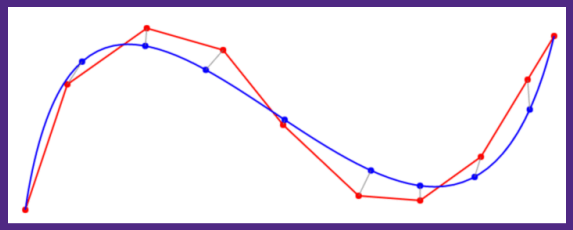
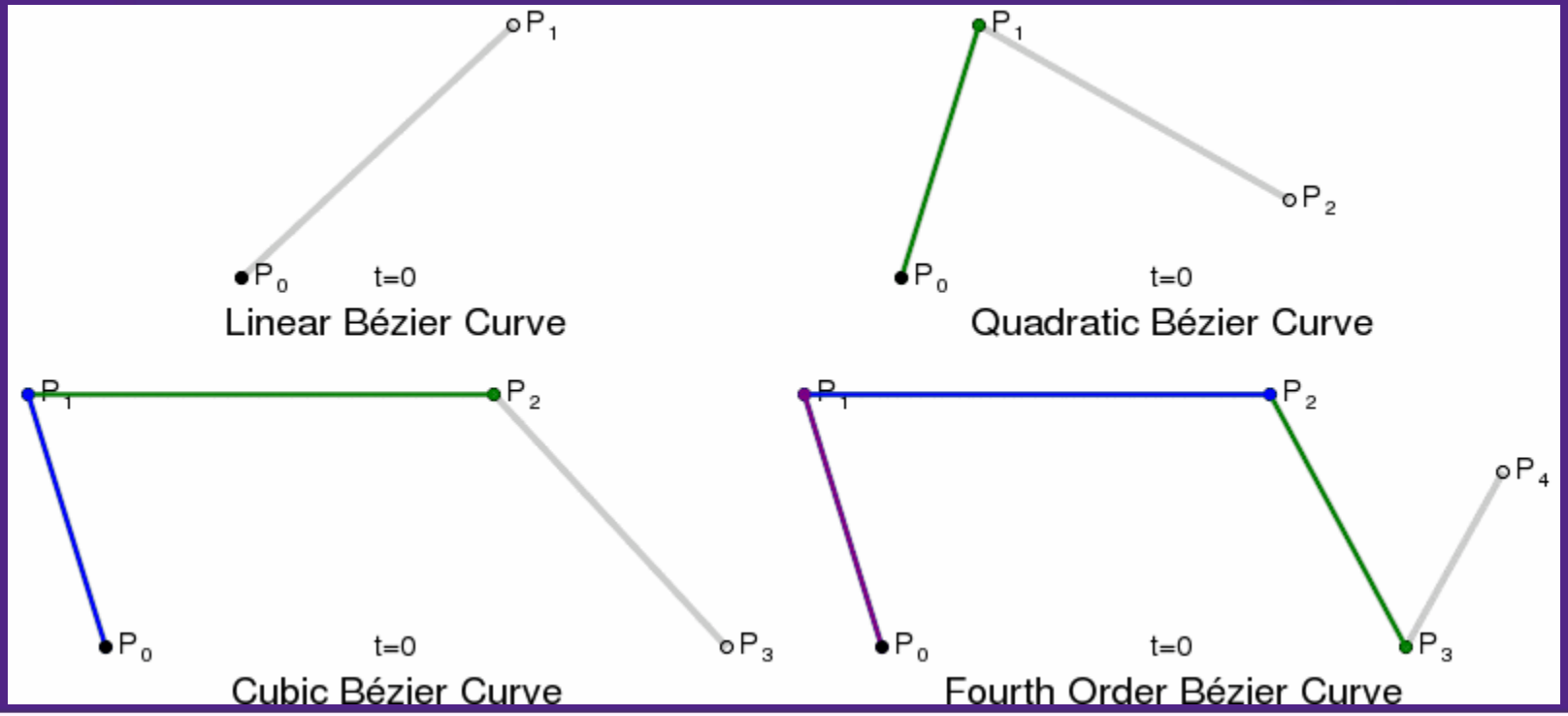


Computer Graphics I: Curve & Surface Design



Curve and Surface Design (2D Focused)

- The end of 2D drawing has two main, and related, topics: tweens and splines.

Lerps and Tweens

- A **linear interpolation** is a simple operation.
- Given two vectors A and B (positions, sizes, attributes, etc.) we can define a function which smoothly blends one vector into the other.
- The lerp defines a simple function P(t) where P(0) = A and P(1) = B:

$$P(t) = A(1 - t) + Bt$$

Lerps and Tweens

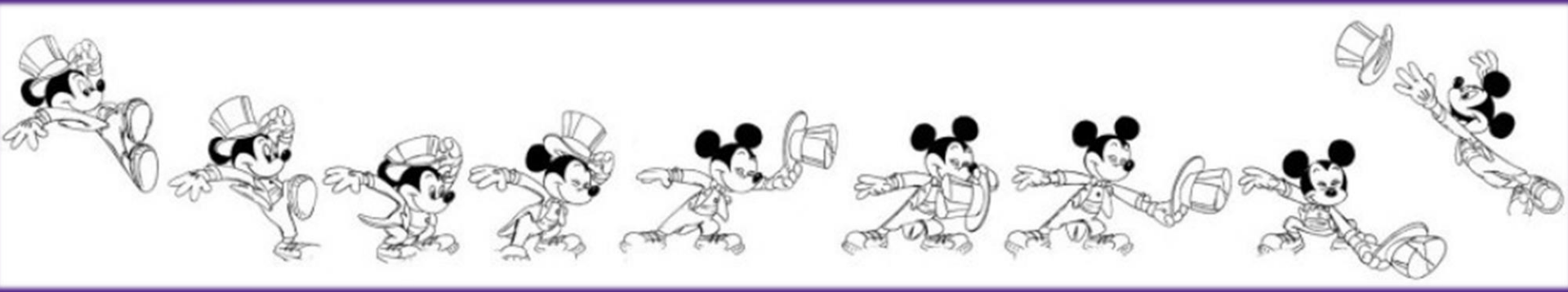
- Another way to think of this blending is that linear interpolation is providing a value that is some weighted average of A and B.
- At t = 0 the value is 100% A.
- At t = 1 the value is 100% B.
- At t = 0.5 the value is exactly half A and half B.
- When a lerp is applied to positions, it is called tween for in-between.

Lerps and Tweens

- Tweens are particularly useful for animation.
- For a particular object with N vertices, we can define tweens for each vertex independently.
- Applying each vertex's tween simultaneously gives the impression of the whole figure being animated.
- Of course, we must define many **key frames** which are the "end points" of each tween.
- We tween from frame i to i + 1 for t = 0...1.
- Then, we tween from frame i + 1 to i + 2 for t = 0...1 again.
- Repeat forever and you have a **key frame** animation.

Lerps and Tweens

- Repeat forever and you have a **key frame** animation.



Curve and Surface Design

- Parametric curves may be used to describe the motion of objects or that of the *synthetic camera*.
- For these curves to describe smooth motions of the camera, they must have a first order derivative that is also continuous.
- If a user wants the camera to look in the direction of the motion, then a vector tangent to the curve at the position of the camera must be computed in such a way as to determine the gaze direction of the camera.
- Suppose the trajectory of the camera is described by the following parametric curve: $p(t)=(X(t), Y(t), Z(t))$

Curve and Surface Design

- The velocity vector is tangent to this curve and can be used to determine the gaze direction:

$$\vec{v}(t) = \frac{dP(t)}{dt} = \left(\frac{dX(t)}{dt}, \frac{dY(t)}{dt}, \frac{dZ(t)}{dt}\right)$$

- It is then clear that this curve design mechanism requires the parametric curve to be differentiable and that its derivative also be differentiable, such that the gaze vector moves smoothly along the camera path.

