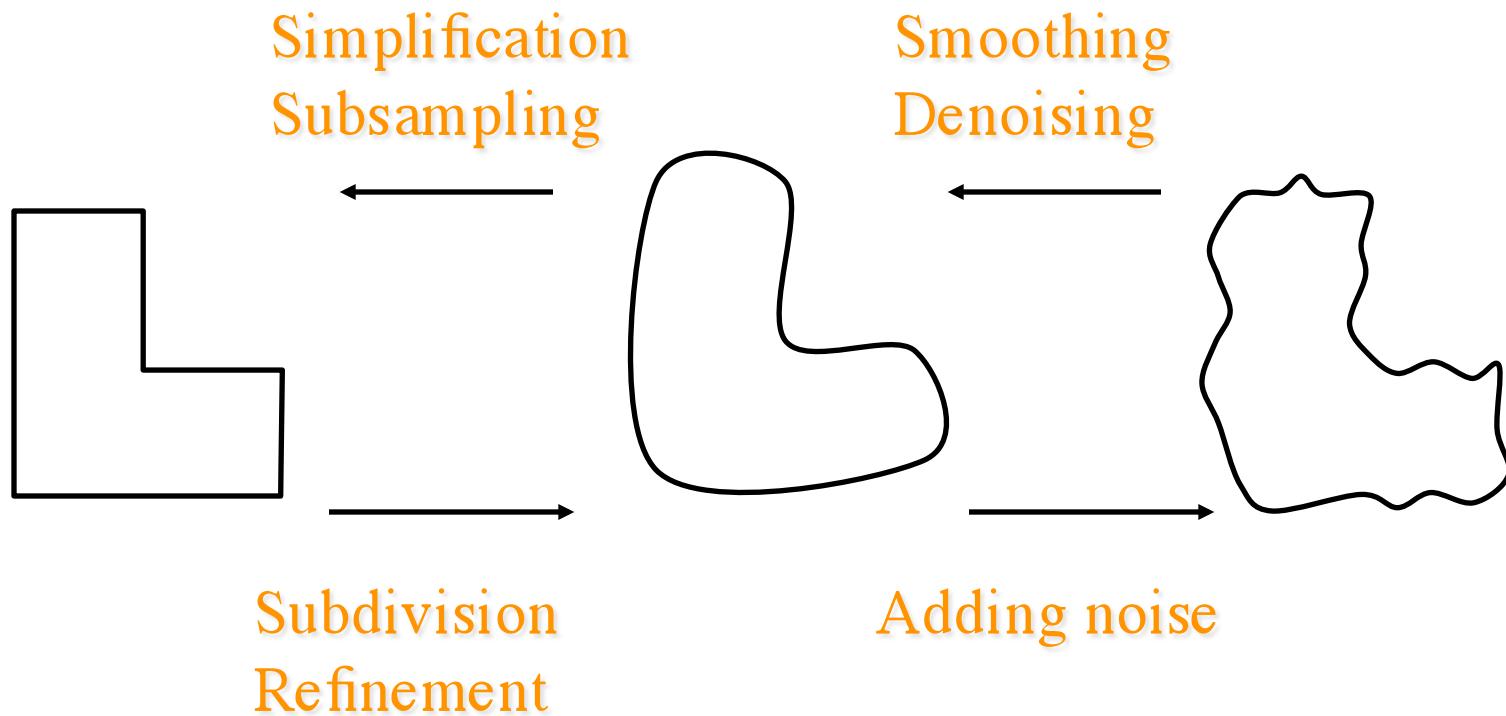


compromise



interpolating

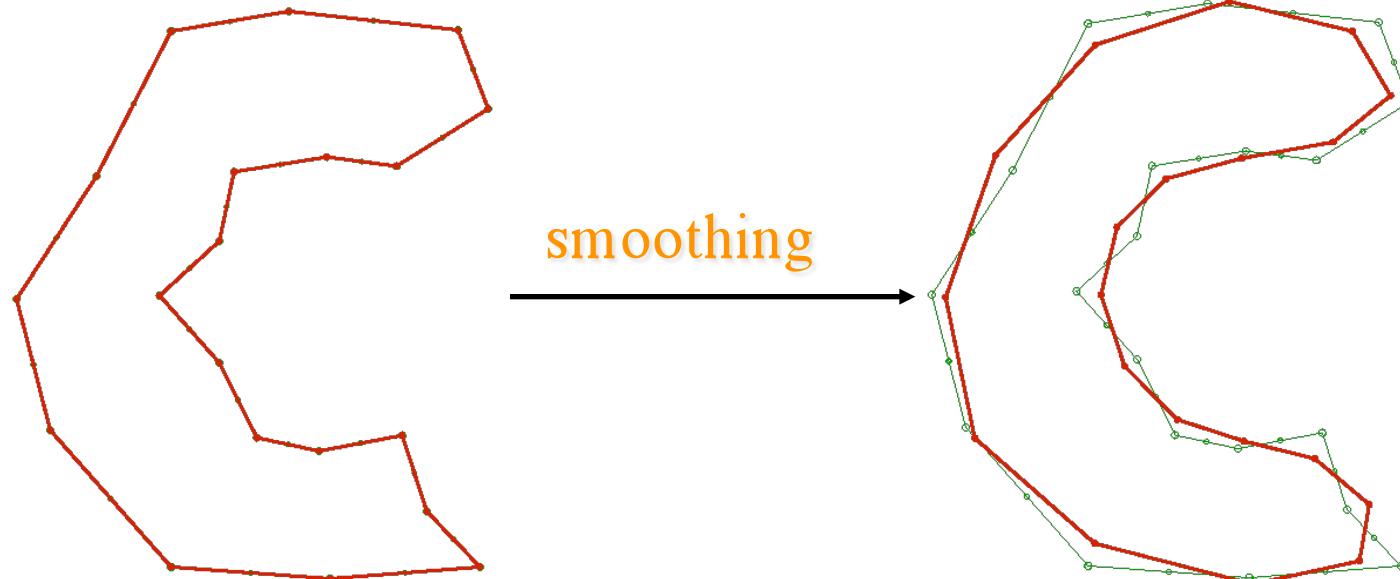
# Operators on polyloops



# Smoothing a polyloop

Given a polyloop L, we wish to move its vertices to produce a curve J that resembles L, but is “smoother” (less jerk?)

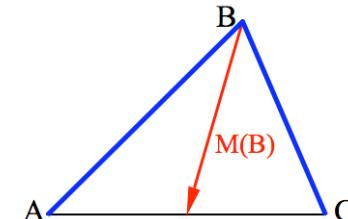
**Smoothing does not change the number of vertices,**  
it only moves them



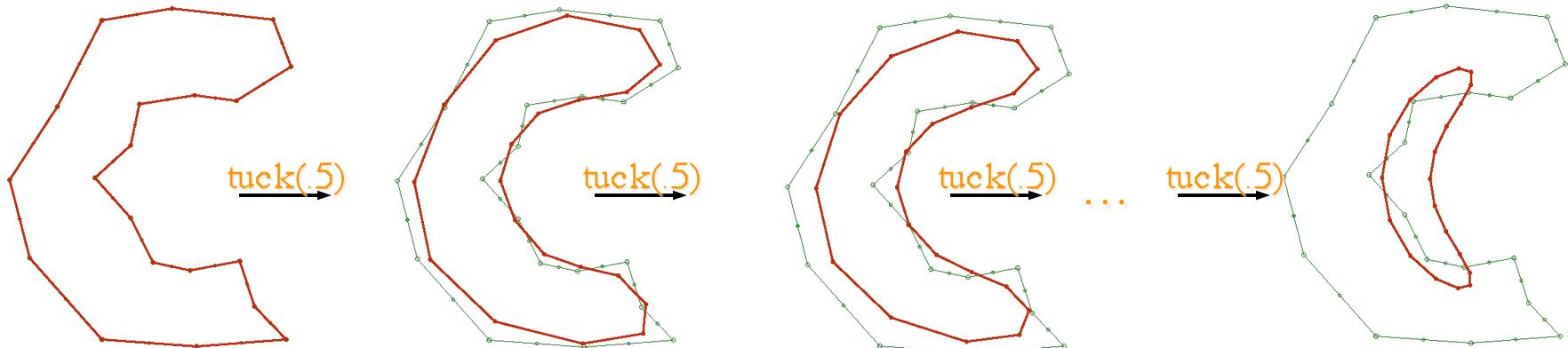
# Tuck smoothing

tuck(s) moves the vertices by executing the following two steps:

- For each vertex B, compute  $M(B)=(BA+BC)/2$ 
  - A (r. C) is the previous (r. next) vertex in the cyclic order
  - Moving B to  $B+M(B)$  brings it to the average of its neighbors
- For each vertex B, move B to  $B+sM(B)$

