Computer Graphics

Curves

Konstantin Tretyakov
kt@ut.ee

Nov 27, 2013

Quiz

- What methods for modeling geometric primitives have we discussed in this course so far?

Quiz

- What methods for modeling geometric primitives have we discussed in this course so far?
  - Mesh (i.e. polygon / polyhedron)
  - Voxel set / point cloud
  - Parametric
    - Line $\boldsymbol{x} = t\boldsymbol{s}$, Segment $x = t\boldsymbol{a} + (1 - t)\boldsymbol{b}$, Triangle
  - Implicit
    - Line, Plane $\boldsymbol{n}^{\boldsymbol{T}}\boldsymbol{x} = 0$, Sphere $||\boldsymbol{x}|| = r$
    - Distance field

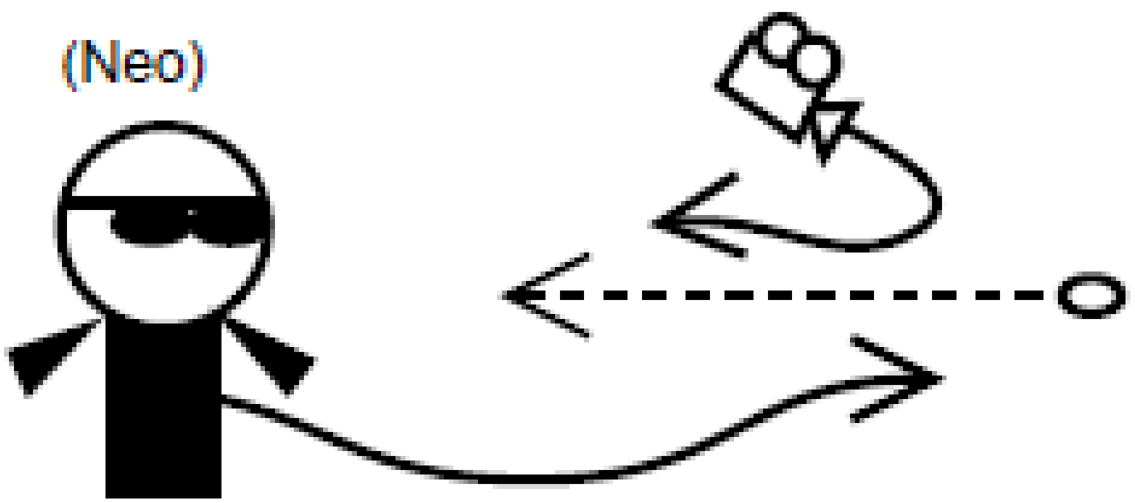
Curves and Surfaces

Enable us to **model smooth primitives** **without the need to resort to:**

- **Crude approximations** (such as a low-poly mesh)
- **Specify lots of parameters** (such as a high-poly mesh)
- **Ugly hacks** (such as “normal interpolation” or “smooth shading”)

Today: Curves

- Curves (in 3D graphics) are primarily used to model **movement trajectories**.



Curves: Main questions

- How to **represent** a curve
  - Explicit, implicit and parametric forms
  - Polynomial, piecewise polynomial, basis-functions
- How to **specify** a curve
  - Curve interpolates given points
  - Curve approximates given points
  - Curve is defined via direction vectors in its start/end points.
  - Curve is defined via laws of physics.

Explicit representation

- Coordinates are represented as a function of some other coordinate:
  - In 2D: $y = f(x)$
  - In 3D: $(y, z) = \big(f(x), g(x)\big)$
- Good: easy to find y or z given x
- Bad: many curves cannot be represented, e.g. “circle”, “vertical line”, etc. Transformations are hard to apply.
- => Practically never used in CG.

Implicit representation

- In two dimensions, an implicit representation for a curve is:

$$f(x, y) = 0$$

- E.g.:
  - Circle: $\|(x, y)\| - r = 0$
  - Parabola: $x^2 - y = 0$

Implicit representation

- In three dimensions, an implicit representation for a curve is:

Implicit representation

- In three dimensions, an implicit representation for a **curve** is a **system of two equations**:

$$\begin{cases} f(x,y,z) = 0 \\ g(x,y,z) = 0 \end{cases}$$

(a single equation $f(x,y,z) = 0$ in 3D represents a **surface**, as it has **two** degrees of freedom)

Implicit representation

- Good:
  - Easy to model common mathematical shapes (circles, straight lines, spheres, etc)
  - Convenient form to compute ray intersections in raycasting algorithms.
- Bad:
  - Hard to model custom shapes.
  - Hard to enumerate the points of a curve/surface directly, i.e. inconvenient for projection-based rendering like the standard pipeline.

Parametric representation

- Curve:

$$\boldsymbol{p}(t) = \boldsymbol{f}(t)$$

- Surface:

$$\boldsymbol{p}(u, v) = \boldsymbol{f}(u, v)$$

- Same general form for 2D, 3D, etc.
- E.g.:
  - Circle: $\boldsymbol{p}(t) = \big(x(t), y(t)\big) = (\cos(t), \sin(t))$
  - Parabola: $\boldsymbol{p}(t) = (t, t^2)$
  - Spiral: $\boldsymbol{p}(t) = (t, \cos(t), \sin(t))$

Parametric representation

- Enumerating points on a curve is straightforward, just evaluate the function for different parameter values.

- Cutting a curve into pieces or combining from several pieces is easy.