Tweens in OpenGL

- Consider two polyline drawings (or any set of primitives) with the same number of vertices.
- We can define a tween between them by creating a bijection (a one-to-one mapping between the vertices)
- Then, we lerp each vertex independently.
- Let's give it a try.

```
std::vector<float> star = {0.0f, 10.0f, 2.5f, 2.5f, 10.0f, 2.5f, 4.0f, -2.
    5f, 7.0f, -10.0f, 0.0f, -5.0f, -7.0f, -10.0f, -4.0f, -2.5f, -10.0f, 2.5
    f, -2.5f, 2.5f, 0.0f, 10.0f};

float v1x = 10* 0.5, v1y = 10*1.5388, v2x = 10*1.30902,
      v2y = 10*0.951056, v3x = 10*1.618034, v3y = 0;
std::vector<float> poly={v1x, v1y, v2x, v2y, v3x, v3y, v2x, -v2y, v1x, -v1
    y, -v1x, -v1y, -v2x, -v2y, -v3x, v3y, -v2x, v2y, -v1x, v1y, v1x, v1y};

glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-20, 20, -20, 20, -1, 1);

float deltaT = 0.005, t = 0, vx, vy;

/* Loop until the user closes the window */
while (!glfwWindowShouldClose(window))
{
    //...
    glBegin(GL_LINE_STRIP);
    for (int i = 0; i < star.size(); i +=2) {
        vx = lerp(star[i], poly[i], t);
        vy = lerp(star[i+1], poly[i+1], t);
        glVertex2f(vx, vy);
    }
    glEnd();

    if (t > 1.0f || t < 0.0f) {
        deltaT *= -1.0f;
    }
    t += deltaT;
    //...
}
```

Non-linear Tweens

- The classic lerp equation A(1 − t) + B(t) can be viewed as a split of unity.
- For any t between 0 and 1, (1 − t) + t = 1.
- This is a two-way partition.
- What if we partition 1 in three ways?
- $1 = ((1 – t) + t)^2$
- $1 = (1- t)^2 + (2 – t)t + t^2$

Non-linear Tweens

- Now, instead of a linear interpolation we can create a smooth quadratic approximation between three points, A, B,C.
- This smooth interpolation is a Bezier curve of degree 2 and can be described by a parametric formula P(t).

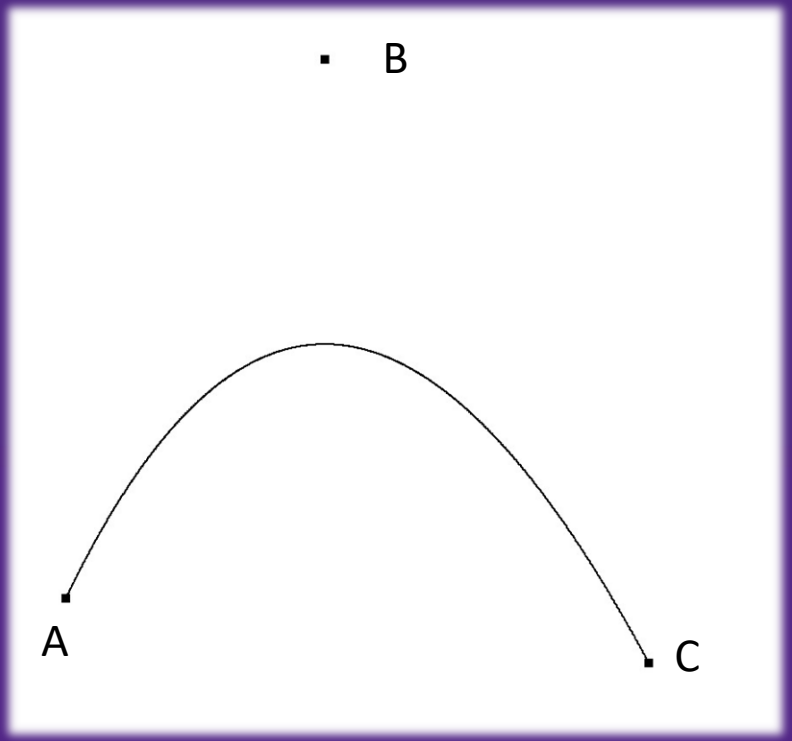
$$P(t) = (1 - t)^2A + 2t(1 - t)B + t^2C$$

Non-linear Tweens

- In this case, we again have P(0) = A and P(1) = C.
- Moreover, there is no value t′ such that P(t′) = B.
- Rather, B acts as a control point which influences the interpolation between A and C.
- One can feel that B acts as a sort of “gravity” for the interpolation.

Non-linear Tweens

- One can feel that B acts as a sort of “gravity” for the interpolation.



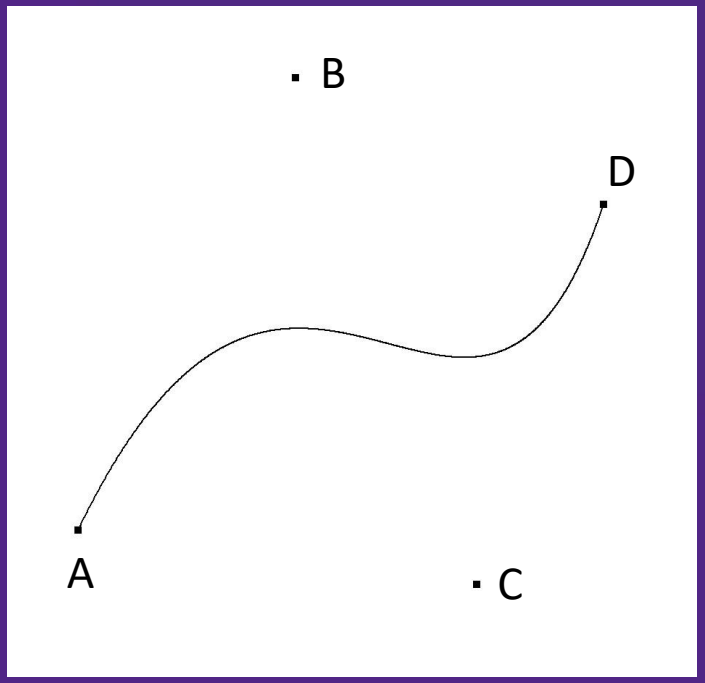
Non-linear Tweens

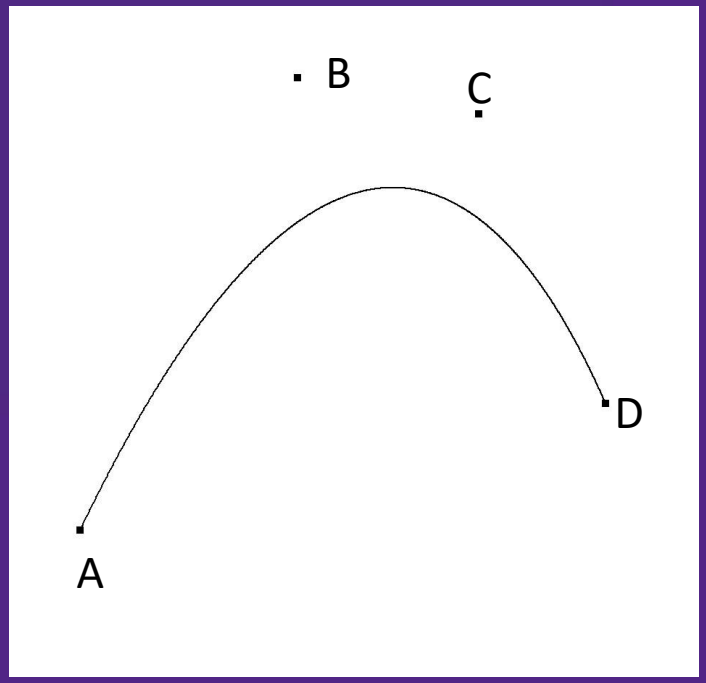
- Cubic interpolation thus occurs between four points, A, B, C, D.
- Again, the idea is to define a function so that P(0) = A and P(1) = D, while B and C act as control points.

$$P(t) = (1 - t)^3A + 3(1 - t)^2tB + 3(1 - t)t^2C + t^3D$$

Non-linear Tweens

$$P(t) = (1 - t)^3A + 3(1 - t)^2tB + 3(1 - t)t^2C + t^3D$$





Designing Polynomial Bezier Curves

- We use control points that are usually specified by a user, and proceed to fit a curve to these points.