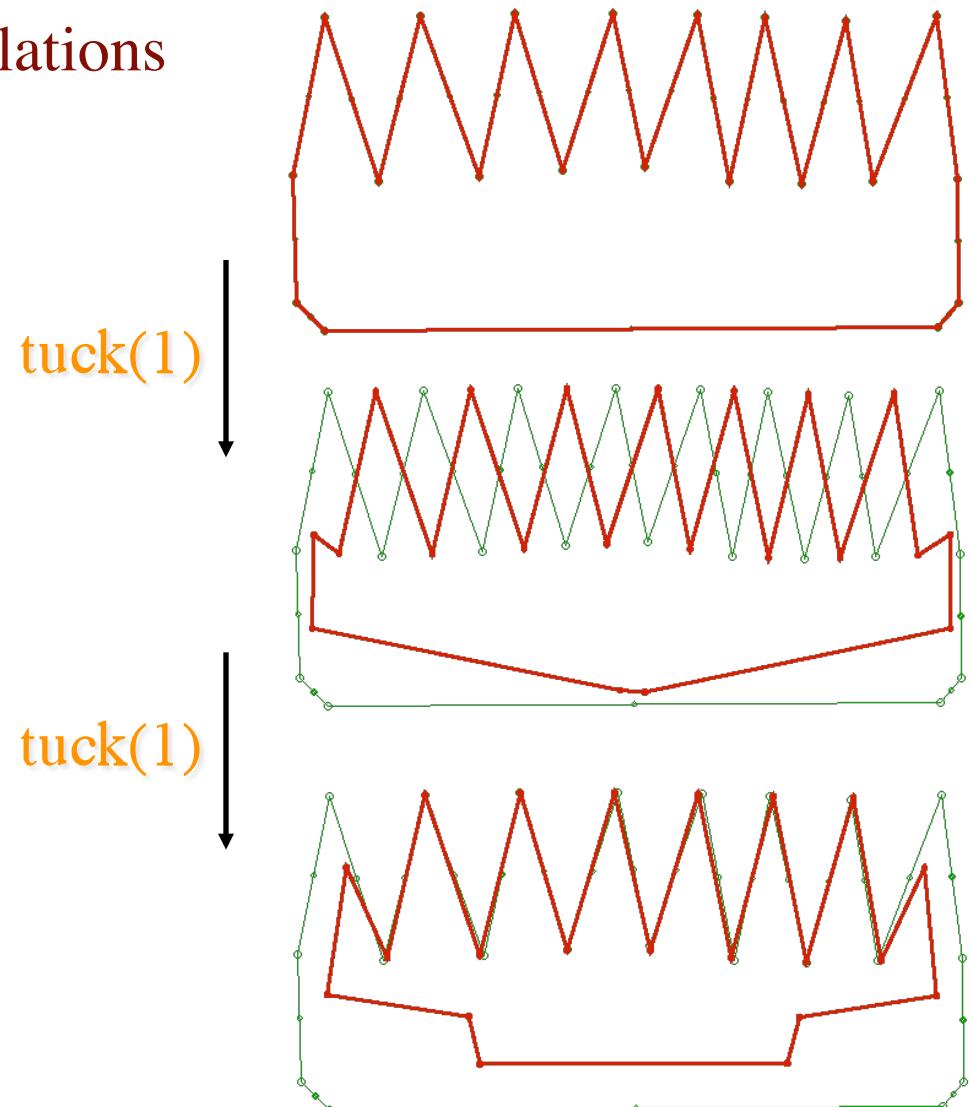


Repeating tuck(s) tends to eliminate sharp features, but shrinks the curve



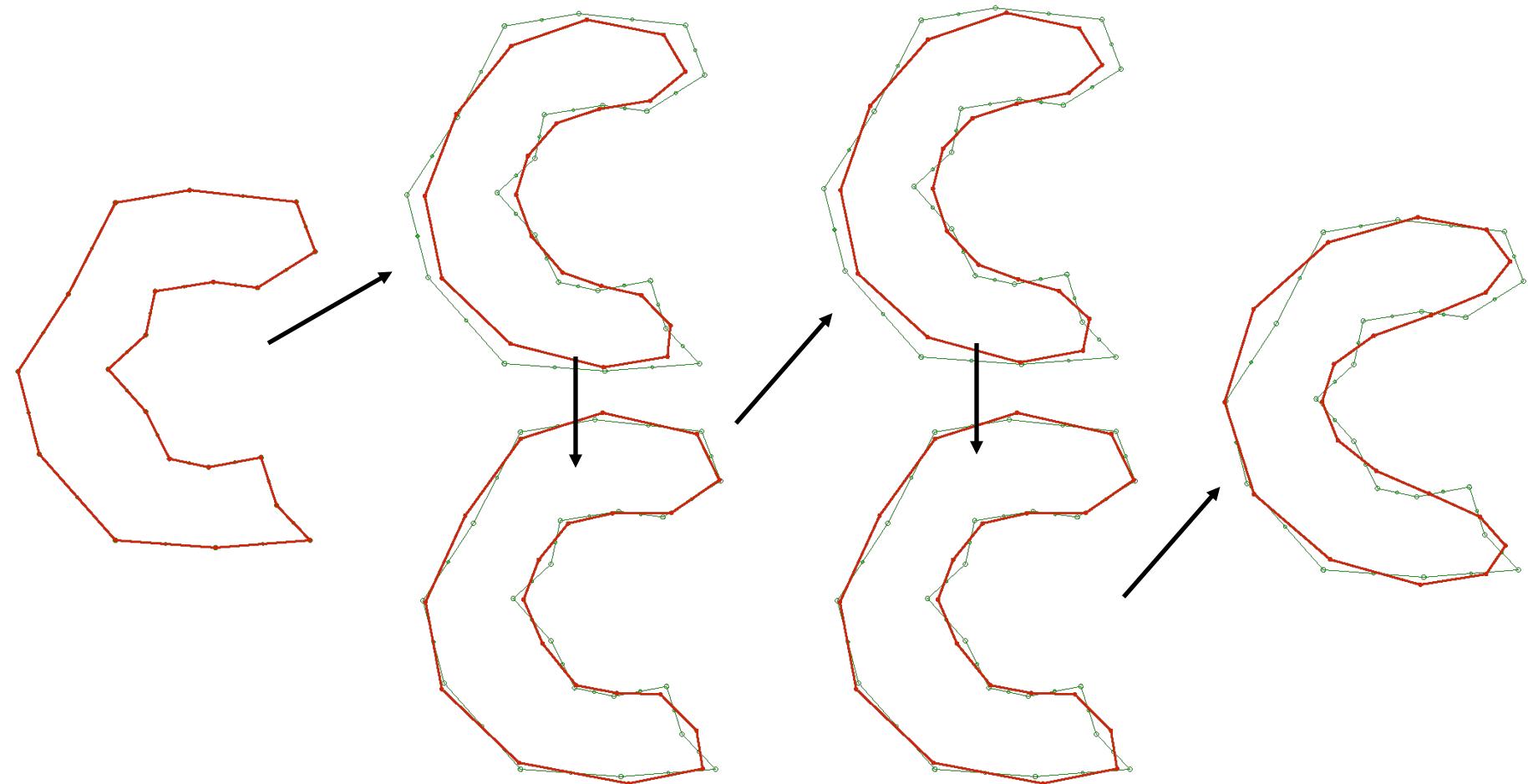
# Avoid oscillations

Keep  $s$  in  $[0,2/3]$  to avoid oscillations



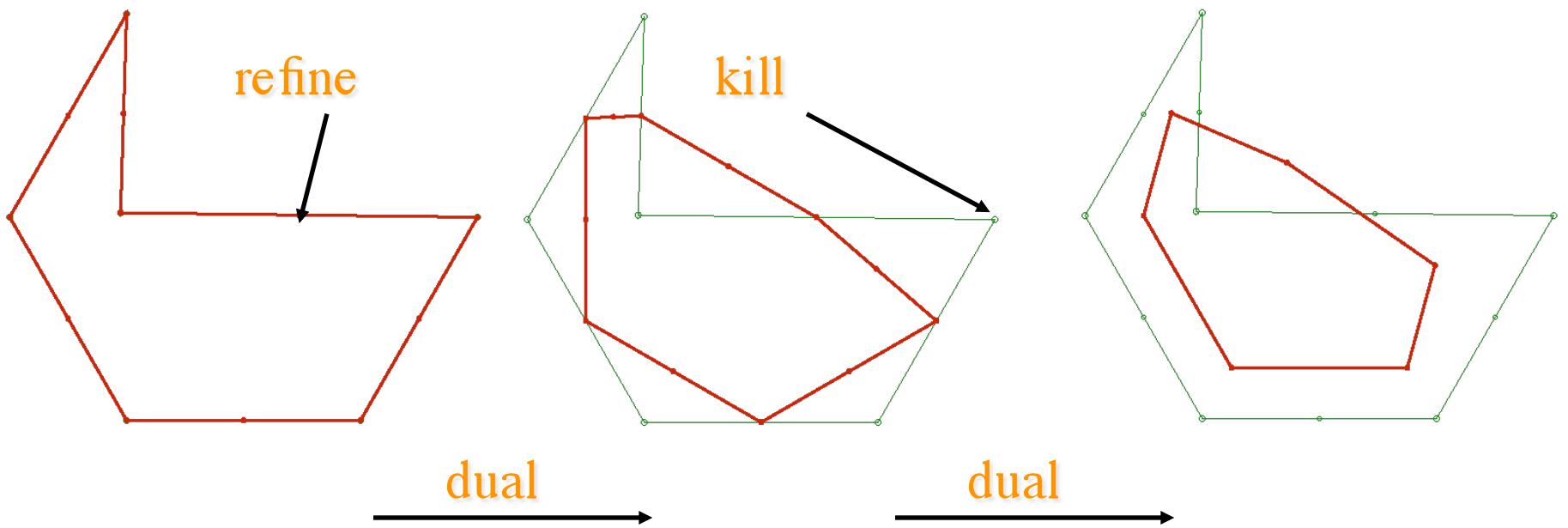
# Avoid shrinking: Tuck-tuck smoothing

- Alternate **tuck(s)**, which shrinks, and **tuck(-s)**, which expands
  - untuck cannot recover the sharp features lost during the tuck



# Tuck(1/2) = dual of dual

- Tuck(.5) is the dual of the dual
  - refine introduces a new vertex in the middle of each edge
  - kill deletes the old vertices
  - dual = (refine, kill)
  - tuck(.5) = (dual, dual)



# Cubic fit tuck-tuck: $(-A+4B+4D-E)/6$

Compute a cubic polynomial curve,  $\mathbf{C}(t)=at^3+bt^2+ct+d$ , through vertices A, B, D, E satisfying:  $\mathbf{C}(-2)=A$ ,  $\mathbf{C}(2)=E$ ,  $\mathbf{C}(-1)=B$ , and  $\mathbf{C}(1)=D$

We want to insert a point  $\mathbf{C}=C(0)=d$  between B and E. Solve for d

$$\mathbf{C}(-2)=-8a+4b-2c+d=A \text{ and } \mathbf{C}(2)=8a+4b+2c+d=E \text{ yields } A+E=8b+2d$$

$$\mathbf{C}(-1)=-a+b-c+d=B, \text{ and } \mathbf{C}(1)=a+b+c+d=D \text{ yields } A+E=8b+2d$$

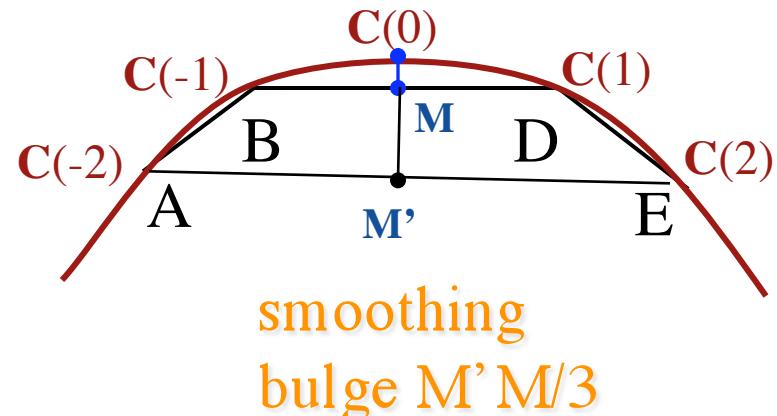
$$A+E=8b+2d \text{ and } A+E=8b+2d \text{ yields } 6d=4(B+D)-(A+E)$$

$$d = (-A+4B+4D-E)/6 = M+M'M/3, \text{ with } M=(B+D)/2, M'=(A+E)/2$$

To insert such a cubic fit for each edge:  $\text{tuck}(2/\sqrt{6})$ ,  $\text{tuck}(-2/\sqrt{6})$

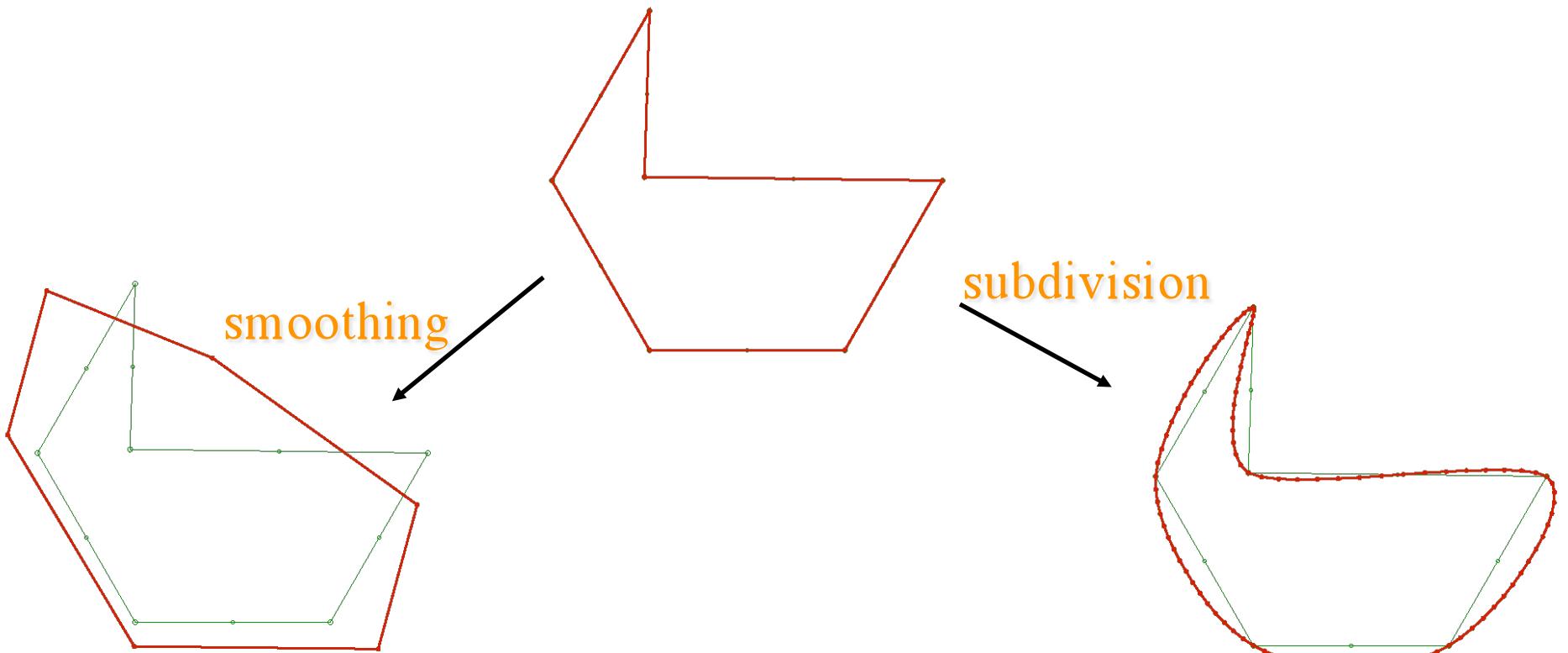
Note that  $2/\sqrt{6}=0.82$ , which may be unstable

Move towards C



# Smoothing vs subdivision

- **Smoothing** should be performed when the desired curve has been densely sampled but the resulting polyloop has **undesired sharp features** (acquisition noise, too many details, sharp changes in direction or curvature)
- **Subdivision** refines the polyloop (**adds more vertices**). It should be performed when the polyloop is a **crude outline** of the desired curve.



# Polyloop refine/smooth operators

---

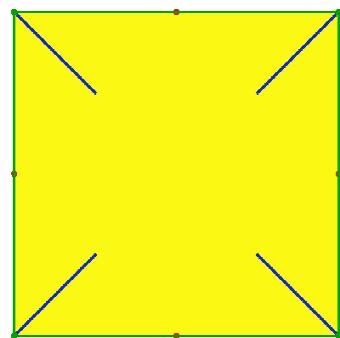
We will use the following operators:

- r (*refine*) introduces a new vertex in the middle of each edge
- k (*kill*) deletes the old vertices (those present before r)
- d (*dual*) performs (r, k)
- t (*tuck*) performs  $\text{tuck}(1/2)$ , which is (d, d)
- $u_s$  (*parameterized untuck*) performs  $\text{tuck}(-s)$
- u (*untuck*) performs  $\text{tuck}(-1/2)$ , same as  $u_{.5}$

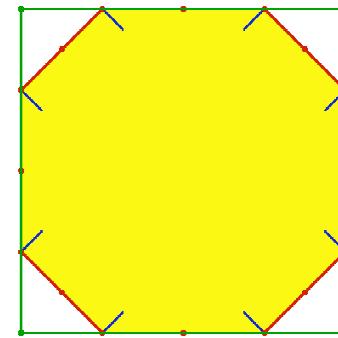
Write a polyloops **smoothing** scheme using these

# Quadratic uniform B-spline (r,d)

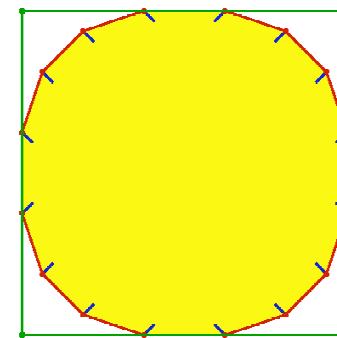
Repeating (r,d) converges to a piecewise quadratic B-spline curve that interpolates the mid-edge points