Parametric representation

- The direction vector (velocity) at each point on a parametric curve can be obtained by differentiating:

$$\boldsymbol{s}(t) = \frac{\partial \boldsymbol{p}(t)}{\partial t} = \left(\frac{\partial x(t)}{\partial t}, \frac{\partial y(t)}{\partial t}, \frac{\partial z(t)}{\partial t}\right)^T$$

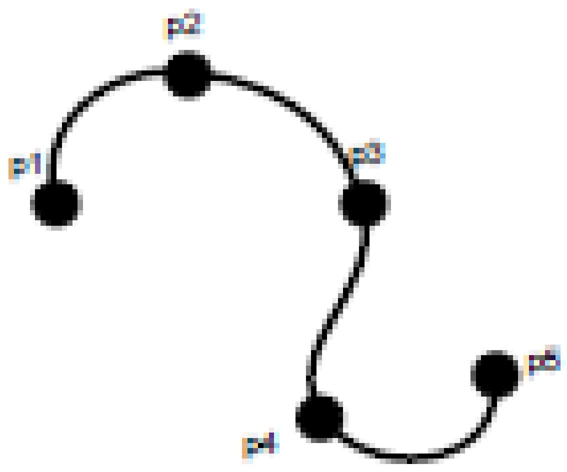
Specifying a curve

- Coming up with a new equation each time we need to make a curve is cumbersome.

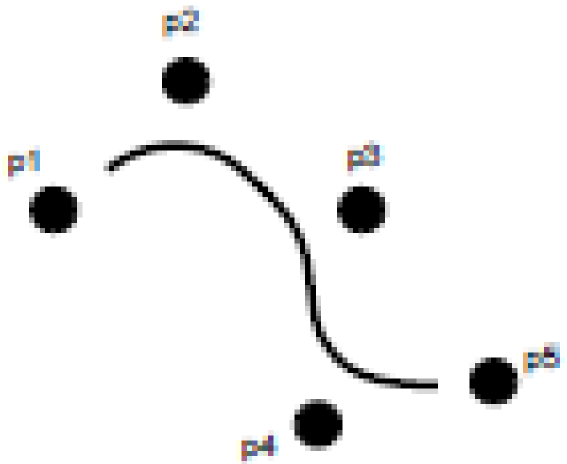
- Instead we want a generic “curve-making” algorithm that will use a few number of parameters and create a curve.

Specifying a curve

- A natural idea is to provide a set of *control points* and require the curve to *interpolate* or *approximate* those.



Interpolation



Approximation

Polynomial curves

The most widespread type of curves in graphics are *polynomial* curves, i.e. curves of the form

$$\mathbf{p}(t) = \mathbf{c}_0 + \mathbf{c}_1 t + \mathbf{c}_2 t^2 + \cdots + \mathbf{c}_n t^n =$$

$$= \begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix} = \begin{pmatrix} c_{00} + c_{01} t + c_{02} t^2 + \cdots + c_{0n} t^n \\ c_{10} + c_{11} t + c_{12} t^2 + \cdots + c_{1n} t^n \\ c_{20} + c_{21} t + c_{22} t^2 + \cdots + c_{2n} t^n \end{pmatrix}$$

Polynomial curves

$$\mathbf{p}(t) = \begin{pmatrix} c_{00} + c_{01}t + c_{02}t^2 + \cdots + c_{0n}t^n \\ c_{10} + c_{11}t + c_{12}t^2 + \cdots + c_{1n}t^n \\ c_{20} + c_{21}t + c_{22}t^2 + \cdots + c_{2n}t^n \end{pmatrix} =$$

$$= \begin{pmatrix} c_{00} & \cdots & c_{0n} \\ c_{10} & \cdots & c_{1n} \\ c_{20} & \cdots & c_{2n} \end{pmatrix} \begin{pmatrix} 1 \\ t \\ t^2 \\ \vdots \\ t^n \end{pmatrix} =: \mathbf{C}\mathbf{T}_n(t)$$

Polynomial curves

So, the general form of a d-dimensional n-th degree polynomial curve is

$$\boldsymbol{p}(t) = \boldsymbol{C}\boldsymbol{T}_n(t)$$

where

$$\boldsymbol{T}_n(t) = (1, t, t^2, \ldots, t^n)^T$$

and

$\boldsymbol{C}$ is a $d \times (n+1)$ *coefficient matrix*

To specify the curve we need to provide $\boldsymbol{C}$.

Polynomial curves

- Providing $\boldsymbol{C}$ directly is non-intuitive.
- Instead we want to provide control points and compute from them the matrix $\boldsymbol{C}$, requiring that the curve
  - Interpolates the control points, or
  - Approximates the control points

Polynomial curves

- Quiz: How many control points do we need to provide to fully specify a n-th degree curve?

Polynomial curves

- Quiz: How many control points do we need to provide to fully specify a n-th degree curve?

  - We eventually need to determine $\boldsymbol{C}$, which is a $d \times (n+1)$ matrix.

Polynomial curves

- Quiz: How many control points do we need to provide to fully specify a n-th degree curve?
  - We eventually need to determine $\boldsymbol{C}$, which is a $d \times (n+1)$ matrix.
  - We need to provide $d \times (n+1)$ parameters.

Polynomial curves

- Quiz: How many control points do we need to provide to fully specify a n-th degree curve?

  - We eventually need to determine $\boldsymbol{C}$, which is a $d \times (n+1)$ matrix.
  - We need to provide $d \times (n+1)$ parameters.
  - Each point is d-dimensional, so fixing $(n+1)$ control points suffices.

Polynomial curves

- Suppose we have fixed $n + 1$ control points $\boldsymbol{p}_0, \boldsymbol{p}_1, \dots, \boldsymbol{p}_n$
- We shall next consider two methods of converting those control points to a parameter matrix $\boldsymbol{C}$.
  - Lagrange' interpolation (interpolation)
  - Bezier curve (approximation)

- NB: In both cases the result is a polynomial curve, the difference is only how the control points are used.

Lagrange interpolation

- Suppose in addition to $\boldsymbol{p}_0, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_n$ we are provided with $n + 1$ parameter values ("timepoints") $t_0, t_1, \ldots, t_n$.

