CG – T13 – Curves

L:CC, MI:ERSI

Miguel Tavares Coimbra

(course and slides designed by Verónica Costa Orvalho)

Suggested reading

- Shirley et al., “Fundamentals of Computer Graphics”, 3rd Edition, CRC Press
  - Chapter 15 - Curves

agenda

1. introduction
2. curves
3. surfaces

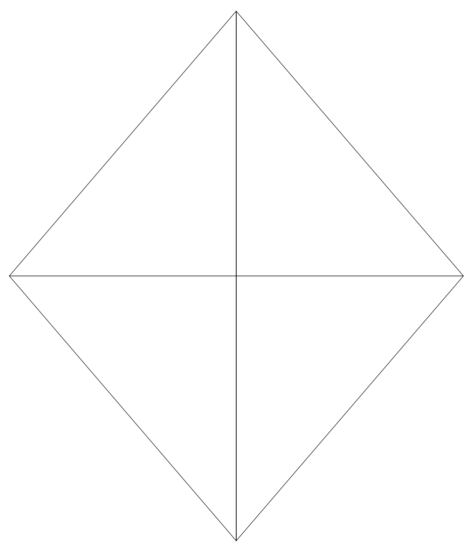
what we know so far?

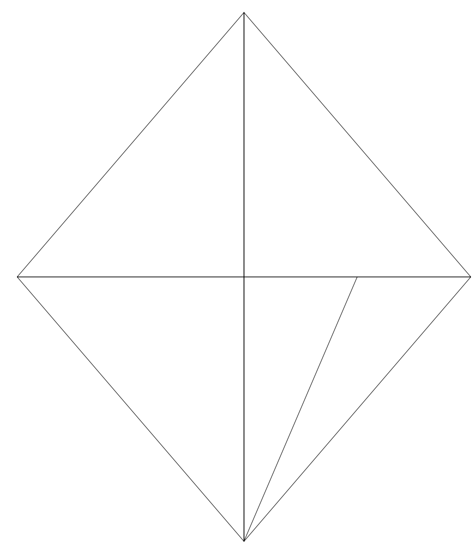
surface modeling using
polygon mesh

what we know so far?

surface modeling using

<u>**polygon mesh**</u>

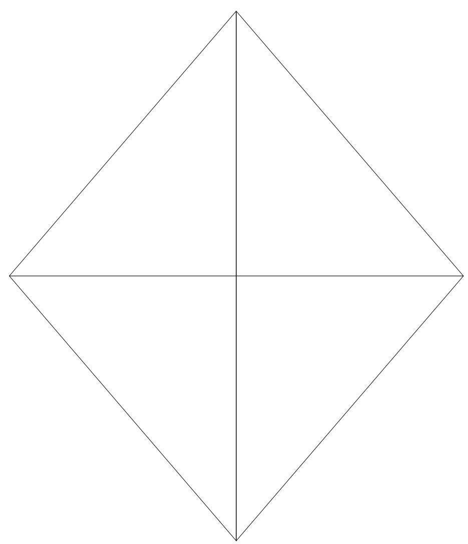




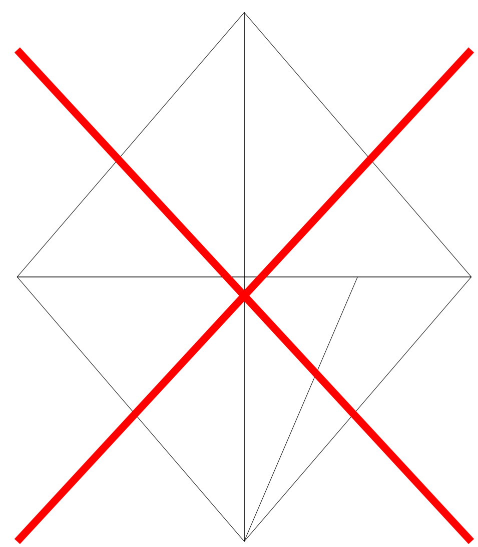
what we know so far?

surface modeling using
<u>polygon mesh</u>

↓



windged-edge
representation



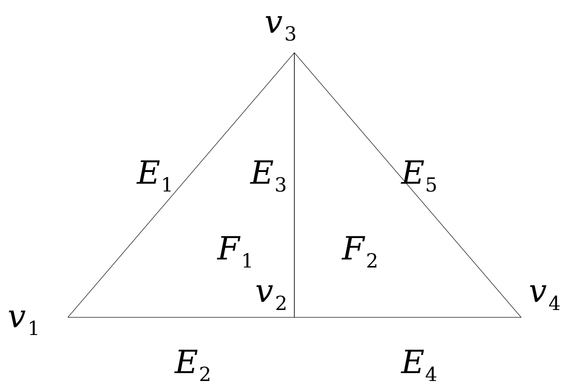
hard to iterate

what we know so far?

surface modeling using
<u>polygon mesh</u>

↓

collection of edges, vertices and faces, where:
. each edge shares at most 2 faces
. a vertex shares at least 2 edges



what we know so far?