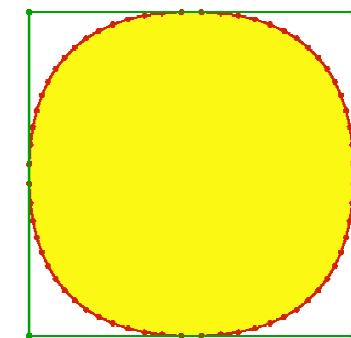
r



rd



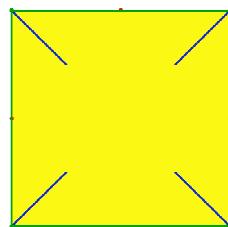
rdrd



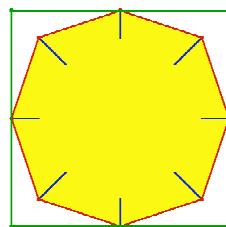
rdrdrdrd

# Cubic uniform B-spline (r,t)

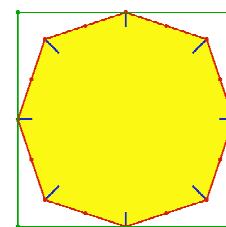
Repeating (r,t), which is (r,d,d) converges to a piecewise cubic B-spline curve that has second degree continuity (no jerk)



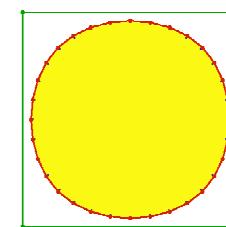
rt



rtr



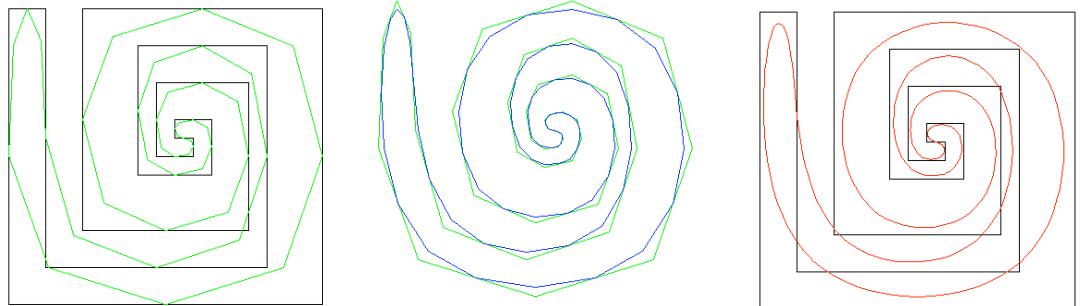
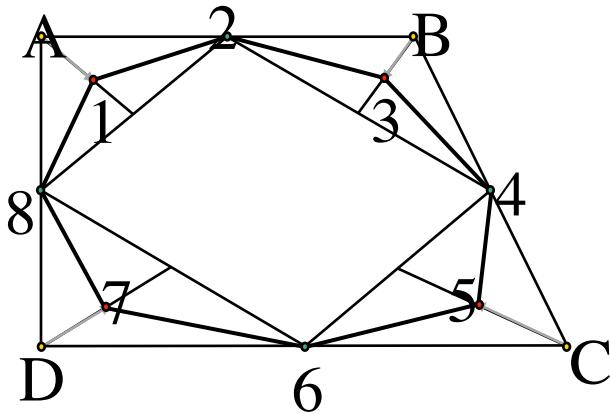
rtrt



rtrtrt

# Split&tweak B-spline subdivision

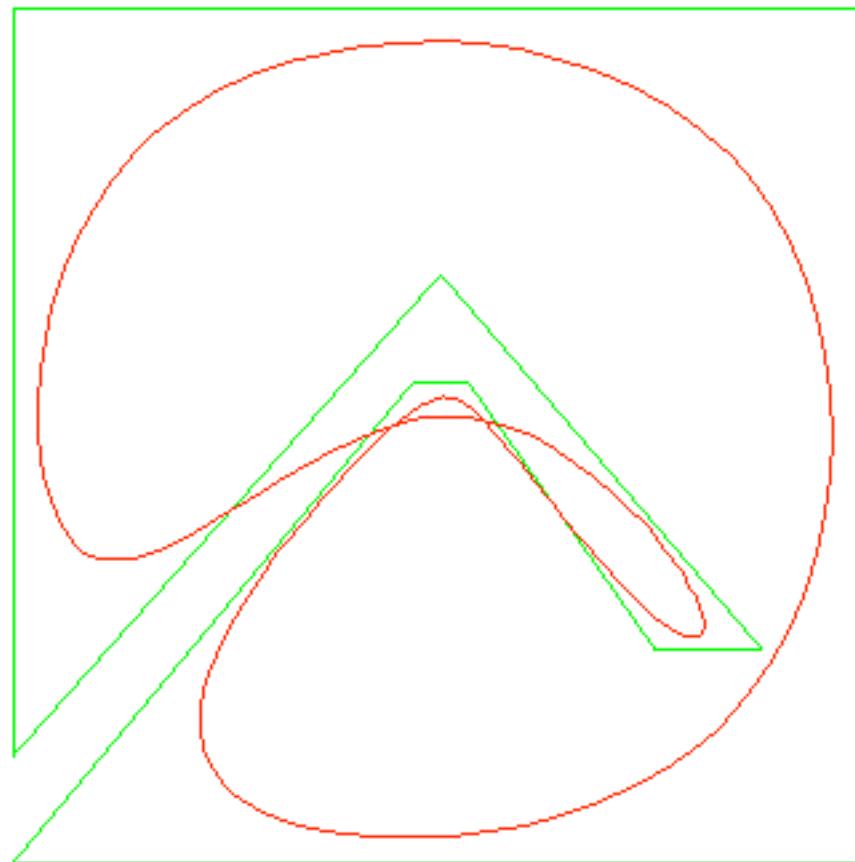
- Given a control polygon, for example (A,B,C,D), repeat the following sequence of two steps, until all consecutive 4-tuples of control points are nearly **co-linear**.
  - “Split”: insert a new control point in the middle of each edge (2,4,6,8)
  - “Tweak”: move the old control points half-way towards the average of their new neighbors (1,3,5,7)
- The polyloop converges rapidly to a smooth curve, which happens to be a cubic B-spline curve.



# Can the limit curve self-intersect?

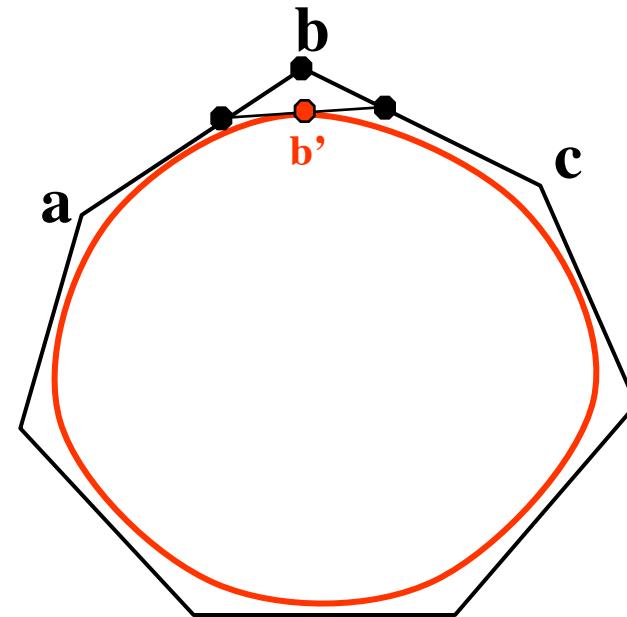
---

- Can it self-intersect when P does not?



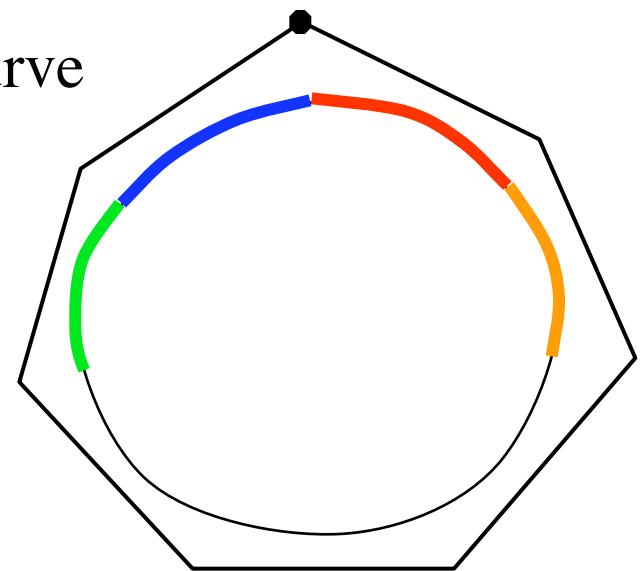
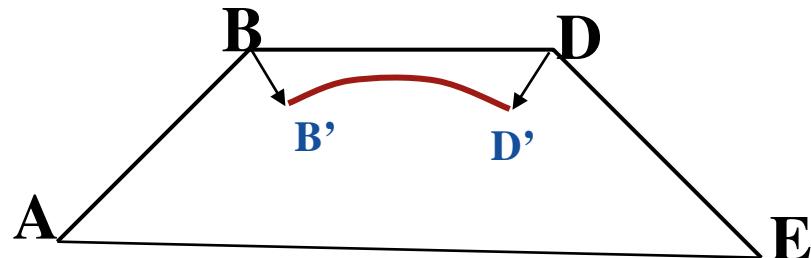
# Where do the control vertices converge to?

- The original control vertex B **converges** to  $B'=(A+4B+C)/6$
- The **tangent** at B' to the B-spline curve is parallel to AC



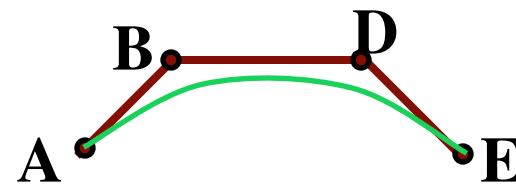
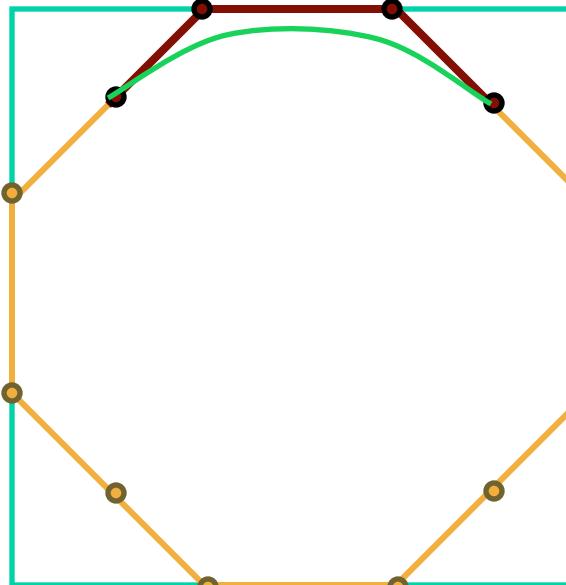
# Local control for B-spline curves

- Trace the original vertices **during subdivision**
  - Do they move along straight lines?
- They subdivide the final curve into **spans**
- **Each span is influenced by 4 control vertices**
- **Each control vertex influences 4 spans**
- Each span converges to a cubic Bezier curve



# Cubic B-spline–to–Bezier conversion

- Introduce 2 new vertices in each edge to split it in 3 equal parts
- Move the old vertices to the average of their new neighbors
- Make groups of 4 consecutive vertices (each sharing its old vertices with consecutive groups)
- Each group is the control polygon (A,B,D,E) of a Bezier curve



# How to evaluate the Bezier control polygon

Let  $t$  be the parameter in  $[0,1]$ .