- Let us seek for a polynomial curve that satisfies

$$\boldsymbol{p}(t_i) = \boldsymbol{p}_i$$

Lagrange interpolation

- Let us seek for a polynomial curve that satisfies

$$\boldsymbol{p}(t_i) = \boldsymbol{p}_i$$

Lagrange interpolation

- Let us seek for a polynomial curve that satisfies

$$\boldsymbol{p}(t_i) = \boldsymbol{p}_i$$

$$\boldsymbol{CT}(t_i) = \boldsymbol{p}_i$$

Grouping together into a matrix equation:

$$\boldsymbol{C}\big(\boldsymbol{T}(t_0)\ \ \boldsymbol{T}(t_1)\ \ \ldots\ \ \ \boldsymbol{T}(t_n)\big) = (\boldsymbol{p}_0\ \boldsymbol{p}_1 \ldots \boldsymbol{p}_n)$$

$$\boldsymbol{CA} = \boldsymbol{P}$$

Lagrange interpolation

Thus, the coefficient matrix of a Lagrange curve is

$$\boldsymbol{C} = \boldsymbol{P}\boldsymbol{A}^{-1}$$

where

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & \ldots & 1 \\ t_0 & t_1 & \ldots & t_n \\ \vdots & \vdots & \ddots & \vdots \\ t_0^n & t_1^n & \ldots & t_n^n \end{pmatrix}$$

$\boldsymbol{A}$ is invertible whenever $i \neq j \Rightarrow t_i \neq t_j$

Lagrange interpolation

- Curves of degree 3 are most popular in computer graphics. Let us construct a third degree Lagrange curve.

Lagrange interpolation

- Suppose the parameter values corresponding to the control points are
  $t_0 = 0, \qquad t_1 = 1/3, \qquad t_2 = 2/3, \qquad t_3 = 1$
- Then

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1/3 & 2/3 & 1 \\ 0 & (1/3)^2 & (2/3)^2 & 1 \\ 0 & (1/3)^3 & (2/3)^3 & 1 \end{pmatrix}$$

Lagrange interpolation

- It follows that

$$\mathbf{M}_L = \mathbf{A}^{-1} = \begin{pmatrix} 1 & -5.5 & 9 & -4.5 \\ 0 & 9 & -22.5 & 13.5 \\ 0 & -4.5 & 18 & -13.5 \\ 0 & 1 & -4.5 & 4.5 \end{pmatrix}$$

and

$$\mathbf{p}(t) = \mathbf{P}\mathbf{M}_L\mathbf{T}(t)$$

Geometry & Basis matrices

$$\mathbf{p}(t) = \mathbf{P}\mathbf{M}_L\mathbf{T}(t)$$

- The matrix $\boldsymbol{P} = (\boldsymbol{p}_0, \boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3)$ will be referred to as the **geometry matrix** of the curve.
- The matrix $\boldsymbol{M}_L$ is the **basis matrix** of the Lagrange interpolation curve.
- We shall see that other curves can also be described in terms of a geometry and basis matrix as $\boldsymbol{p}(t) = \boldsymbol{GMT}(t)$

Blending functions

$$\mathbf{p}(t) = \mathbf{P}\mathbf{M}_L\mathbf{T}(t)$$

- Consider the product $\boldsymbol{M}_L\boldsymbol{T}$(t):

$$\mathbf{M}_L\mathbf{T}(t) = \begin{pmatrix} 1 - 5.5t + 9t^2 - 4.5t^3 \\ 9t - 22.5t^2 + 13.5t^3 \\ -4.5t + 18t^2 - 13.5t^3 \\ t - 4.5t^2 + 4.5t^3 \end{pmatrix} =: \begin{pmatrix} b_0(t) \\ b_1(t) \\ b_2(t) \\ b_3(t) \end{pmatrix}$$

- The functions b_i are the **blending functions** of the curve.

Blending functions

- The third degree curve can be represented via blending functions as

$$\boldsymbol{p}(t) = b_0(t)\boldsymbol{p}_0 + b_1(t)\boldsymbol{p}_1 + b_2(t)\boldsymbol{p}_2 + b_3(t)\boldsymbol{p}_3$$

- Compare it to the line segment equation, which is, incidentally, a first degree polynomial interpolating (Lagrange) curve.

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

Blending functions

- We could actually find the Lagrange blending functions directly, looking for $b_i(t)$ s.t.

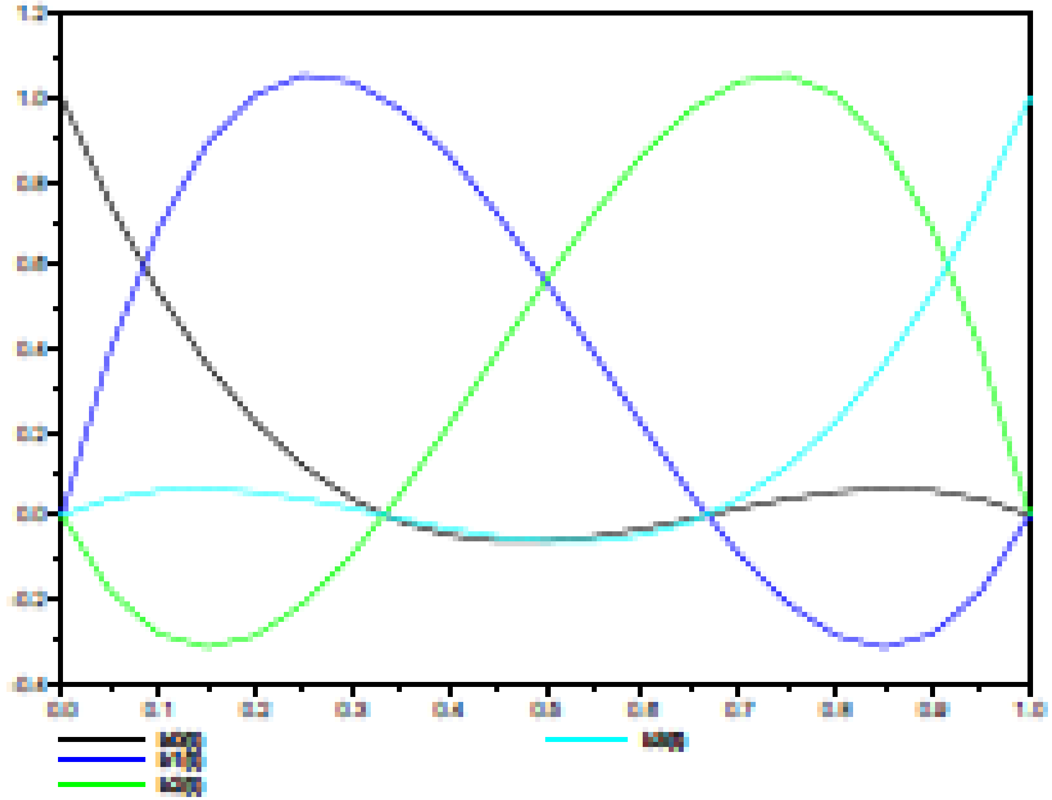
$$b_i(t_i) = 1, \qquad b_i(t_j) = 0, \qquad \text{if } i \neq j$$