- There are two general methods of accomplishing this:
  - a curve interpolates the points if the curve passes through all the control points, and
  - a curve approximates the points if the curve contains the start and end points only.

Designing Polynomial Bezier Curves

- Suppose we have three control points P_0 , P_1 , and P_2
- Let's find an approximating curve to those points.
- First, let's define a parametric form for the first segment from P_0 to
- P_1 and from P_1 to P_2 :

$$A(t) = (1-t)P_0+t\,P_1$$

$$B(t) = (1-t)P_1+t\,P_2$$

- We can define a new segment between $A(t)$ and $B(t)$:

$$P(t) = (1-t)A+tB(t)$$

Designing Polynomial Bezier Curves

- This results in $P(t)$ approximating the three control points for $t \in [0,1]$. $P(t)$ has a parametric representation, obtained as:

$$P(t) = (1-t)A(t) + t\,B(t)$$

$$= (1-t)((1-t)P_0 + t\,P_1) + t((1-t)P_1 + t\,P_2)$$

$$= (1-t)^2 P_0 + 2(1-t)t\,P_1 + t^2 P_2$$

Designing Polynomial Bezier Curves

- This parametric form is a parabola and approximates the control points. If we had **four** control points, a similar reasoning would lead us to the following expression for their approximation:

$$P(t)=(1-t)^3 P_0+3(1-t)^2 t\, P_1+3(1-t)t^2 P_2+t^3 P_3$$

- which is a cubic polynomial in t approximating the four control points. These coefficients to the control points are known as Bernstein's polynomials:

$$B^3_0(t) = (1-t)^3$$

$$B^3_1(t) = 3(1-t)^2t$$

$$B^3_2(t) = 3(1-t)t^2$$

$$B^3_3(t) = t^3$$

Designing Polynomial Bezier Curves

- It is important to note that Bernstein's polynomials sum to unity within $t \in [0,1]$
- Hence by rewriting the parametric approximating function for four points as,

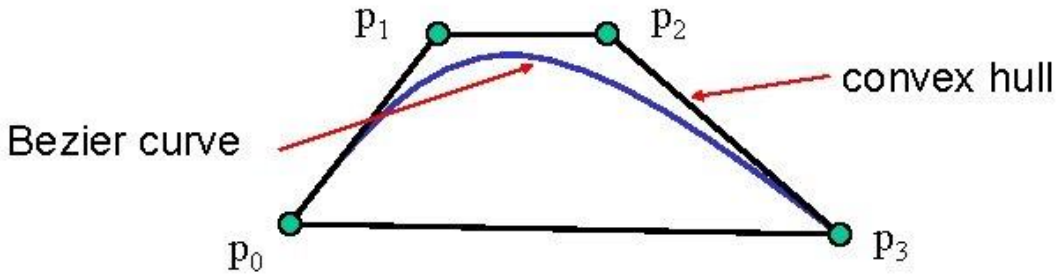
$$p(t)=B^3_0P_0+B^3_1P_1+B^3_2P_2+B^3_3P_3$$

- it becomes clear that it represents a convex affine combination of the four control points.
- Bernstein's polynomials used in this context are referred to as blending functions, as they blend the control points into a smooth approximation.

Designing Polynomial Bezier Curves

Convex Hull Property

- The properties of the Bernstein polynomials ensure that all Bezier curves lie in the convex hull of their control points
- Hence, even though we do not interpolate all the data, we cannot be too far away



Angel: Interactive Computer Graphics 4E © Addison-Wesley 2005 9

Designing Polynomial Bezier Curves

- When L+1 points are present, then the Bezier curve is given by:

$$\sum_{k=0}^{L} B_k^L(t) P_k$$

where

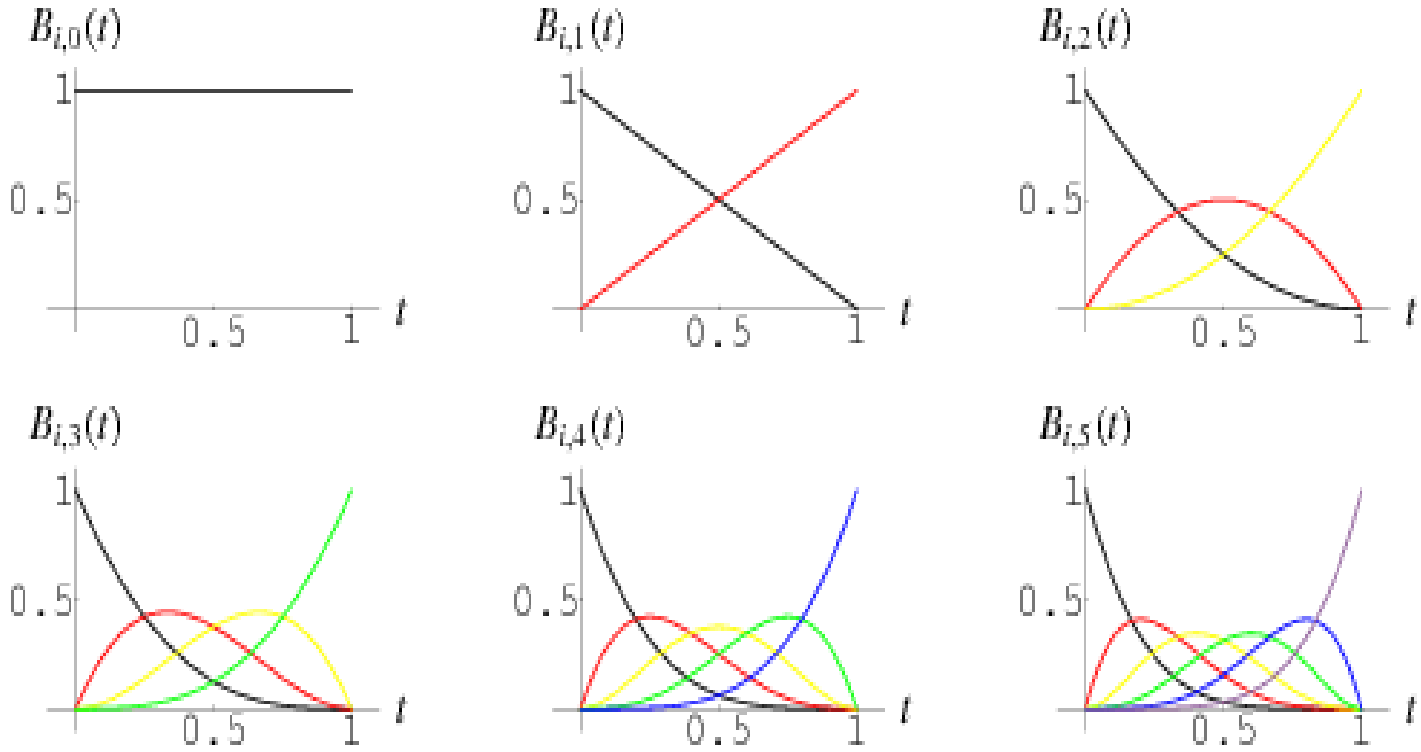
$$B_k^L(t) = \left(\frac{L!}{k!\,(L-k)!}\right)(1-t)^{L-k} t^k$$

for k < L+1

Designing Polynomial Bezier Curves

- There are multiple properties for the Bezier curves:
  - The first and last control points of a Bezier curve are always interpolated
  - A Bezier curve is affine-invariant
  - A Bezier curve is always within the boundaries defined by the convex hull of the control points

Designing Polynomial Bezier Curves



Designing Polynomial Bezier Curves

- Derivatives of Bezier curves exist and they are continuous:

$$P'(t) = L \sum_{k=0}^{L-1} B_k^{L-1}(t) \Delta P_k$$

where $\Delta P_k = P_{k+1} - \Delta P_k$

Designing Polynomial Bezier Curves

- While Bezier curves are useful for some applications, they suffer from problems in other types of use.
- For instance, the exponents involved tend to be large when one has multiple control points.
- Additionally, local control of the curve is hard to achieve since the entire set of control points influence the curve at any point (except at t=0 and t=1).