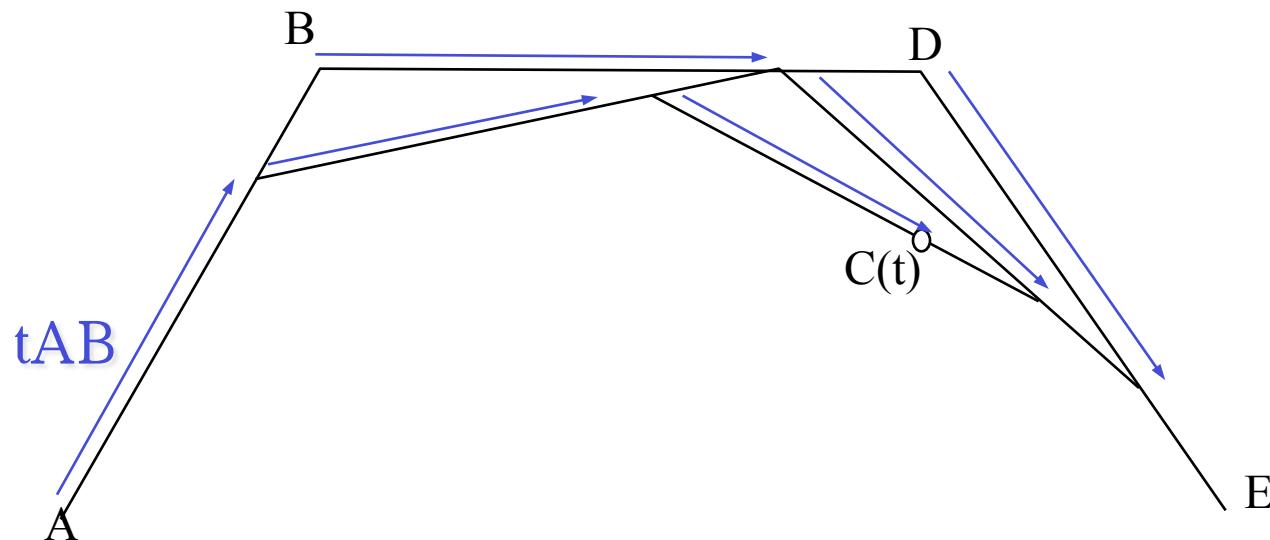
Let  $C(t)$  be the point on the Bezier curve corresponding to  $t$

You can compute  $t$  as

$$C(t) = s(s(s(A,t,B),t,s(B,t,C)), t, s(s(B,t,C),t,s(C,t,D)))$$

where  $s(A,t,B)$  returns  $A + tAB$

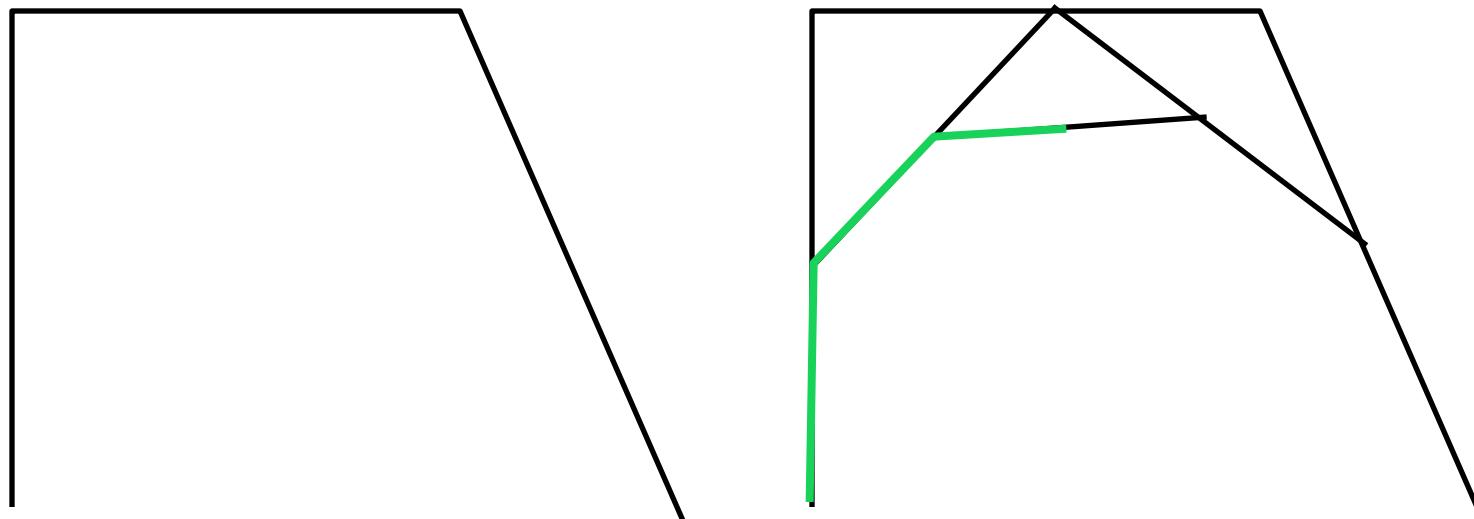
Note that this is a cubic polynomial in  $t$



# How to subdivide the Bezier control polygon

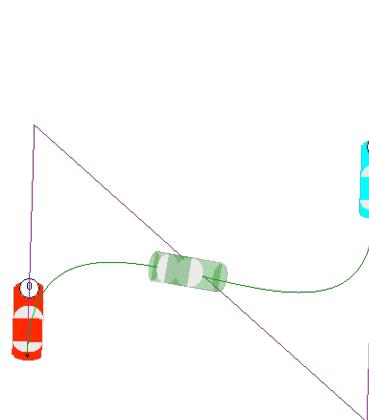
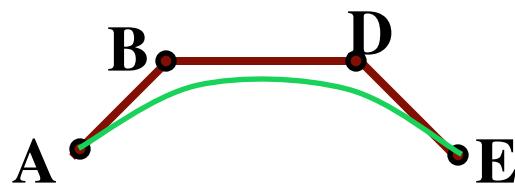
---

- Perform the evaluation of  $C(0.5)$  and keep the construction points



# Convex hull

- Each Bezier curve lies in the convex hull of 4 control vertices
- The convex hull of 4 points in 3D?
  - A tetrahedron
- The convex hull of 4 points in 2D?
  - Contained in the union of 3 triangles: ABD, ABE, BDE



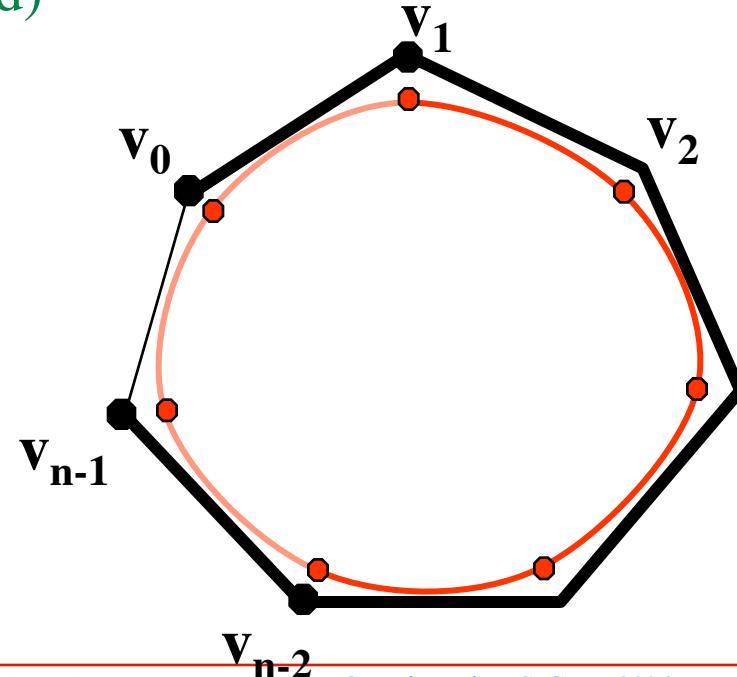
# How can we use the convex hull property?

---

- Display uncertainty (region guaranteed to contain the limit curve)
  - Shade triangles (in 2D) or tetrahedra (in 3D)
- Decide when to stop the subdivision
  - All triangles or tetrahedra are nearly linear
- Quickly decide when two B-spline curves in 2D are not intersecting
  - The convex hulls do not overlap
  - Useful for accelerating collision detection and for computing the exact collision time

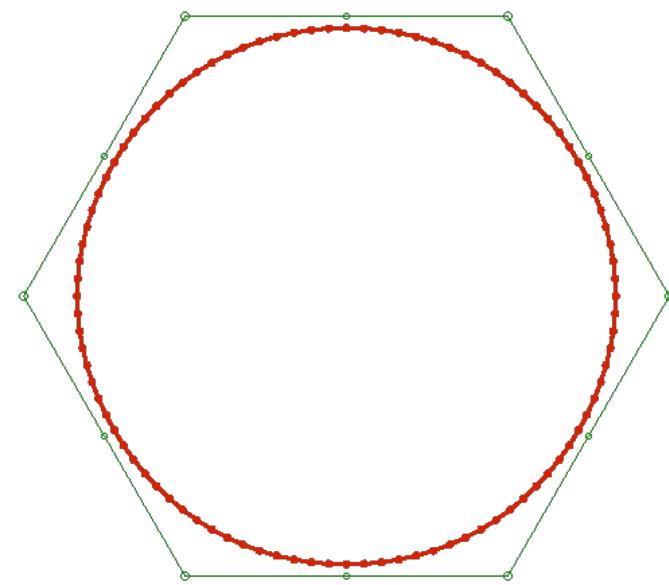
# Open curves

- If you want to subdivide an open polygon
  - Trace the original vertices through the subdivision process
  - They split the curve into spans (one per original edge)
- Delete the span  $(v_{n-1}, v_n)$  and the two adjacent spans
  - $v_0$  should not influence the head (beginning of the curve)
  - $v_{n-1}$  should not influence the tail (end)

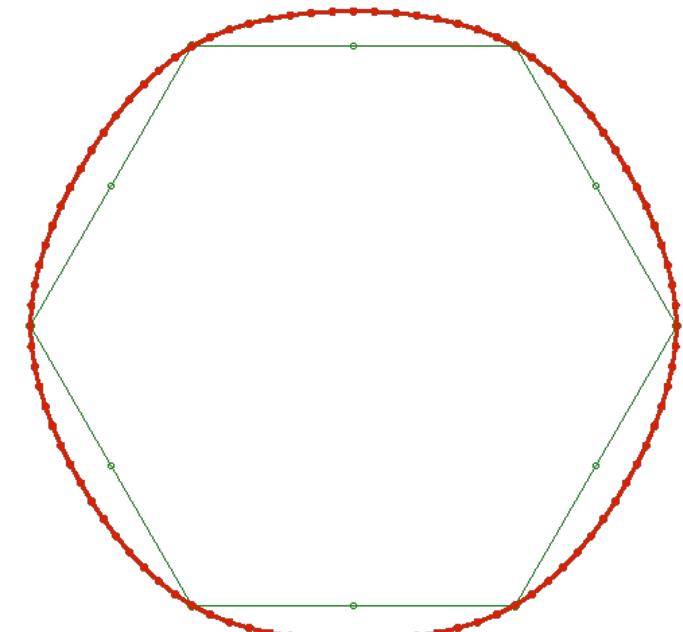


# B-spline curves are not interpolating

- The cubic B-spline is **approximating**
  - It does not go through the control vertices
- You may prefer the 4-point, which is **interpolating**



B-spline (approximating)



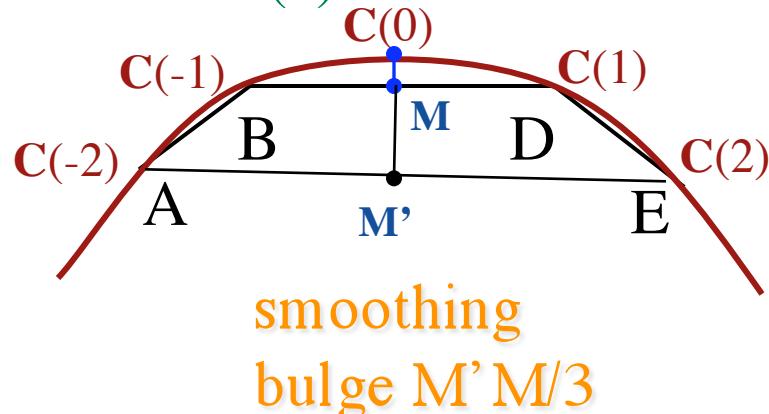
4-point (interpolating)

# 4-point subdivision

- We want to leave the original vertices of L where they are.
- Hence, we displace (bulge-out) each inserted mid-edge point C.
- By how much? Fit a cubit  $C=C(t)$  through A, B, D, and E
  - $C(t)=at^3+bt^2+ct+d$ , and thus  $C(0)=d$

**We use different constraints than the cubic fit tuck-tuck!**

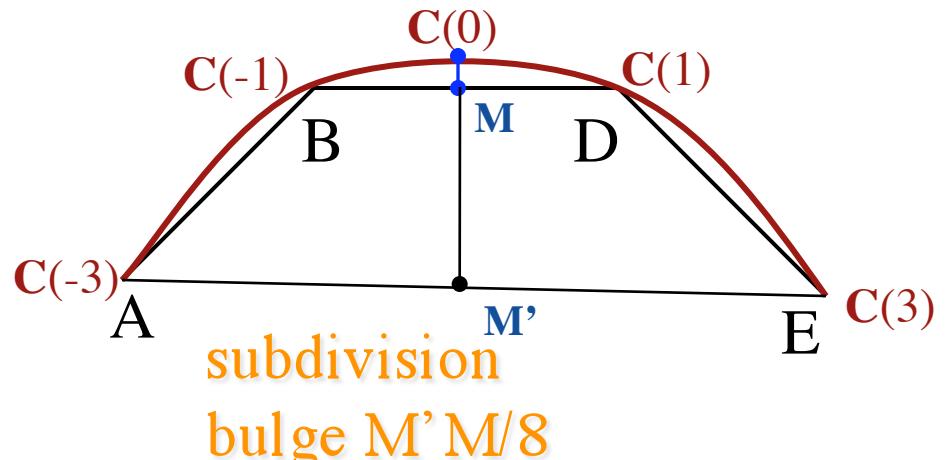
- $B=C(-1)=-a+b-c+d$
- $D=C(-1)=a+b+c+d$
- $A=C(-3)=-27a+9b-3c+d$
- $E=C(-3)=27a+9b+3c+d$
- $C=C(0)=d=M+M'M/8$



$$M=(B+D)/2=b+d \quad M'-9M=-8d$$

$$M'=(A+E)/2=9b+d$$

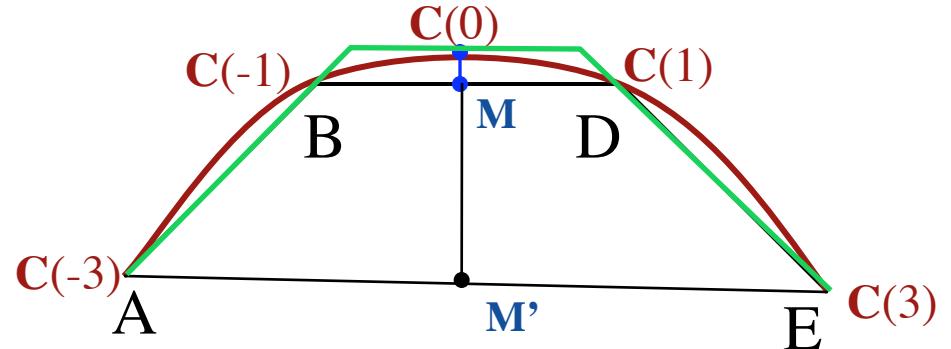
$$8M+M-M'=8d$$



# Implementing 4-point subdivision

- Design the construction of  $C = M + M'M/8$  using  $s(A,t,B)$
- Avoid computing  $M'$ 
  - instead extrapolate  $AB$  and  $ED$  and take the mid-point
- We will use this construction for smoothing key-frames motions