- $b_i(t) = \dfrac{(t-t_0)(t-t_1)\ldots(t-t_{i-1})(t-t_{i+1})\ldots(t-t_n)}{(t_i-t_0)(t_i-t_1)\ldots(t_i-t_{i-1})(t_i-t_{i+1})\ldots(t_i-t_n)}$

- (See how it works out for degree 1, for example)

Blending functions

- Lagrange blending functions for degree 3 curve



Blending functions

- Lagrange blending functions satisfy

$$\sum_i b_i(t) = 1$$

- This ensures an important property of the curve:

$$\boldsymbol{A}\boldsymbol{p}(t) + \boldsymbol{d} = \boldsymbol{A}\left(\sum_i b_i(t)\boldsymbol{p}_{\boldsymbol{i}}\right) + \boldsymbol{d} =$$

$$= \sum_i b_i(t)(\boldsymbol{A}\boldsymbol{p}_{\boldsymbol{i}} + \boldsymbol{d})$$

Blending functions

To transform a curve by an affine transformation it suffices to transform the control points.

- This ensures an important property of the curve:

$$\boldsymbol{Ap}(t) + \boldsymbol{d} = \boldsymbol{A}\left(\sum_i b_i(t)\boldsymbol{p_i}\right) + \boldsymbol{d} =$$

$$= \sum_i b_i(t)(\boldsymbol{Ap_i} + \boldsymbol{d})$$

Intermediate summary

- We are talking about **polynomial parametric curves**.
- Such curves can be represented using a **geometry** and a **basis matrix**

$$\boldsymbol{p}(t) = \boldsymbol{GMT}(t)$$

- Alternatively, such curves can be represented using **blending functions**:

$$\boldsymbol{p}(t) = \sum_i b_i(t)\boldsymbol{p}_i$$

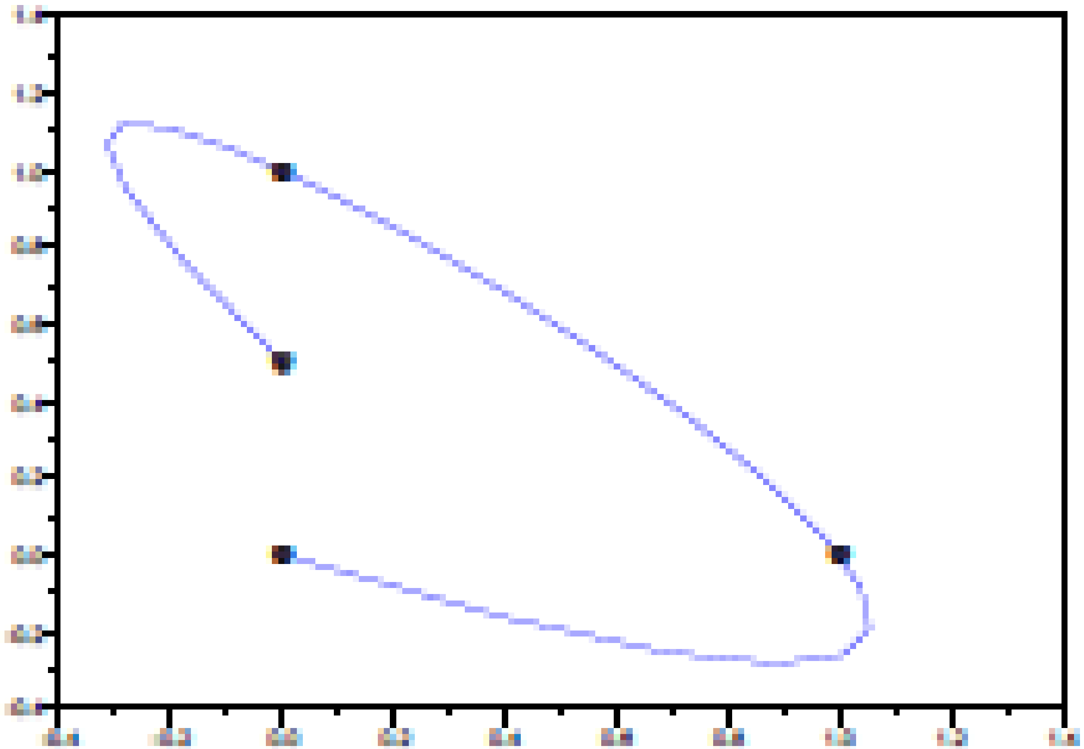
- Blending functions should sum to 1 for any t.

Intermediate summary

- We have derived the geometry and basis matrices for the 3^{rd} degree Lagrange curve.
- Lagrange curve is an interpolating curve.
- Turns out we have already known a 1^{st} degree Lagrange curve, which is just a line segment between two points.