Designing Polynomial Bezier Curves

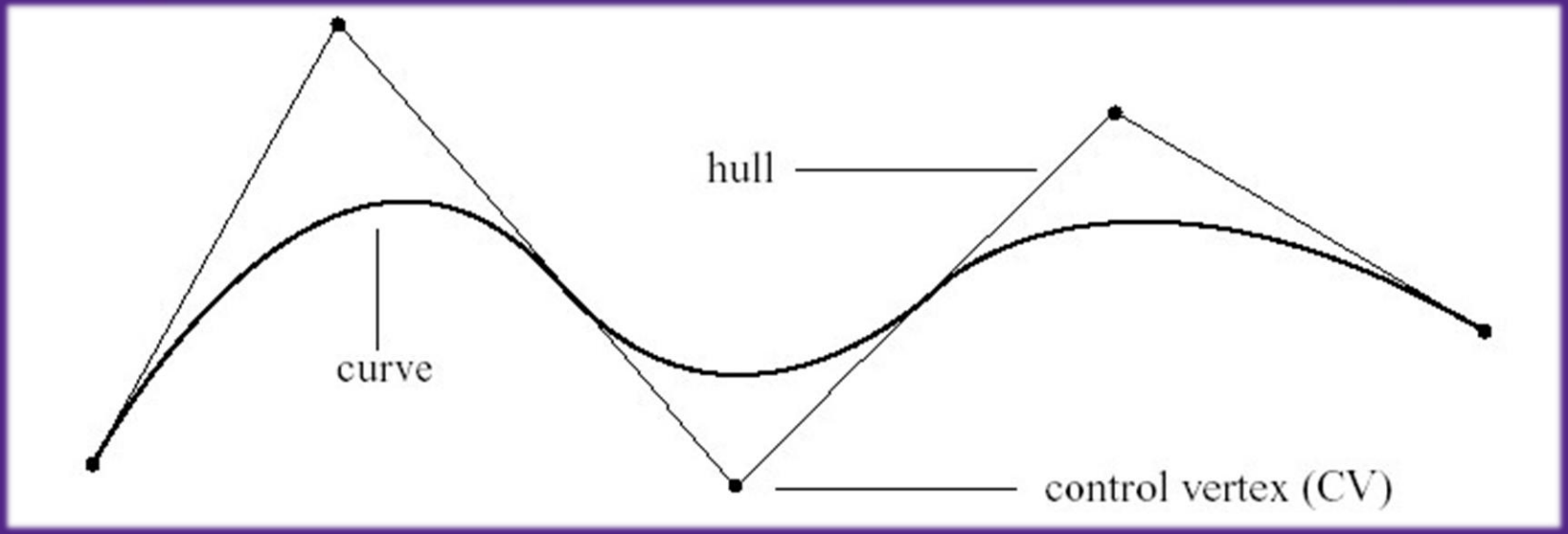
- It is useful to define a set of blending functions that possess the properties of Bezier curves, but that also provide local control over the curve.
- That is to say, we would like the influence of control points to be local at any point on the curve.

Splines

- Consider the points in space: A, B, C, D, E where we want each pair to be connected.
- As polylines, this is easy, just draw the line segments (A, B), (B, C), (C, D), (D, E). But what if we want to use Bezier curves to connect each pair of points?
- This problem needs to be solved in areas like computer-aided design, different typefaces, and every image/video manipulation program.
- The pen tool of photoshop, for example, draws splines.

Splines

- In graphics, the goal of splines is to create a curve in space which connects many dots in sequence.
- Basically a polyline, but multiple curves instead of multiple line segments.



Splines

- Splines are really just functions defined piecewise by polynomials.
- A **Bezier Spline** is a spline where each piece is a Bezier curve.
- Consider a slightly formal definition.
- We look to define a continuous function S : [a, b] 7→ ***R***.

Splines

- Formally, we say that S ∈ C0, the set of continuous functions.
- The interval [a, b] will be split into many sub-intervals, and each interval defined by a particular polynomial.
- Since S should be continuous, the intermediate end points of the polynomials must be equal.
- Let's say we have k such uniform subintervals. Thus, we have [t0, t1), [t1, t2), . . . , [tk−1, tk] with $a = t_0$ and $b = t_k$.
- We will define a Bezier curve Pi(t) for each interval.

Splines

$$S(t) = \begin{cases} P_1(t), a \leq t < t_1 \\ P_2(t), a \leq t < t_2 \\ P_3(t), a \leq t < t_3 \\ \vdots \\ P_k(t), a \leq t < t_k \end{cases}$$

- Since each Bezier curve is usually defined on the interval [0,1], we must do a little more work.
- We must scale the range $[t_i, t_i+1]$ to [0, 1]. That's quite easy. For Pi(t) let $t' = \frac{t-t_i}{t_{i+1}-t_i}$ and then compute $P_i(t)$

Splines

- But, in practice, particularly in computer graphics graphics where we just use Bezier splines to draw curves, we can just let the interval for each P_i be [0, 1].
- Let's do an example.
- Consider the points A, B, C that we want connect using cubic Bezier curves. One from A to B, and another from B to C. Without any control points,

Splines

- We would just use straight lines to connect them. This is the same result as letting each control point equaling the closer end point.
- We can draw each piece of the spline independently and simultaneously using t in the range [0, 1].

$$P_1(t) = (1 - t)^3 A + 3(1 - t)^2 t c_1 + 3(1 - t) t^2 c_2 + t^3 B$$

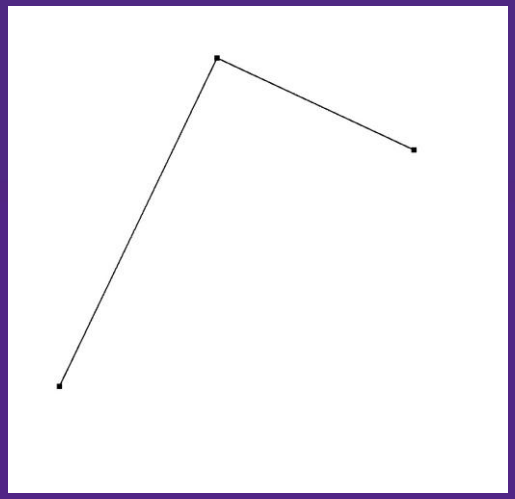
$$P_2(t) = (1 - t)^3 B + 3(1 - t)^2 t c_3 + 3(1 - t) t^2 c_4 + t^3 C$$

Splines

- We can draw each piece of the spline independently and simultaneously using t in the range [0, 1].

$$P_1(t) = (1 - t)^3 A + 3(1 - t)^2 t c_1 + 3(1 - t) t^2 c_2 + t^3 B$$

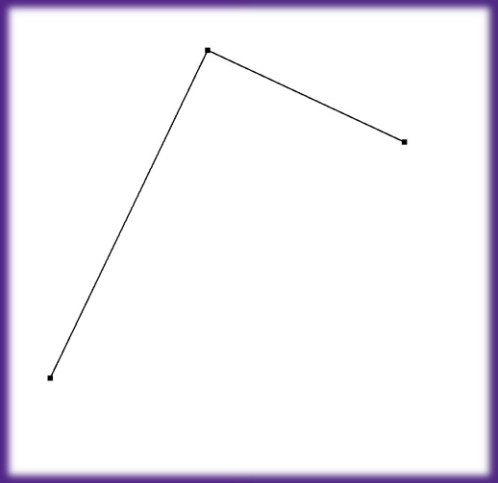
$$P_2(t) = (1 - t)^3 B + 3(1 - t)^2 t c_3 + 3(1 - t) t^2 c_4 + t^3 C$$