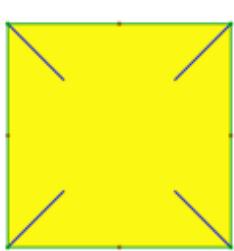
Compare with your neighbor



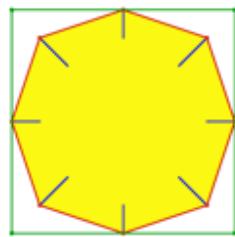
$s( s(A,9/8,B) , .5, s(E,9/8,D) )$

# 4-point = (r,t,u<sub>1</sub>)

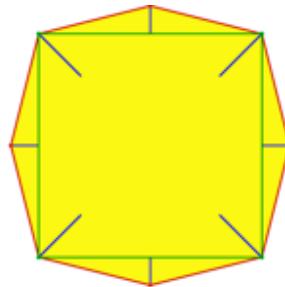
---



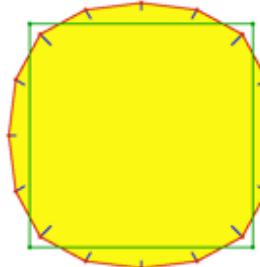
r



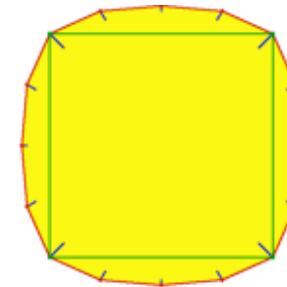
rt



rtu<sub>1</sub>



rtu<sub>1</sub>rt



rtu<sub>1</sub>rtu<sub>1</sub>

Refine (r) puts a mid-edge point C at M.

Surprise! (t,u<sub>1</sub>) produces the cubic-fit bulge M'M/8

Verify by inspection...

# Compare cubic B-spline & 4-point

---

- Which one interpolates the original vertices?
- Which one produces the smoothest curve?
  - When is this important?
- Can both have cusps (non smooth points)?
- How can you force the B-spline curve to go through a point?
- Which one has a convex-hull property?
  - When is this important?

# Cubic B-spline and four-points

- From James Koch

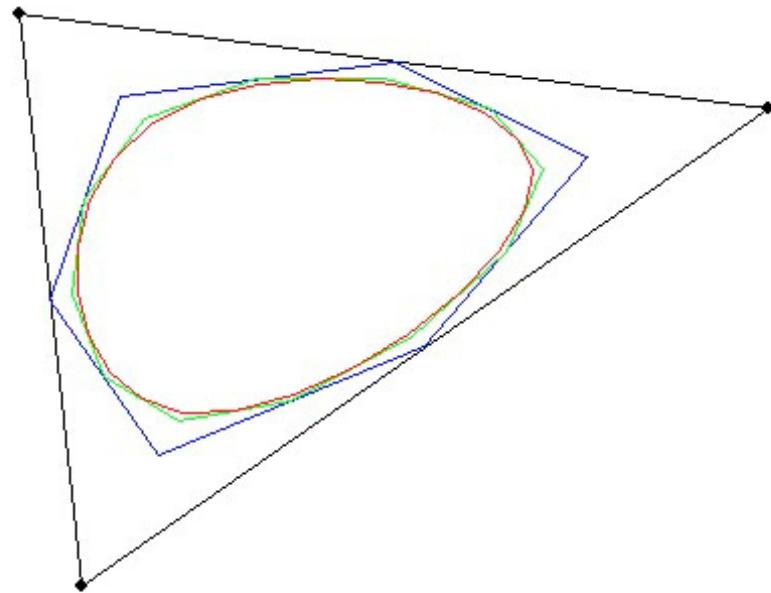


Figure 1. A triangle with several iterations of B-spline subdivision

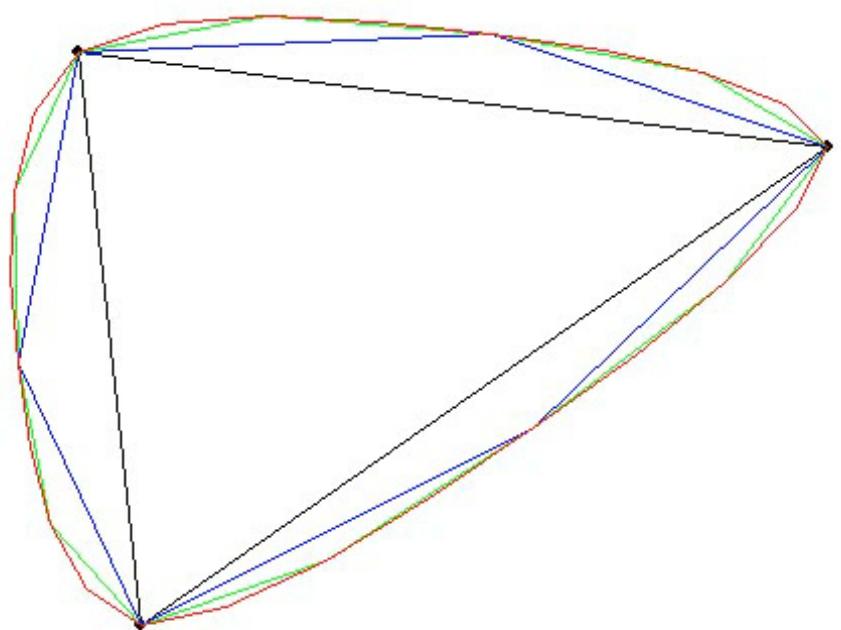
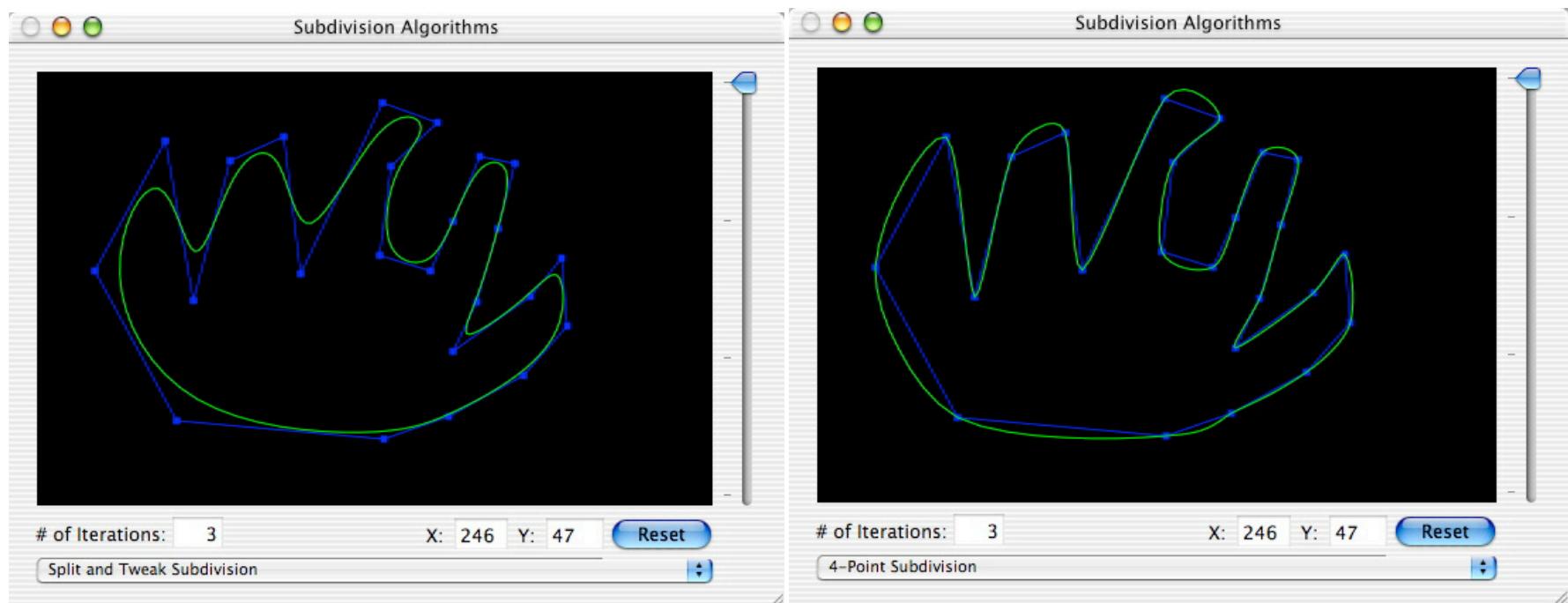


Figure 2. A triangle with several interations of four-point subdivision

# Which one is smoother?

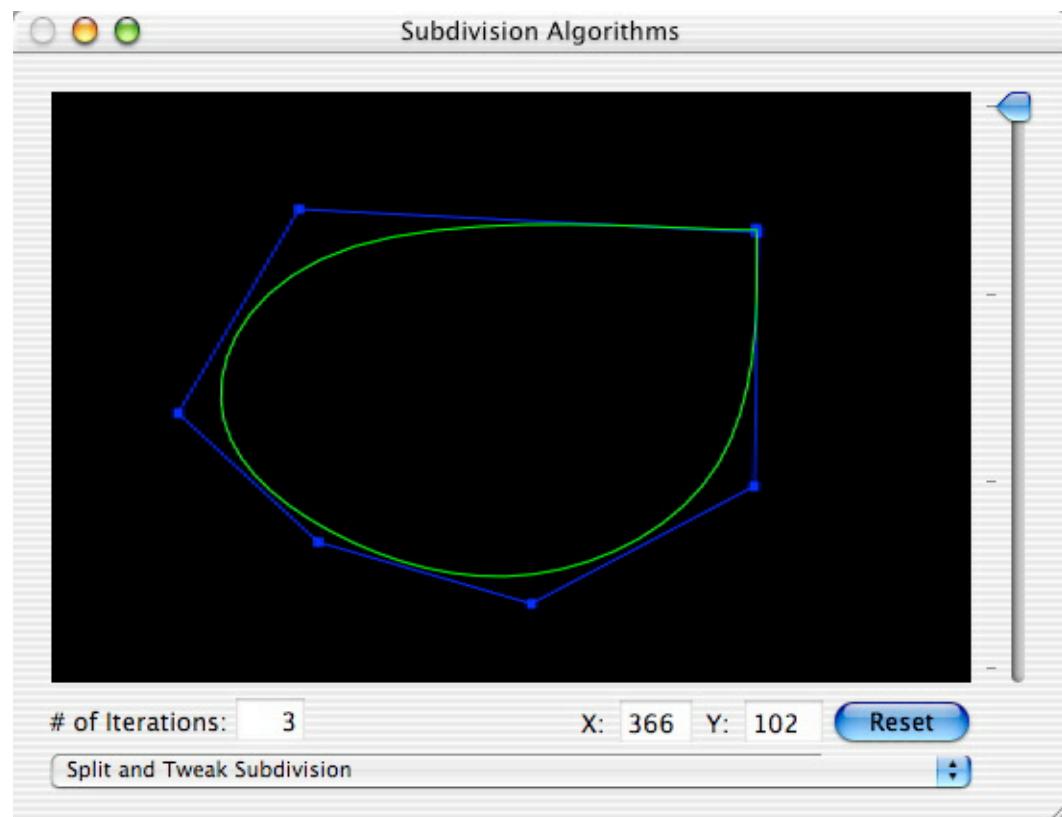
- By Stevie Strickland



# Making a sharp corner in a B-spline curve

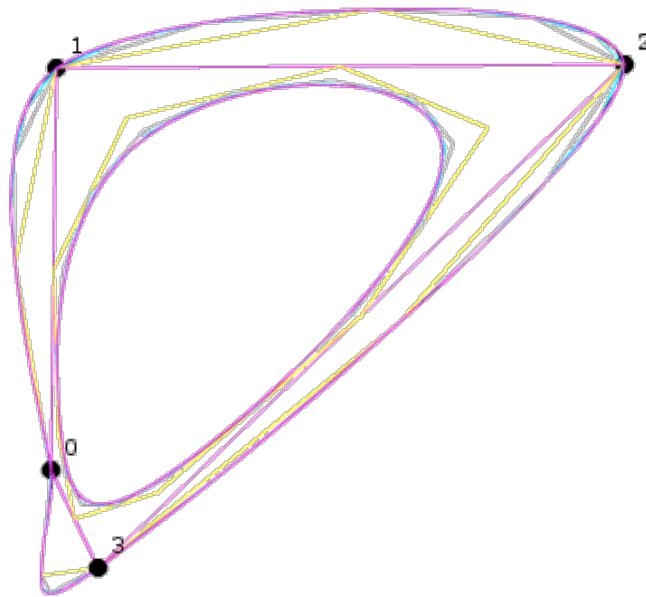
- By Stevie Strickland

- HOW?