Lagrange curve

Lagrange curves higher than degree 1 are rarely used as they tend to “wiggle around” too much.



Bezier' curve

- Much "calmer" are Bezier' curves.
- Bezier curve of degree n:

$$\mathbf{p}(t) = \sum_{i=0}^{n} B_i^{(n)}(t)\mathbf{p}_i$$

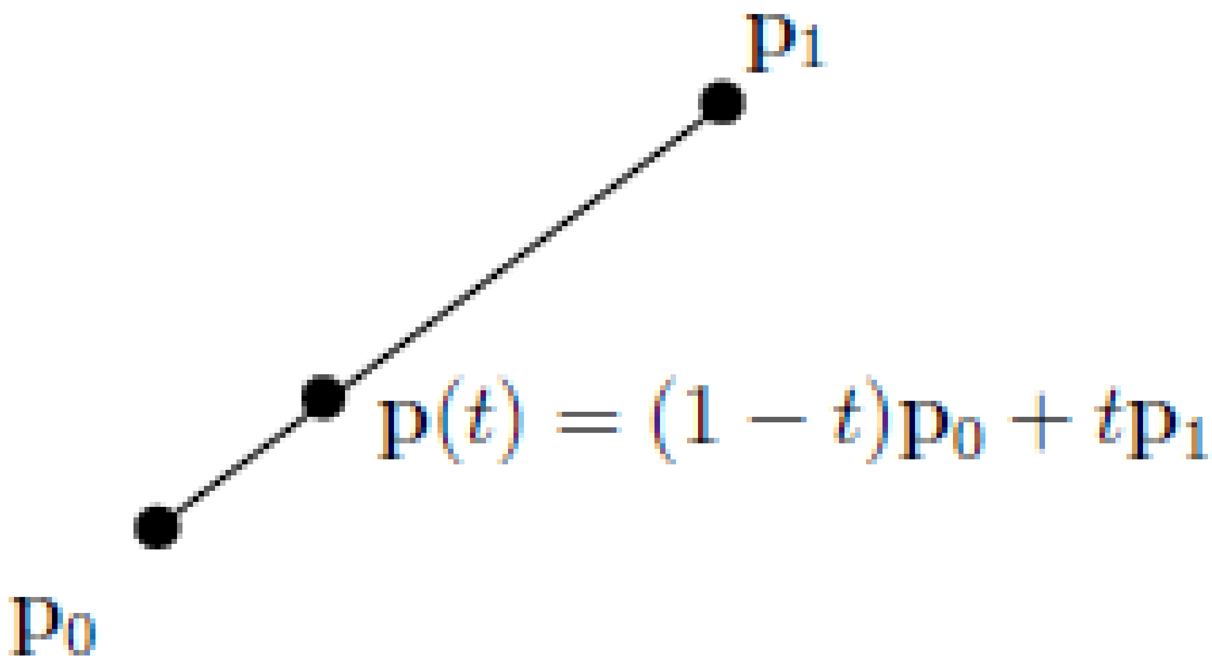
where

$$B_i^{(n)}(t) = \binom{n}{i}(1-t)^{n-i}t^i$$

are *Bernstein' polynomials*.