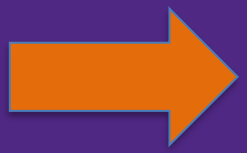


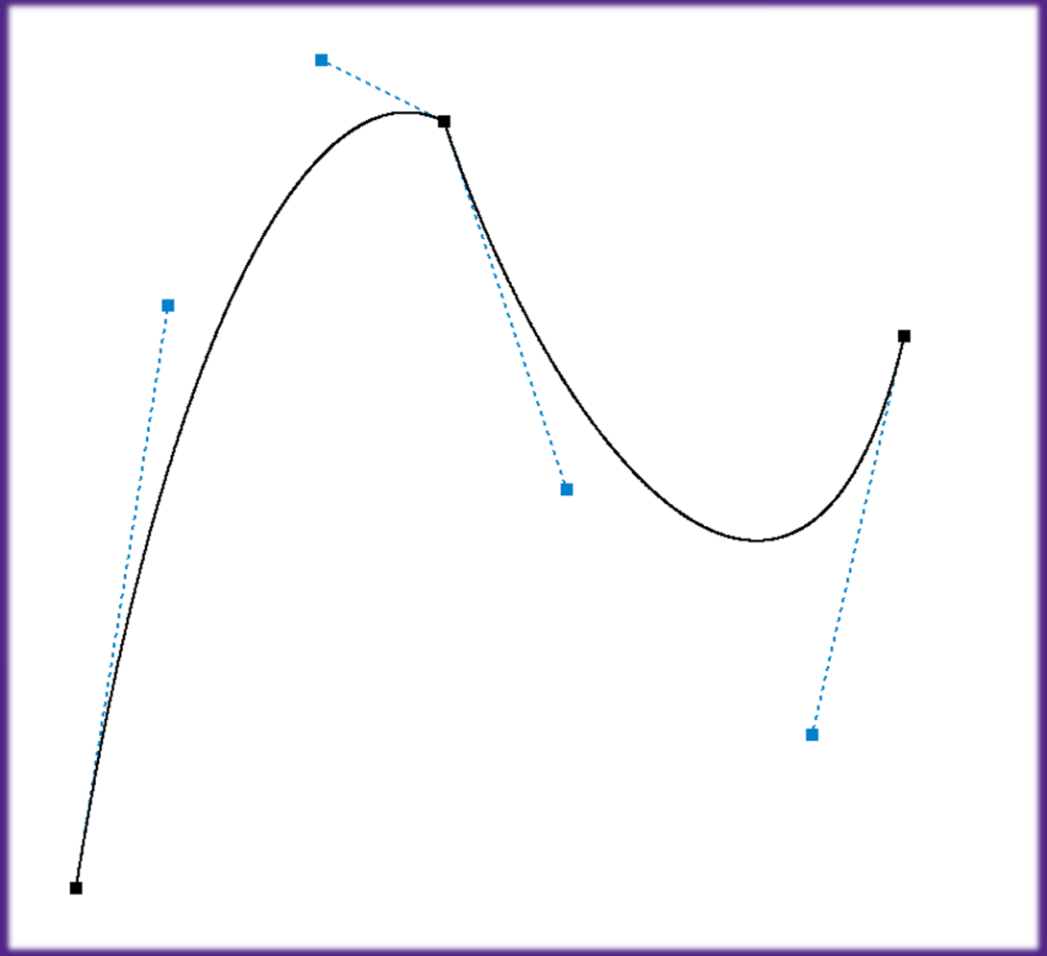
Now we want curves and non-trivial control points. If we specify those control points ahead of time, it's easy:

```
float Ax = -15, Ay = -10;
float Bx = -3, By = 15;
float Cx = 12, Cy = 8;

float c1x = -12, c1y = 9;
float c2x = -7, c2y = 17;
float c3x = 1, c3y = 3;
float c4x = 9, c4y = -5;

glPointSize(10.0f);
glBegin(GL_POINTS);
    glVertex2f(Ax, Ay);
    glVertex2f(Bx, By);
    glVertex2f(Cx, Cy);
glEnd();
glPointSize(2.0f);
glBegin(GL_POINTS);
    for (int i = 0; i < N; ++i) {
        float t = (float) i;
        t /= N;
        glVertex2f(bezier4(Ax,c1x,c2x,Bx,t), bezier4(Ay,c1y,c2y,By,t));
        glVertex2f(bezier4(Bx,c3x,c4x,Cx,t), bezier4(By,c3y,c4y,Cy,t));
    }
glEnd();
```

Splines







Splines

- While this spline is continuous, it is no particularly smooth.
- Look at what happens right before the middle point B and right after.
- There's a sharp crease in the curve.
- Typically, we like to avoid that in splines.
- Mathematically, this is caused by the derivative of the spline S(t) not being continuous; there is a sharp jump in the slope of S(t).

Splines

- We could get into the mathematics of it, or, we could just give the intuition.
- For an interior point on a Bezier spline: if the control point just before it, the interior point itself, and the control point just after it all fall on a straight line, then the spline will have a continuous first derivative and the curve will appear more “smooth”.

Let's calculate the line connecting control point 2 and B. Recall that the formula for the line between two points is:

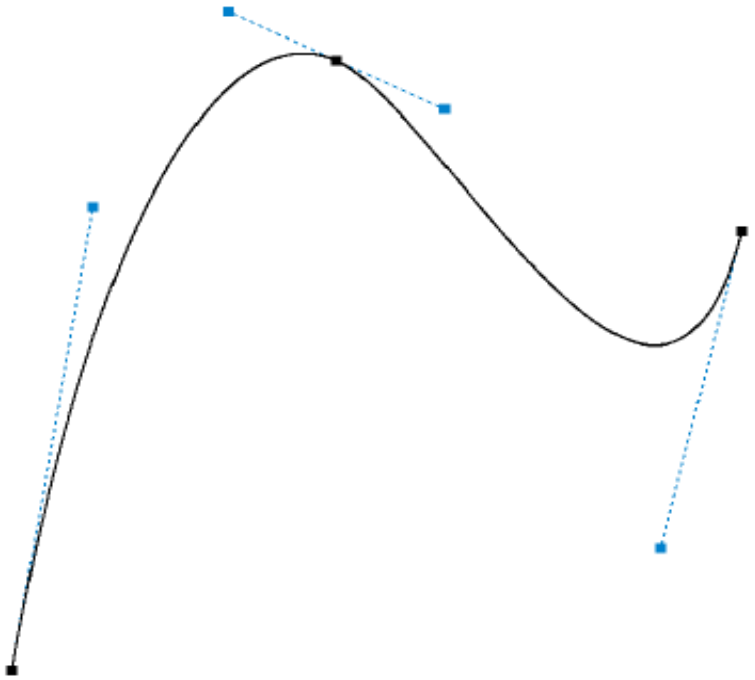
$$y - y_1 = \frac{y_1 - y_0}{x_1 - x_0}(x - x_1)$$

$$c_2 = (-7, 17) \quad B = (-3, 15)$$

$$y = \frac{15 - 17}{-3 + 7}(x + 7) + 17 = \frac{-x}{2} + \frac{27}{2}$$

So, we should move c_3 to also be on this line. Let's keep its x-coordinate the same and compute its correct y coordinate as:

$$c_{3,y} = \frac{-1}{2} + \frac{27}{2} = 13$$



Piecewise Polynomial Curves and Splines

- By defining piecewise polynomial curves as blending functions, we can control their extent over the domain of the parameter t and thus have only a subset of blending functions determining the influence of neighboring control points on the curve in a local way.

- Understanding check – What advantage does this provide over the polynomial curve?