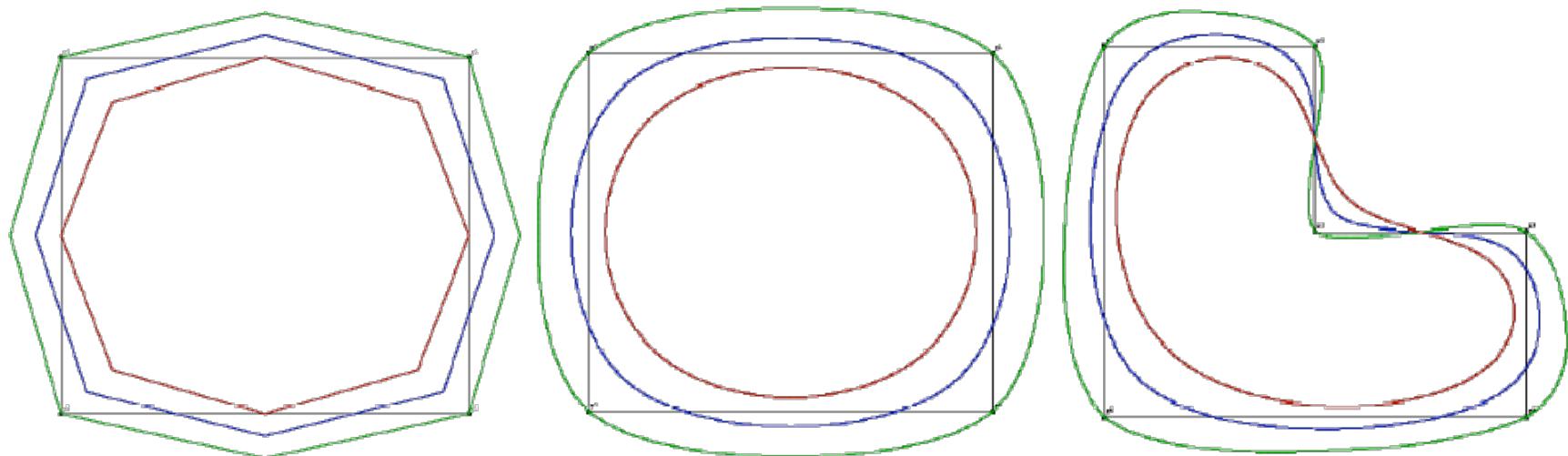
# A compromise

- B-spline subdivision rounds the corners and thus shrinks a convex shape
- 4-points subdivision bulges the edges out and hence grows a convex shape
- Both produce curves that are relatively far from the initial control polygon
- We may want a compromise
  - Closer to the polygon
  - Bulge less
  - Cut less



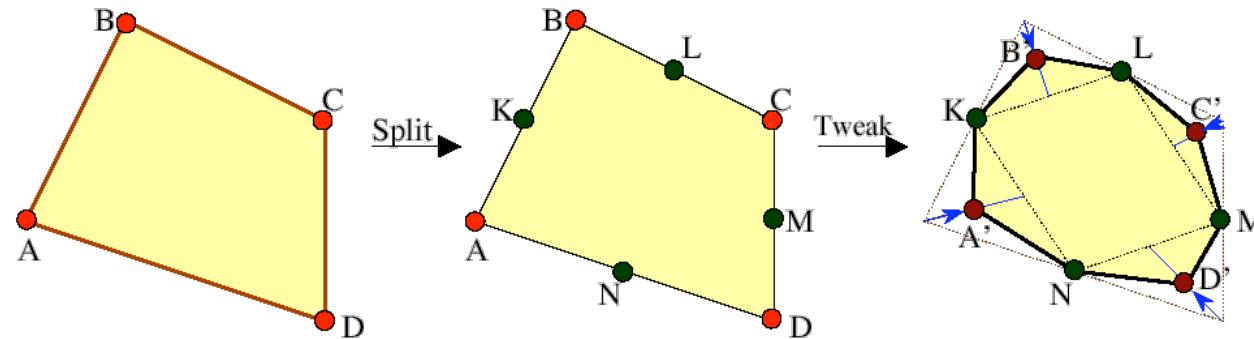
# Jarek's compromise: $J_4$

- Split each edge as before (in both schemes)
- Tweak old vertices by half of the B-spline displacement and the new vertices by half of the 4-point displacement

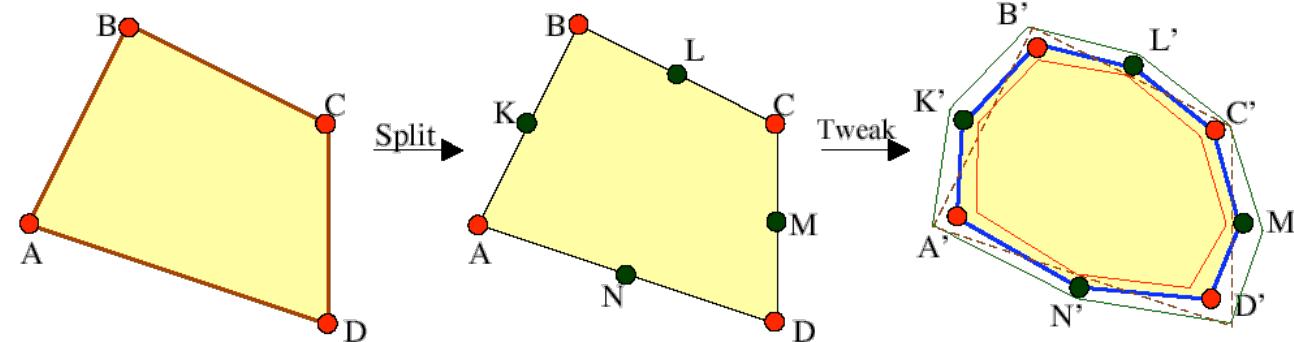


# 3 subdivision schemes

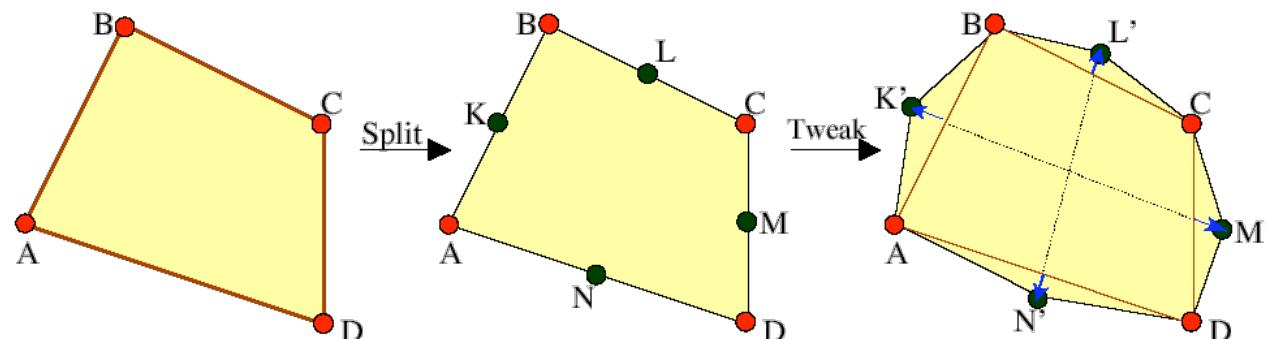
Bezier



$J_4$



4-points



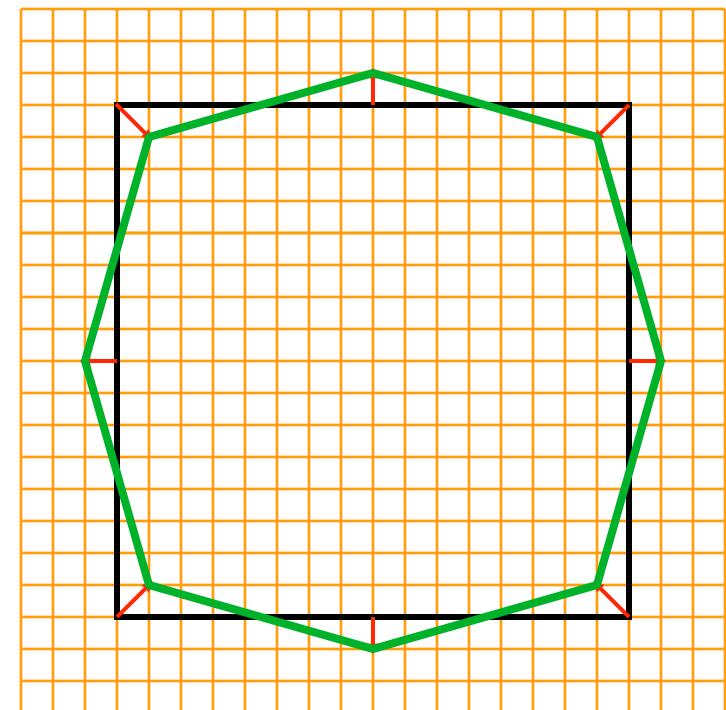
# Implementing $J_4$

---

1 - For each vertex  $v_i$ , compute the average  $\mathbf{a}_i$  of its neighbors and store a displacement vector  $\underline{\mathbf{D}}_i = (\mathbf{a}_i - \mathbf{v}_i)/8$ .

2 - For each edge  $(v_i, v_{i+1})$ , insert a new vertex  $\mathbf{m}_i = (v_i + v_{i+1})/2$ , and displace it by  $-(\underline{\mathbf{D}}_i + \underline{\mathbf{D}}_{i+1})/2$ .

3 - Displace each old vertex  $v_i$  by  $\underline{\mathbf{D}}_i$ .



# Three subdivision schemes

---

- B-spline (uniform cubic B-spline), most popular:
  - Introduces new vertices at edge mid-points (split)
  - Tucks in the old vertices by 1/2 the displacement towards the average of their new neighbors (tweak)
  - Very smooth result: piecewise cubic parametric formulation
  - Does not interpolate the original vertices
  - Shrinks convex regions
- 4-points:
  - Leaves the old vertices unchanged
  - Bulges the edges by displacing their mid-point away from the average of the neighbors (split & tweak)
  - Interpolates the original vertices
  - Grows convex regions
- $J_4$ 
  - Combines half of the tweaks of the other two
  - Final curve is closer to the original polygon

# A unifying theory: Js refinements

- $J_s = (r, t, u(s/8))$

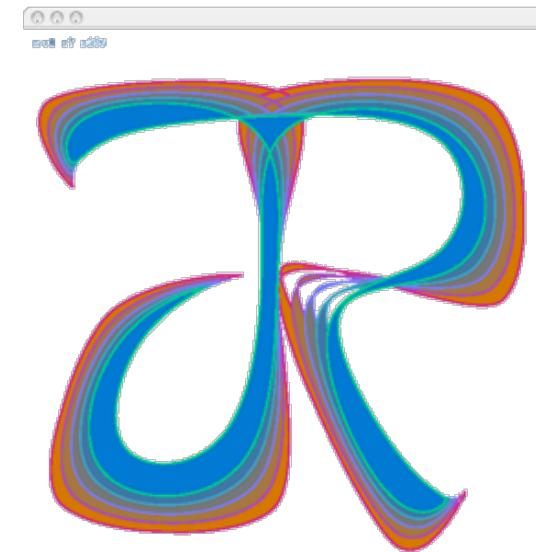
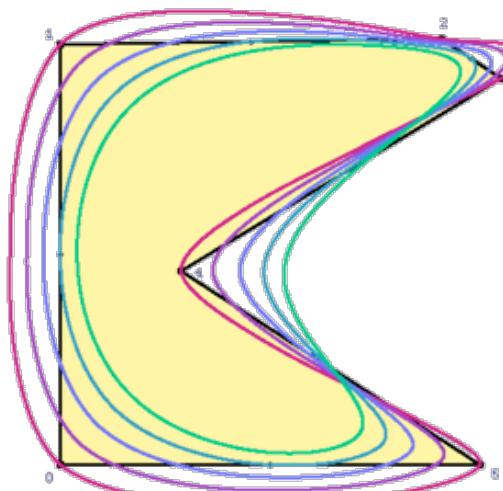
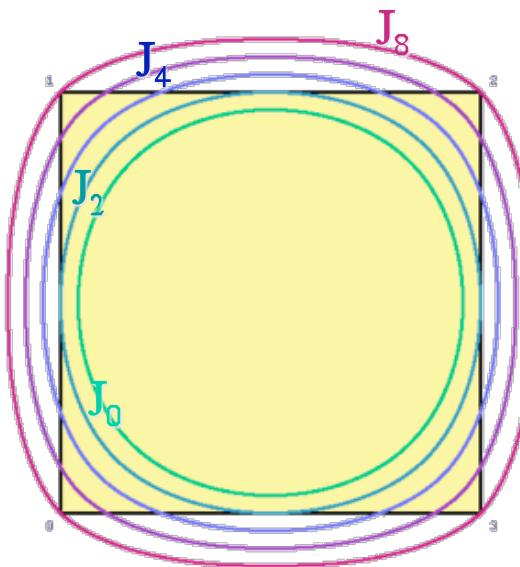
$J_0$  = cubic B-spline

$J_2$  = osculates edges (smoother than quadratic B-spline)