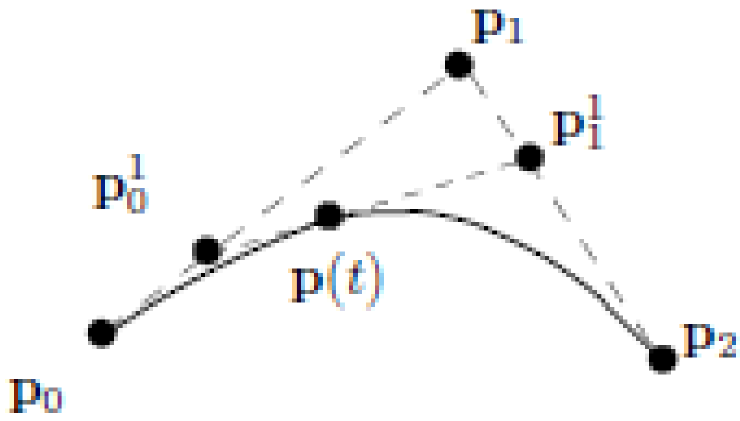
Bezier' curve

- Degree 1 Bezier' curve



Bezier' curve

- Degree 2 Bezier' curve

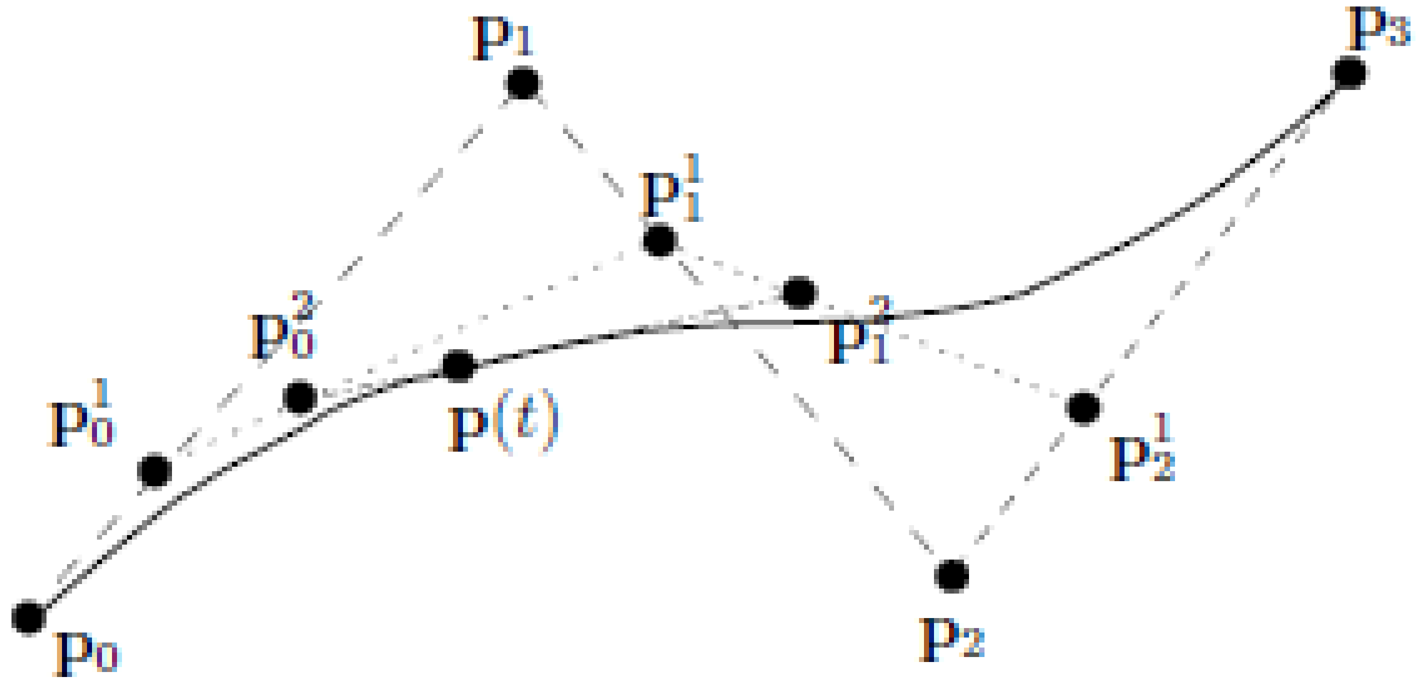


$$\mathbf{p}_0^1 = (1-t)\mathbf{p}_0 + t\mathbf{p}_1 \qquad \mathbf{p}_1^1 = (1-t)\mathbf{p}_1 + t\mathbf{p}_2$$

$$\mathbf{p}(t) = (1-t)\mathbf{p}_0^1 + t\mathbf{p}_1^1 = (1-t)^2\mathbf{p}_0 + 2(1-t)t\mathbf{p}_1 + t^2\mathbf{p}_2$$

Bezier' curve

- Degree 3 Bezier' curve



Bezier' curve

- Degree 3 Bezier' curve

$$
\begin{aligned}
\mathrm{p}_0^1 &= (1-t)\mathrm{p}_0 + t\mathrm{p}_1 &\qquad \mathrm{p}_0^2 &= (1-t)\mathrm{p}_0^1 + t\mathrm{p}_1^1 \\
\mathrm{p}_1^1 &= (1-t)\mathrm{p}_1 + t\mathrm{p}_2 &\qquad \mathrm{p}_1^2 &= (1-t)\mathrm{p}_1^1 + t\mathrm{p}_2^1 \\
\mathrm{p}_2^1 &= (1-t)\mathrm{p}_2 + t\mathrm{p}_3 & &
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{p}(t) &= (1-t)\mathrm{p}_0^2 + t\mathrm{p}_1^2 \\
&= (1-t)^3\mathrm{p}_0 + 3(1-t)^2 t\mathrm{p}_1 + 3(1-t)t^2\mathrm{p}_2 + t^3\mathrm{p}_3 \\
&= \sum_{i=0}^{3} B_i^{(3)}(t)\mathrm{p}_i
\end{aligned}
$$

Bernstein' polynomials

$$B_0^{(0)} = 1$$

$$B_0^{(1)} = (1-t) \qquad B_1^{(1)} = t$$

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

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

$$\boxed{B_i^{(n)}(t) = \binom{n}{i}(1-t)^{n-i}t^i}$$

Bezier’ curve blending functions

- The blending functions of a Bezier’ curve (i.e. Bernstein polynomials) satisfy the sum-to-one requirement.

- In addition, for all $t \in [0,1]$
$$0 \leq B_i^n(t) \leq 1$$
which guarantees that the Bezier curve is always **within the convex hull** of its control points.