Piecewise Polynomial Curves and Splines

- Consider the following piecewise polynomial:

$$g(t) = \begin{pmatrix} a(t) = \frac{1}{2}t^2, t \in [0,1] \\ b(t) = \frac{3}{4} - \left(t - \frac{3}{2}\right)^2, t \in [1,2] \\ c(t) = \frac{1}{2}(3-t)^2, t \in [2,3] \end{pmatrix}$$

- The support for this piecewise polynomial is [0,3]
  The joints are the location where the polynomials meet
  The knots are the t values at which the polynomials meet

Piecewise Polynomial Curves and Splines

$$g(t) = \begin{pmatrix} a(t) = \frac{1}{2}t^2, t \in [0,1] \\ b(t) = \frac{3}{4} - \left(t - \frac{3}{2}\right)^2, t \in [1,2] \\ c(t) = \frac{1}{2}(3-t)^2, t \in [2,3] \end{pmatrix}$$

- The support for this piecewise polynomial is [0,3]
  The joints are the location where the polynomials meet
  The knots are the t values at which the polynomials meet

- The polynomial g(t) would work well for local control of a curve

Definition of a B-Spline Function

The definition of a B-Spline is:

$$P(t) = \sum_{k=0}^{L} P_k N_{k,m}(t)$$

where $N_{k,m}(t)$ is the k^{th} B-Spline of order m. There are L+1 control points and a knot vector $\vec{T}(t_0,t_1,\ldots)$ specifies the values of t at the joints.

Definition of a B-Spline Function

What is the shape of a quadratic B-Spline $N_{0,3}(t)$, given a knot vector knot vector $\vec{T}(t_0,t_1,\ldots)$?

$$= \frac{t}{2}N_{0,2}(t)+\frac{3-t}{2}N_{1,2}(t)$$

$$= \frac{t}{2}\left[\left(\frac{t-t_0}{t_1-t_0}\right)N_{0,1}(t)+\left(\frac{t_2-t}{t_2-t_1}\right)N_{1,1}(t)\right]+\left(\frac{3-t}{2}\right)\left[\left(\frac{t-t_1}{t_2-t_1}\right)N_{1,1}(t)+\left(\frac{t_3-t}{t_3-t_2}\right)N_{2,1}(t)\right]$$

Definition of a B-Spline Function

This turns out to be:

$$N_{0,3} = \begin{pmatrix} \frac{1}{2}t^2, t \in [0,1] \\ \frac{3}{4} - \left(t - \frac{3}{2}\right)^2, t \in [1,2] \\ \frac{1}{2}(3-t)^2, t \in [2,3] \end{pmatrix}$$

In general, the cubic spline is most used.

Definition of a B-Spline Function

We can consider instead $N_{0,4}(t)$, which is symmetrical at t=2 and can be compactly written as:

$$N_{0,4} = \begin{pmatrix} u(1-t), 0 \leq t \leq 1 \\ v(2-t), 1 \leq t \leq 2 \\ v(t-2), 2 \leq t \leq 3 \\ u(t-3), 3 \leq t \leq 4 \end{pmatrix}$$

where $u(t) = 1/6(1-t)^3$ and $v(t) = 1/6(3t^3-6t^2+4)$

Definition of a B-Spline Function

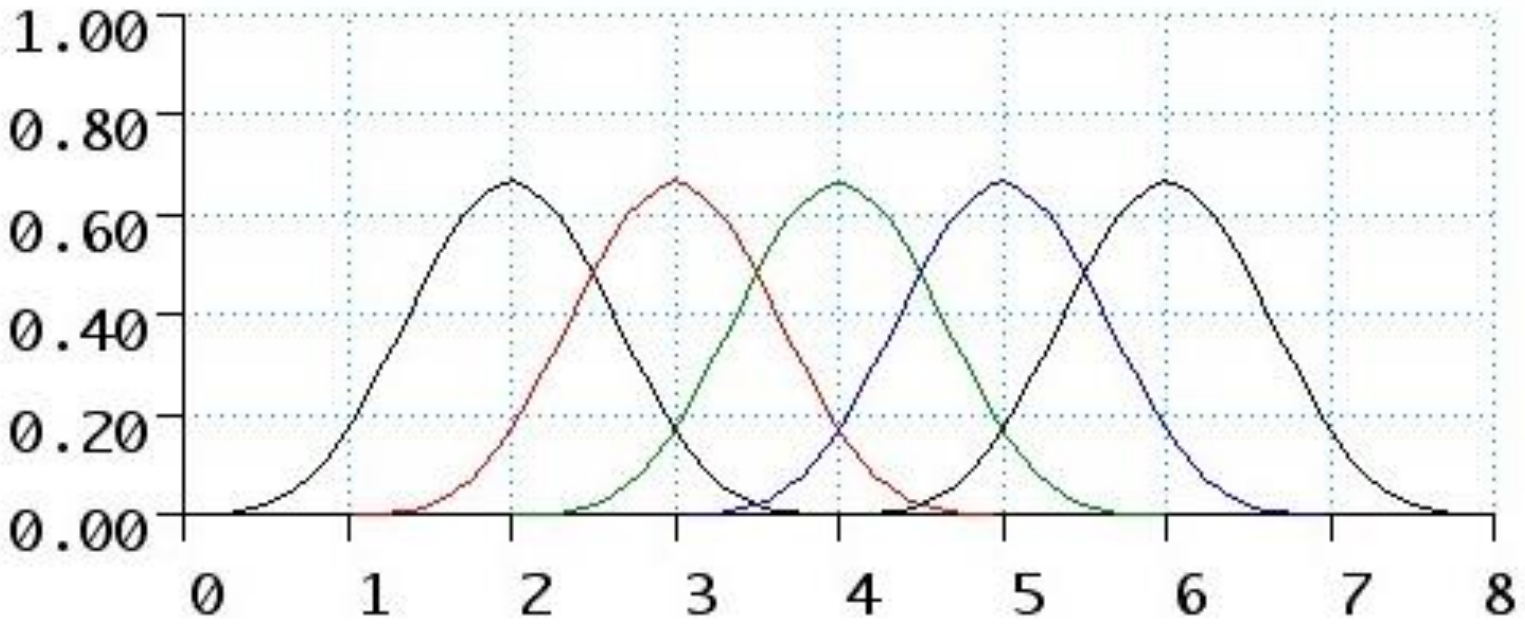
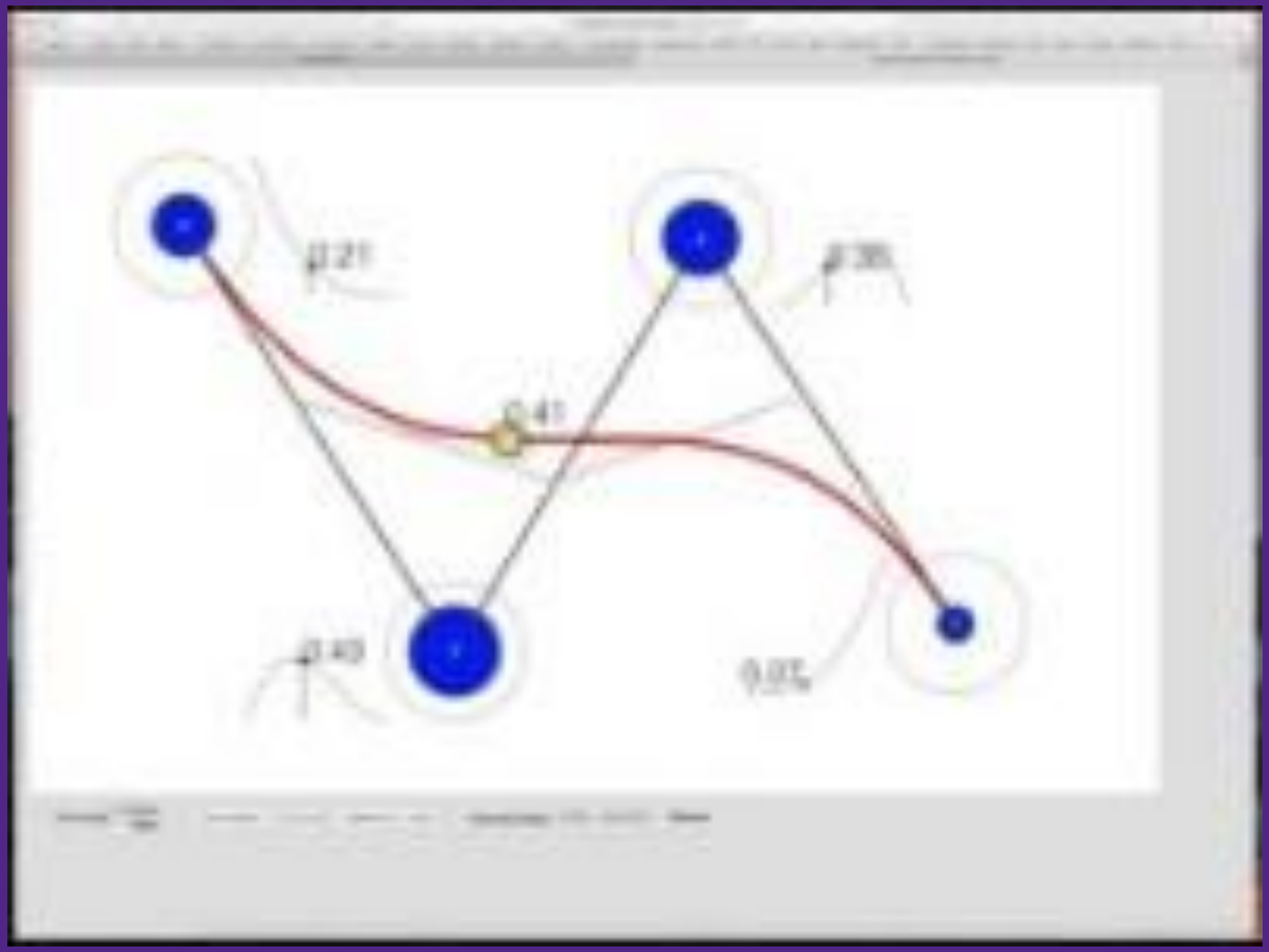