$J_4$  = in general preserves area well

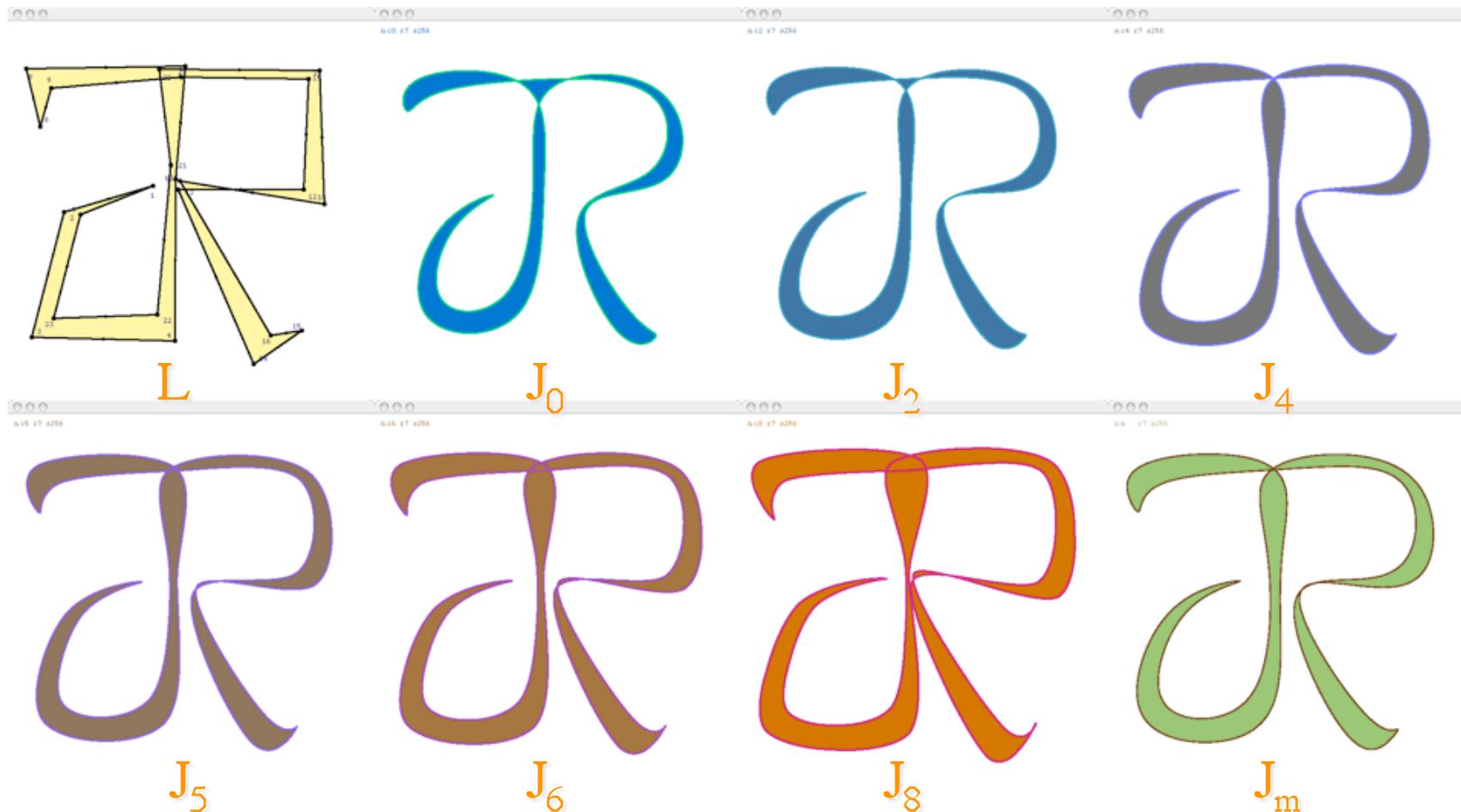
$J_5$  = reduces discrepancy with L (good for LoD)

$J_8$  = 4-point (interpolating)



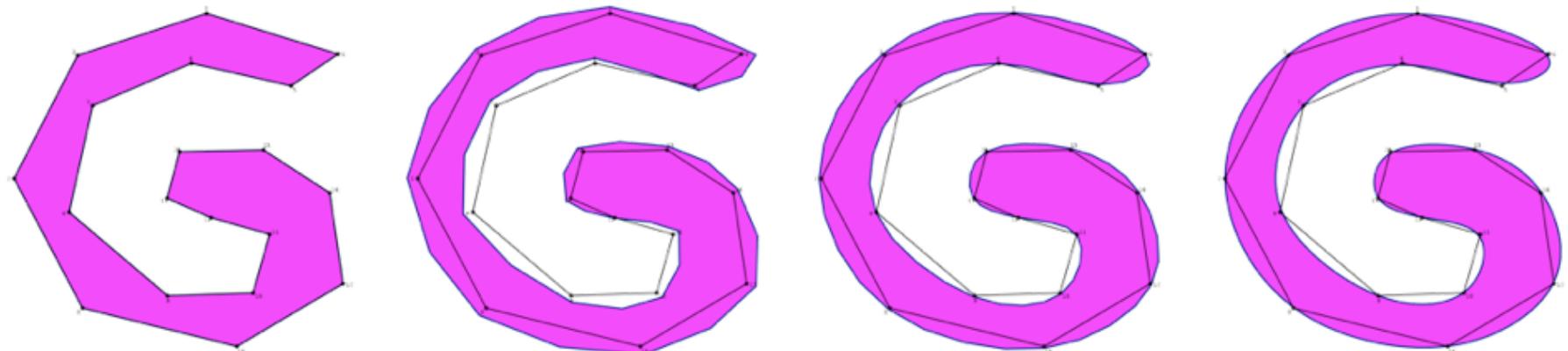
# Comparing $J_s$ schemes

$J_m = J_2, J_0, J_{-2}, J_{-2}, J_{-2} \dots$



# Retrofitting

- A new control polygon may be computed so that the subdivision curve passes through, or close to, the original control vertices.



# Continuity measures for paths and curves

---

- Parametric (path)
  - $C^0$ : Continuous trajectory at finite speed ( $\infty$  acceleration?)
  - $C^1$ : Velocity is a continuous function of parameter t
  - $C^2$ : Acceleration is a continuous function of parameter t
  - $C^3$ : Jerk is a continuous function of parameter t
  
- Geometric (curve)
  - $G^0$ : Continuous curve, no gap
  - $G^1$ : Tangent is a continuous function of arclength
  - $G^2$ : Curvature is a continuous function of arclength

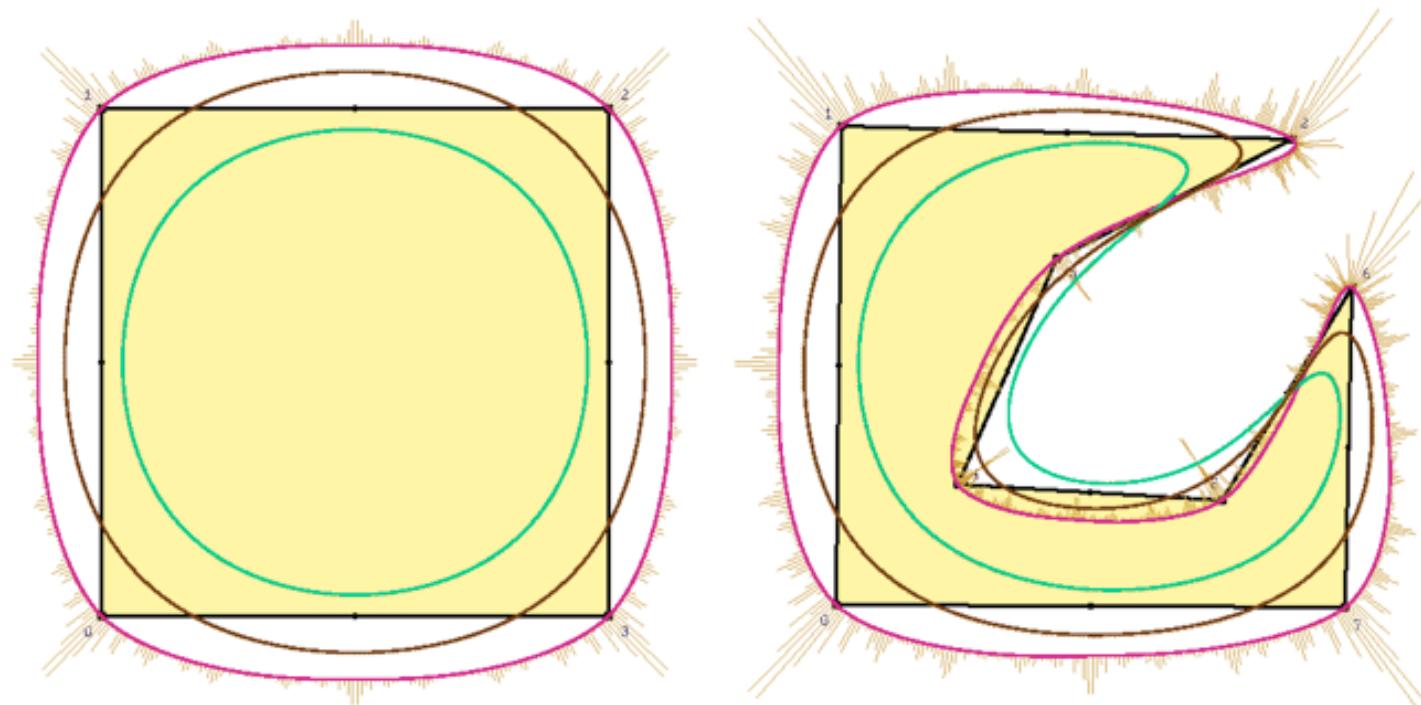
# Smoothness

- Visualize (lack of smoothness) using jerk

$J_0$ : (no visible jerk) second degree (curvature) continuous, I.e.,  $C^2$

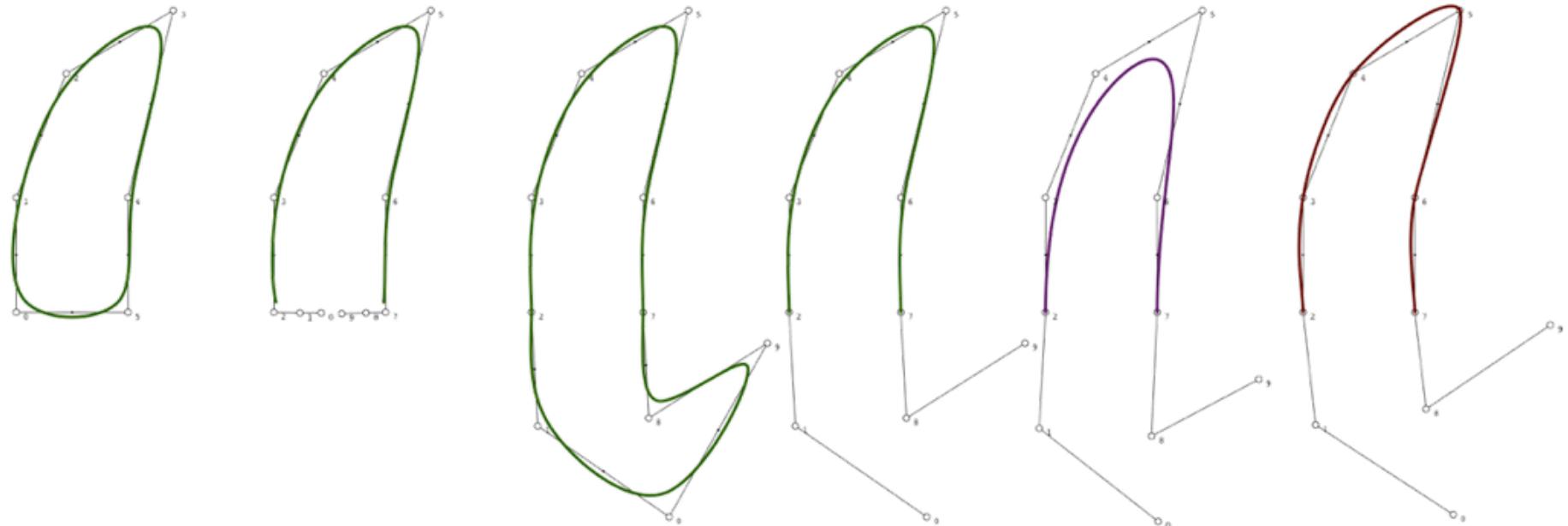
$J_8$ : (total jerk) first degree (tangent) continuous, I.e.  $C^1$

$J_m$  (brown) is  $C^3$



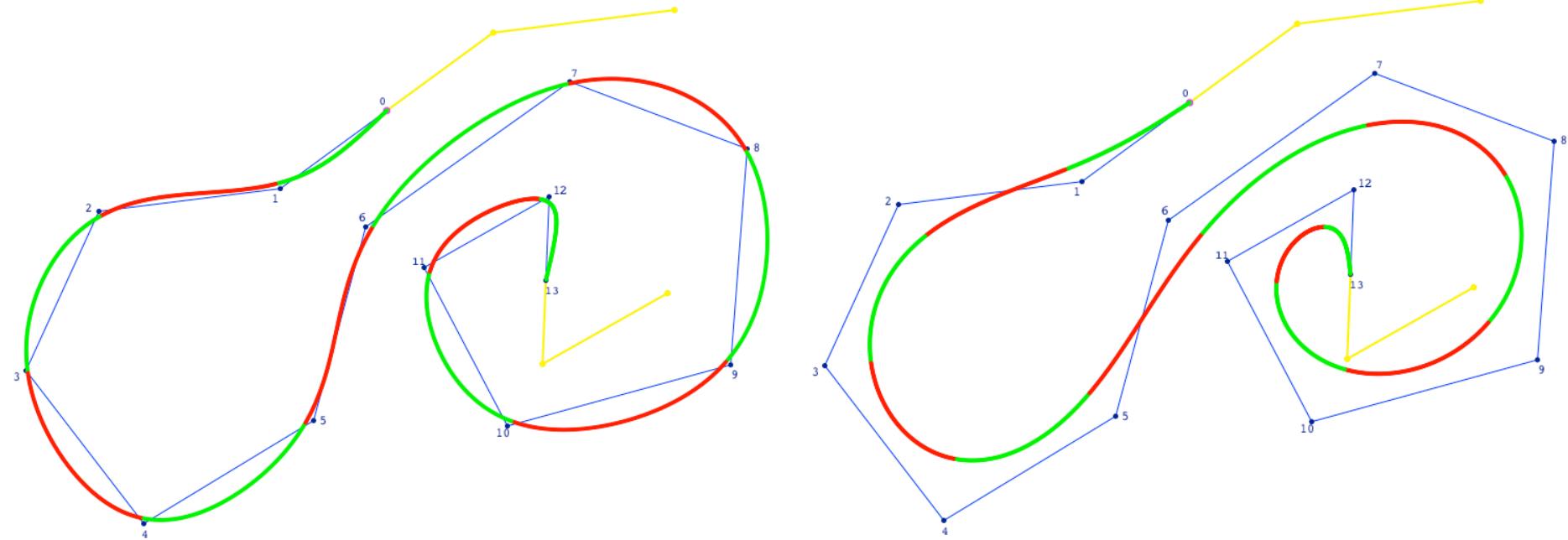
# Open curves

- Temporarily add (guesses of) 2 extra points on each end
- Subdivide as before
- Render curve, except 5 spans
- Remove the extra points