Bezier' curve

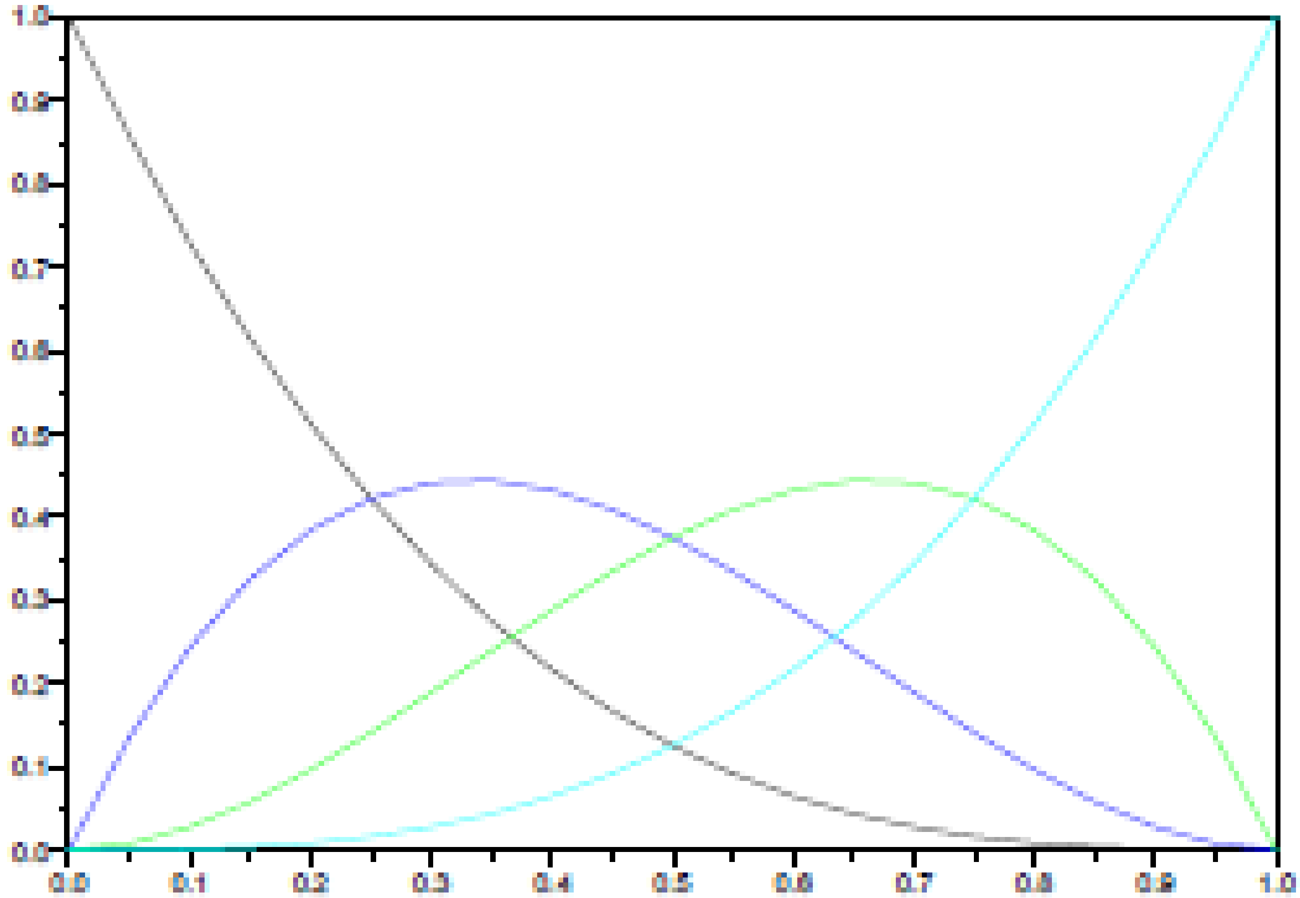
- Again, the most widespread are Bezier' curves of degree 3:

$$p(t) = (1-t)^3 p_0 + 3(1-t)^2 t p_1 + 3(1-t) t^2 p_2 + t^3 p_3$$

- In matrix form:

$$\boldsymbol{p}(t) = \boldsymbol{P}\boldsymbol{M}_B\boldsymbol{T}(t) \qquad \boldsymbol{M}_B = \begin{pmatrix} 1 & -3 & 3 & -1 \\ 0 & 3 & -6 & 3 \\ 0 & 0 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Bernstein’ blending functions



Bezier' curves

- Consider the direction (derivative) of the Bezier' curve at the start' and end point

$$\boldsymbol{s}(t) = \frac{\partial \boldsymbol{p}(t)}{\partial t} = \boldsymbol{P}\boldsymbol{M}_B \frac{\partial \boldsymbol{T}(t)}{\partial t}$$

where

$$\frac{\partial \boldsymbol{T}(t)}{\partial t} = \begin{pmatrix} 0 \\ 1 \\ 2t \\ 3t^2 \end{pmatrix}$$

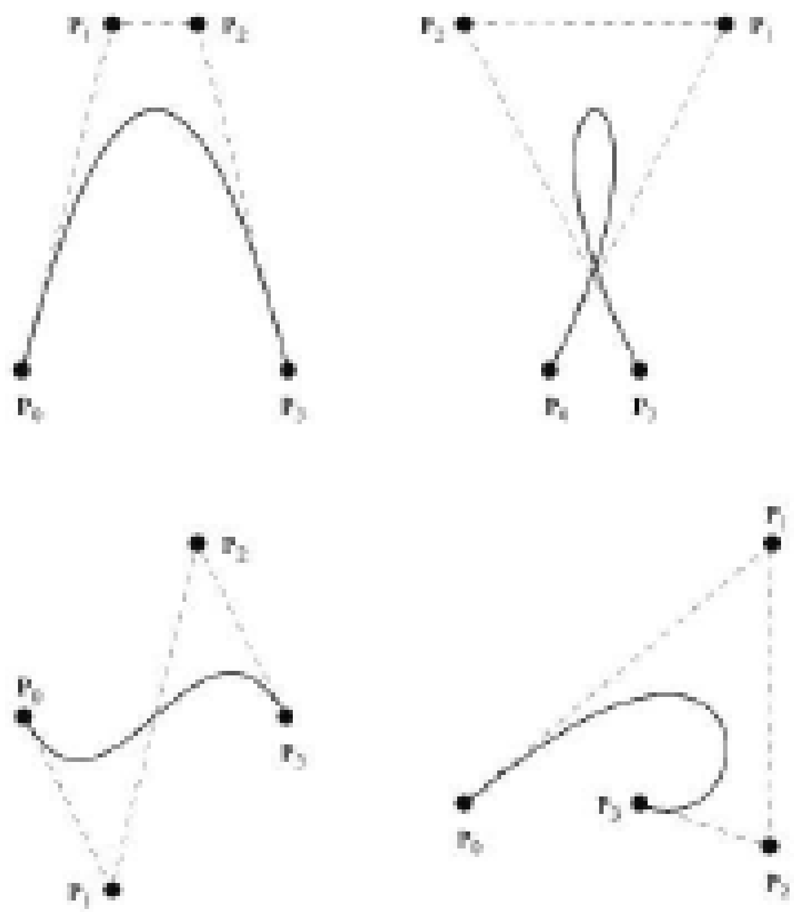
Bezier' curves

- From this we get

$$\boldsymbol{s}(0) = 3(\boldsymbol{p}_1 - \boldsymbol{p}_0), \qquad \boldsymbol{s}(1) = 3(\boldsymbol{p}_3 - \boldsymbol{p}_2)$$

- This provides certain "intuitiveness" to Bezier curves.
- It also lets us combine several Bezier' pieces into a longer smooth curve.