Illustration 2: The shape of the polynomials for a uniform B-Spline

Interactive Curve / Spline Demo



Interactive Curve / Spline Demo

- https://richardfuhr.neocities.org/BusyBCurves.html

Open B-Splines

- Bezier splines are special cases of B-Splines. B-Splines acting as Bezier splines are called open B-Splines.
- With open B-Splines, the knot vector (known as the standard knot vector in this case) is arranged so that the B-Spline will interpolate the first and the last point.
- In order to achieve this, we create a knot vector containing $L+m+1$ knots. The first m knots $t_0,\ldots,t_{m-1}$ are set to zero while the last m knots $t_{L+1},\ldots,t_{L+m}$ are set to $L-m+2$

Open B-Splines

- The knots $t_m,\ldots,t_L$ start at value 1, and increase by one up to $L-m+1$.
- Hence, the order m cannot exceed the number of control points.
- Also, it is easy to verify that when $m=L+1$, then $N_{k,L+1}(t)=B_k^L(t)$

- For example, if we have $L+1=8$ control points and we are using cubic B-Splines ($m=4$), then the standard knot vector is $T=(0,0,0,0,1,2,3,4,5,5,5,5)$

- This B-Spline will interpolate the first and last control points.

Open B-Splines

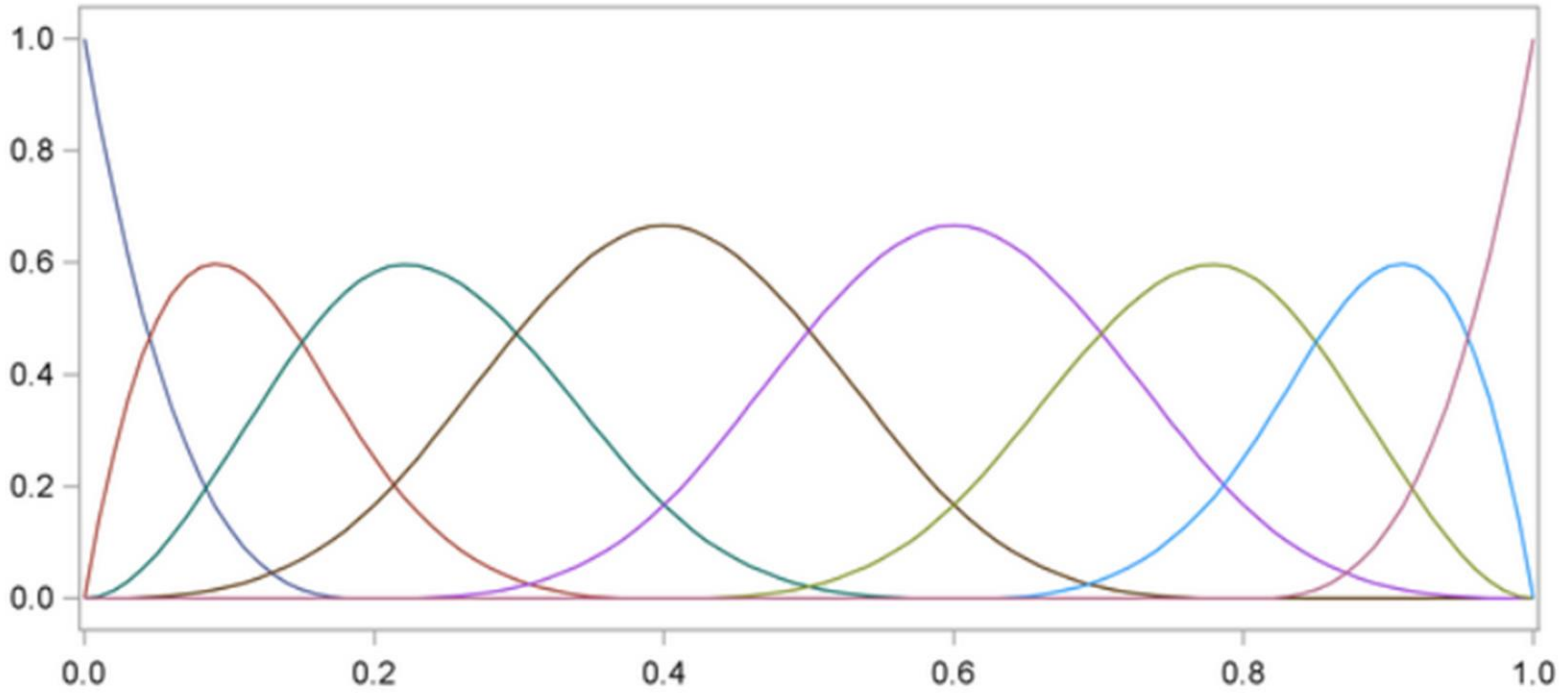


Illustration 3: The shape of the polynomials for an open B-Spline with standard knot vector.

Properties of B-Splines

- They are piecewise polynomials of order m which are $m-2$ smooth (that is to say, they have $m-2$ continuous successive derivatives.
- $N_{k,m}(t)$ Has support over $[t_k, t_{k+m}]$.
  - Quick understanding check, what does 'support' mean here?
- The last and first control points are interpolated when the standard knot vector is used.
- A B-Spline is always contained within the convex hull formed by the control points.

Properties of B-Splines

- Putting several control points at the same coordinates attracts the curve more strongly to that point.
- If we have a Spline of order m , then m−1 consecutive multiple points at the same coordinates will make the spline interpolate that point.
- While knot multiplicity and control point multiplicity are not equivalent, some of the effects are similar.