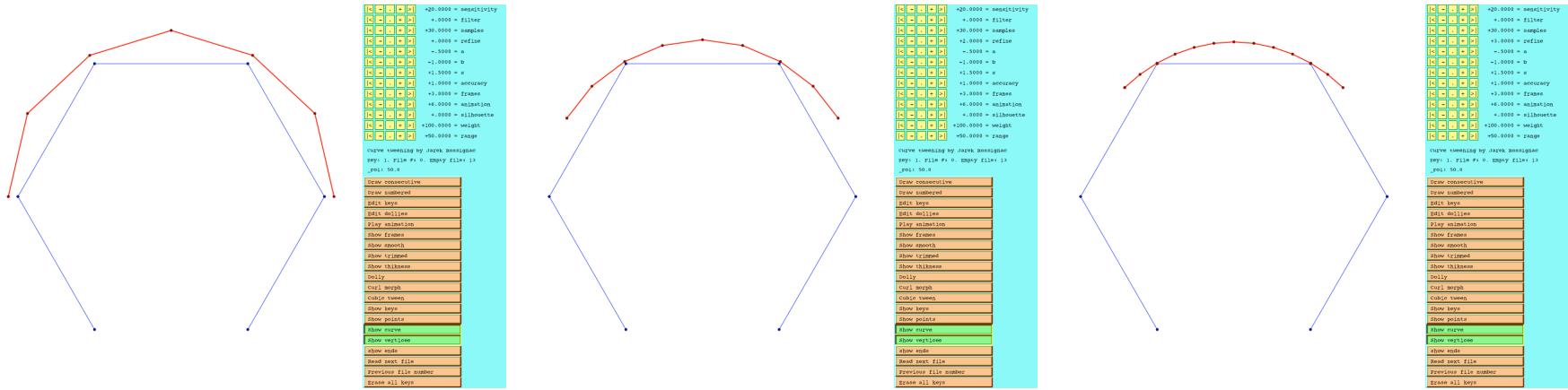
# Spans

- Subdivision bends each original edge into a span



# Per span subdivision

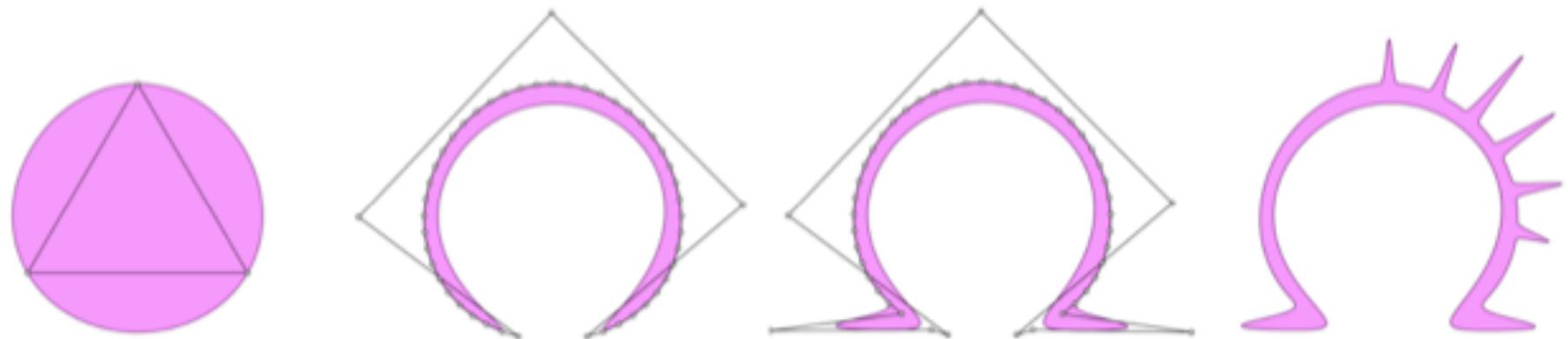
- You do not need to subdivide the entire curve at the same time
- You can subdivide one span at a time (need 6 control vertices)



- Or use a sliding window (ringing), which requires storing a ring of only 4 vertices per level of subdivision (see paper)

# Multiresolution design

- At each level of subdivision, let the user specify adjustment vectors for some of the vertices
- Retain these vectors and apply them during rendering
- This lets the user control large and small features with few strokes



# What about curves in 3D?

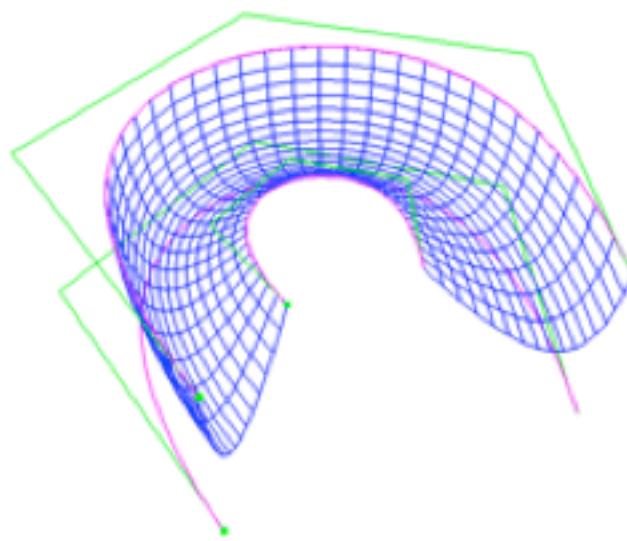
---

- The smoothing and subdivision procedures work for any dimension
  - 1D for speed control in P1a
  - 2D for curves, paths, and trajectories of cars
  - 3D for curves, paths, and trajectories of airplanes
  - 2D for designing and smoothing trajectories on surfaces (skier)
  - In the space of screws for smoothing key-frames motions...

# What about animation?

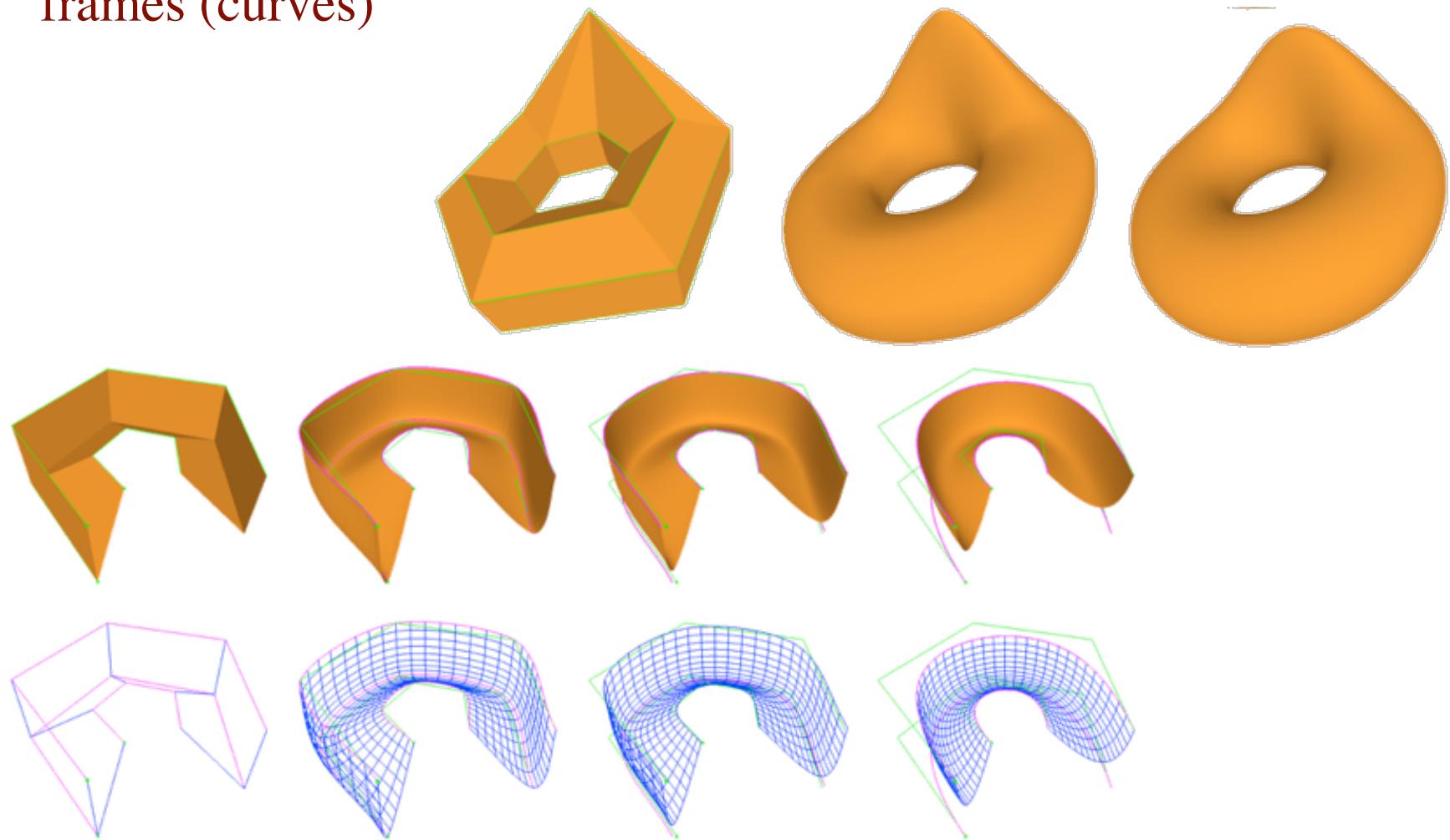
---

- At each frame, draw subdivision curve
- Its control vertices move with time
- Their trajectories are each defined as a subdivision curve



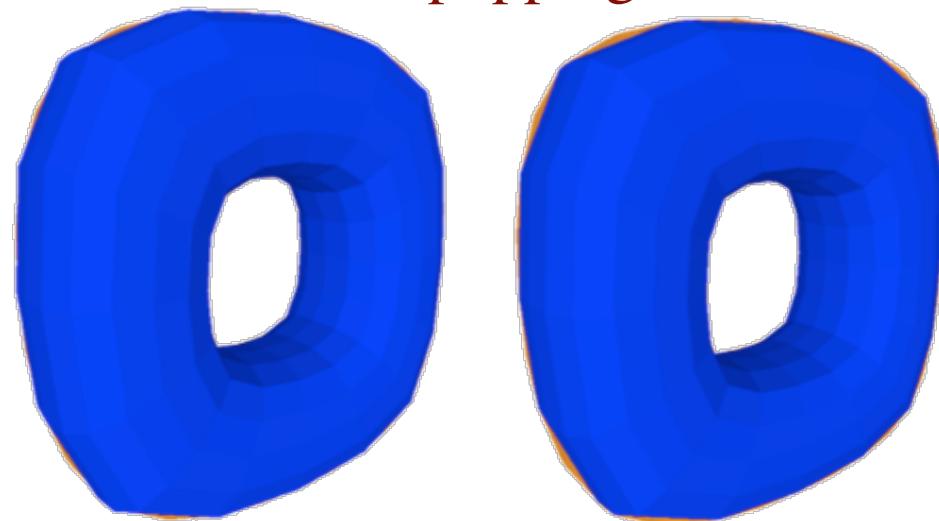
# What about surfaces?

- Same as curve animation, but draw quads between consecutive frames (curves)



# Adaptive resolution

- The level of subdivision of a curve or surface should be adjusted to the resolution and the size of its projection on the screen
- When the size changes, as we switch resolution, a popping may be perceived.
- Jarek subdivision exhibits less popping than four-point and cubic B-spline
- Geomorphs are also used to smoothen the popping over several frames



# Examples of quiz/exam questions

---

- Define/illustrate curve, path, motion, trajectory
- Explain interpolating and approximating
- Constructions of velocity, normal, acceleration, radius of curvature, jerk
- Implementation of r, k, d, t, u, u(x) steps
- Write polyloop smoothing using them
- Why tuck-tuck is better than tuck
- Cubic fit formulation for tuck-tuck
- Difference between smoothing and subdivision
- Subdivision formula for quadratic B-spline
- Subdivision formula for cubic B-spline
- Spans, local control, open curves
- Bezier control polygons from polyloop
- Cubic Bezier evaluation and subdivision
- Convex hull of cubic bezier curve
- 4-point subdivision as cubic fit
- Draw 4-points, jarek, and cubic B-spline for a simple polyloop by hand
- General formulation for jarek subdivision  $J_s$
- Properties of  $J_0$ ,  $J_3$ ,  $J_4$ ,  $J_5$ ,  $J_8$