Bezier’ curves



Hermite' curves

- A popular reparameterization of the Bezier' curve is to specify the curve via its start and end points $\boldsymbol{p}_0, \boldsymbol{p}_3$ and its direction vectors at those points $\boldsymbol{s}_0, \boldsymbol{s}_3$.
- The geometry matrix is then

$$\boldsymbol{G} = (\boldsymbol{p}_0, \boldsymbol{p}_3, \boldsymbol{s}_0, \boldsymbol{s}_3)$$

- The basis matrix (verify!):

$$\boldsymbol{M}_H = \begin{pmatrix} 1 & 0 & -3 & 2 \\ 0 & 0 & 3 & -2 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & -1 & 1 \end{pmatrix}$$

Intermediate summary

- **Lagrange curve**: interpolation, “jumpy”
- **Bezier curve**: approximation, always within convex hull of control points. Blending functions are called *Bernstein polynomials*.
- The familiar $(1 - t)\boldsymbol{p_0} + t\boldsymbol{p_1}$ is both a Lagrange and a Bezier curve with respect to its control points.

Piecewise polynomial curves

- Polynomial curves are somewhat limited when it comes to describing long stretches.

- A much better approach is to make a long curve by concatenating ("glueing") several short ones together.

- Such **piecewise polynomial curves** are called **splines**.

Piecewise polynomial curves

- If you glue together Lagrange curves, you’ll get a “Lagrange spline”
- If you glue together Bezier’ curves, you’ll get a “Bezier spline”
- If you glue together Hermite’ curves, you’ll get a “Hermite spline”
- Many other spline types can be constructed this way (read up on those).
- We shall consider “Natural splines” here.

The art of glueing curves

- Let $\boldsymbol{p}(t), t \in [0,1]$ and $\boldsymbol{q}(t), t \in [0,1]$ be two curves.
- When combining them into one, we might require:
  - Continuity: $\boldsymbol{p}(1) = \boldsymbol{q}(0)$
  - Visual smoothness: $\frac{\partial \boldsymbol{p}(1)}{\partial t} = \alpha \frac{\partial \boldsymbol{q}(0)}{\partial t}, \quad \alpha > 0$
  - Parametric (true) smoothness: $\frac{\partial \boldsymbol{p}(1)}{\partial t} = \frac{\partial \boldsymbol{q}(0)}{\partial t}$

The art of glueing curves

- We say that the curve is C^k-smooth *(order k parametrically smooth)*, if

$$\forall m \in 1 \ldots k \qquad \frac{\partial^m \boldsymbol{p}(1)}{\partial t^m} = \frac{\partial^m \boldsymbol{q}(0)}{\partial t^m}$$

- We say the curve is G^k smooth *(k-th order geometrically smooth)*, if it is C^{k-1}-smooth and

$$\frac{\partial^k \boldsymbol{p}(1)}{\partial t^k} = \frac{\partial^k \boldsymbol{q}(0)}{\partial t^k}$$

The art of glueing curves

- G^1-smoothness suffices for a visually smooth-looking curve.

- If the curve is used as a trajectory, you typically want at least C^2-smoothness (so that acceleration is not changing in jumps).

(Natural) Splines

- Suppose now we have points $\boldsymbol{p}_0, \dots, \boldsymbol{p}_n$. The goal is to find a *maximally-smooth* piecewise polynomial curve, with pieces connecting at the control points.

- The whole curve will thus consist of pieces $\{\boldsymbol{q}_0(t), \boldsymbol{q}_1(t), \dots, \boldsymbol{q}_{n-1}(t)\}$, where each $\boldsymbol{q}_i(t), t \in [t_i, t_{i+1}]$ determines the piece between control points $\boldsymbol{p}_i$ and $\boldsymbol{p}_{i+1}$.

Linear spline

- A linear spline is a piecewise-linear curve

$$\mathbf{q}_i(t) = \frac{t - t_i}{t_{i+1} - t_i}\mathbf{p}_i + \frac{t_{i+1} - t}{t_{i+1} - t_i}\mathbf{p}_{i+1}$$

it only achieves C^0-smoothness

Quadratic spline

- Quadratic spline: piecewise-quadratic curve.
- Now that we have more parameters, we require C^1-smoothness:

$$\mathbf{q}_i(t_{i+1}) = \mathbf{q}_{i+1}(t_i)$$

$$\mathbf{q}_i'(t_{i+1}) = \mathbf{q}_{i+1}'(t_i)$$

- This requirement plus interpolation requirement, plus two arbitrarily fixed values for the curve derivative at start and end points (typically 0) uniquely determine the whole curve.

Cubic spline

- Piecewise-cubic curve, allows to achieve C^2-smoothness.
- Similarly to the quadratic spline, requires to specify derivatives at start and end points (typically 0).
- Similarly to the quadratic spline, requires to solve a system of equations to fit.

Cubic spline

- Cubic spline is a popular choice to represent smooth movement.
- They can be inconvenient, though, because adding or removing points affects the whole curve.
- Alternative: B-splines. Next week.

Summary

- Polynomial curve:

$$\mathbf{p}(t) = \mathbf{c}_0 + \mathbf{c}_1 t + \cdots + \mathbf{c}_n t^n := \mathbf{C}\mathbf{T}_n(t)$$

- Representation via geometry and basis matrices

$$\mathbf{p}(t) = \mathbf{G}\mathbf{M}\mathbf{T}(t)$$

- Representation via blending functions

$$\mathbf{p}(t) = \sum_{i=0}^{n} b_i(t)\mathbf{p}_i, \qquad \sum_{i=0}^{n} b_i(t) = 1$$

Summary

- Interpolating curves:
  - Lagrange' curve
  - Polynomial (natural) spline
- Approximating curves:
  - Bezier' curve
  - B-spline
- Specific basis matrices for some cubic curves: $\boldsymbol{M}_L, \boldsymbol{M}_B, \boldsymbol{M}_H$.