Properties of B-Splines

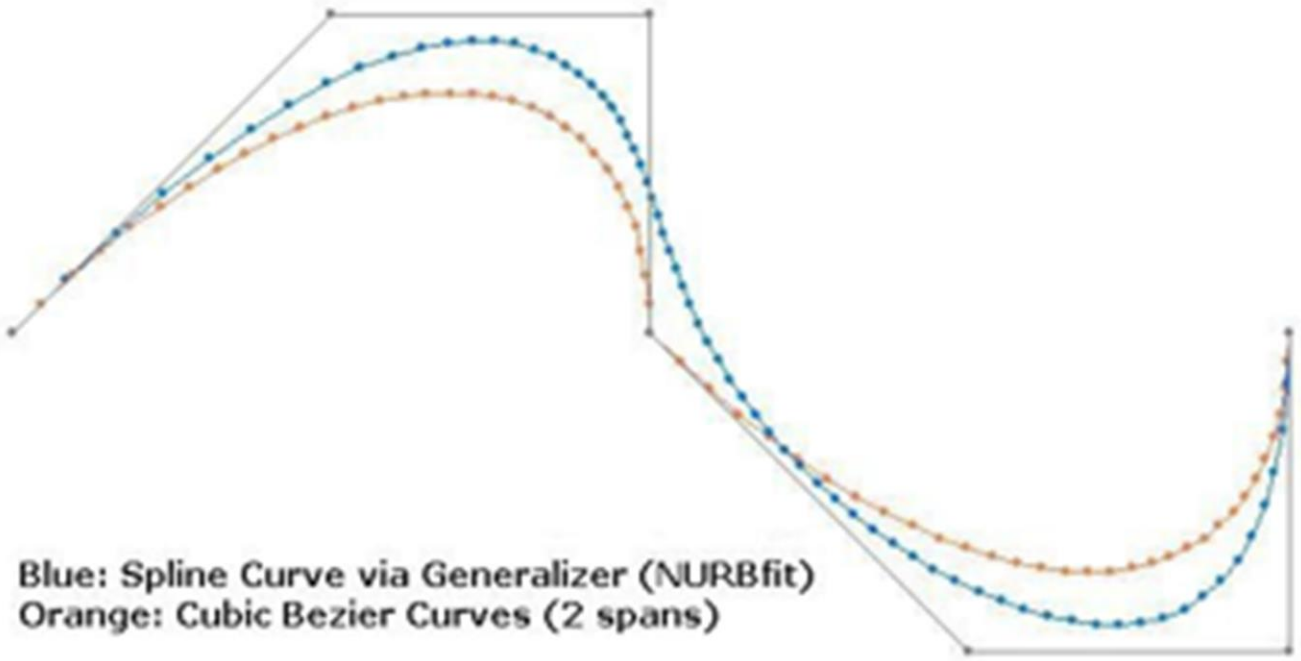


Illustration 4: For the same set of control points the Bezier and B-Spline curves can be quite different.

Rational Splines

- We may also use weights to even better control the shape of the curve.
- For each control point we can associate a weight, known as a shape parameter.
- If there are L+1 control points, then we would have the weights: $\Omega=(\omega_0, \omega_1,\ldots,\omega_L)$ for control points $P_0, P_1,\ldots, P_L$, and the rational spline would be written as:

$$P(t) = \sum_{k=0}^{L} R_k(t)$$

Rational Splines

- the rational spline would be written as:

$$P(t) = \sum_{k=0}^{L} R_k(t)$$

where

$$R_k(t) = \frac{\omega_k P_k N_{k,m}(t)}{\sum_{j=0}^{L} \omega_j N_{j,m}(t)}$$

B-Spline Surface Patch

- The generalization for curves to 2d surface patches is an extension from the previous. A B-Spline patch is given by:

$$P(u,v) = \sum_{i=0}^{M} \sum_{k=0}^{L} P_{i,k} N_{i,m}(u)\, N_{k,n}(v)$$

and, in the case of a rational B-Spline surface patch (NURB):

$$P(u,v) = \frac{\sum_{i=0}^{M} \sum_{k=0}^{L} \omega_{i,k} P_{i,k} N_{i,m}(u)\, N_{k,n}(v)}{\sum_{i=0}^{M} \sum_{k=0}^{L} \omega_{i,k} N_{i,m}(u)\, N_{k,n}(v)}$$

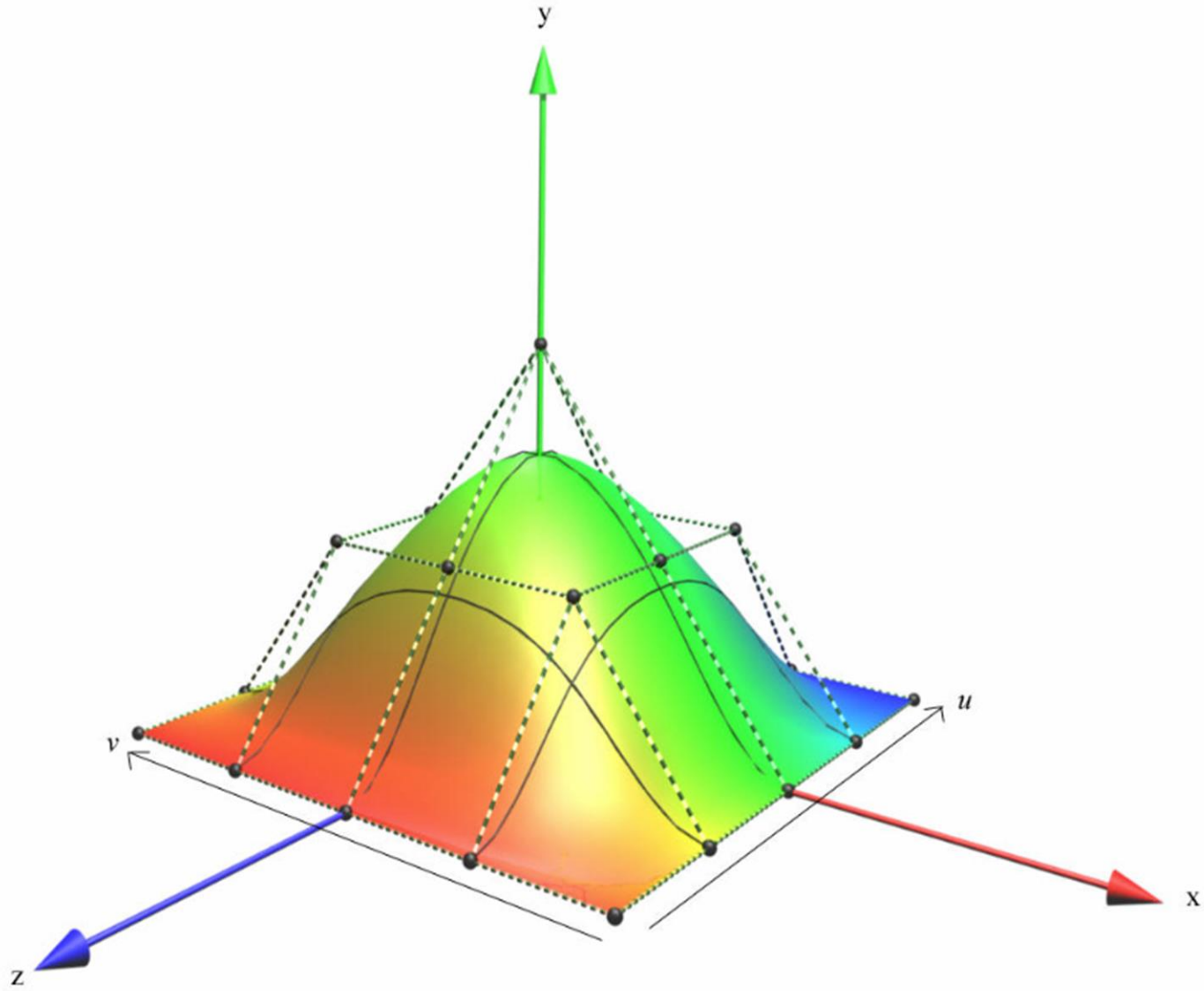


Illustration 5: A NURBS surface defined with control points

(Optional) Video Content

https://www.youtube.com/watch?v=cxX-AA3xslo

(Optional) Video Content

https://youtu.be/PBgDEUn_ss8

(Optional) Video Content

https://youtu.be/aVwxzDHniEw