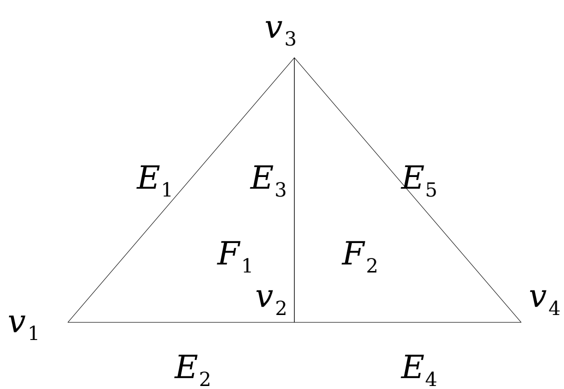
surface modeling using
<u>polygon mesh</u>

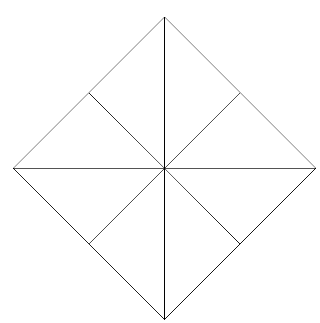
↓

collection of edges, vertices and faces, where:

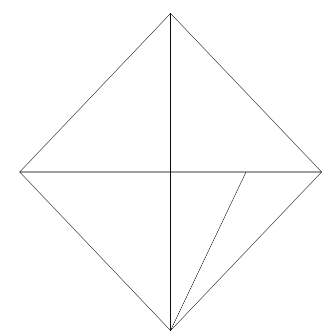
. each edge shares at most 2 faces

. a vertex shares at least 2 edges





windged-edge representation

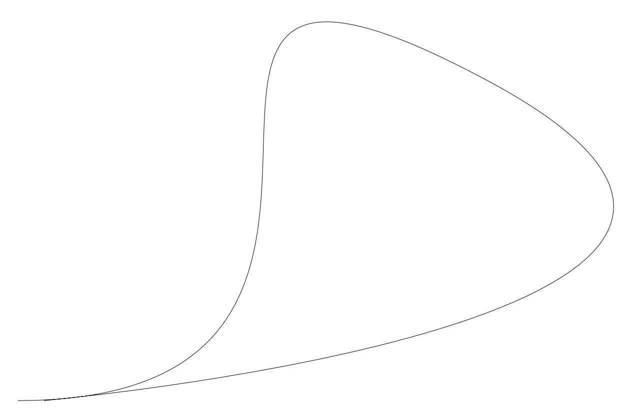


hard to iterate

what we know so far?

surface modeling using
polygon mesh

how can we represent a curve surface?



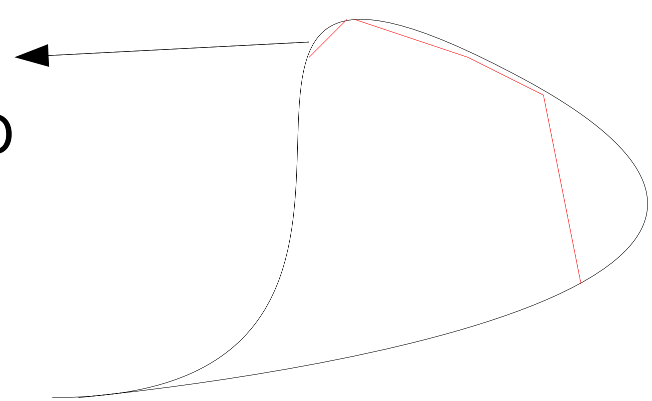
2D representation of a curve surface

what we know so far?

surface modeling using
polygon mesh

how can we represent a curve surface?

lines
lines strip
triangles
...



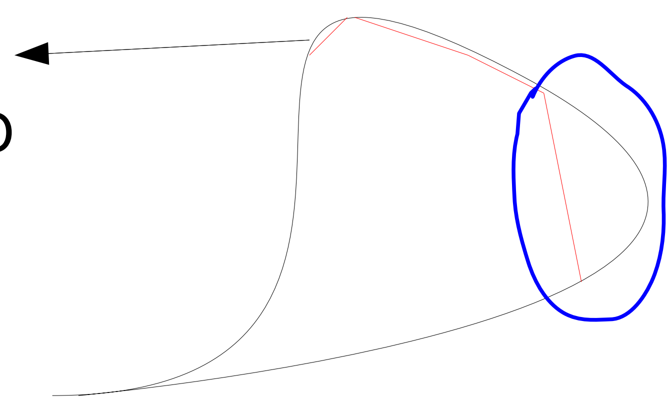
2D representation of a curve surface

what we know so far?

surface modeling using
polygon mesh

how can we represent a curve surface?

lines
lines strip
triangles
...



polygon meshs are
hard to represent
curved surfaces

2D representation of a curve surface

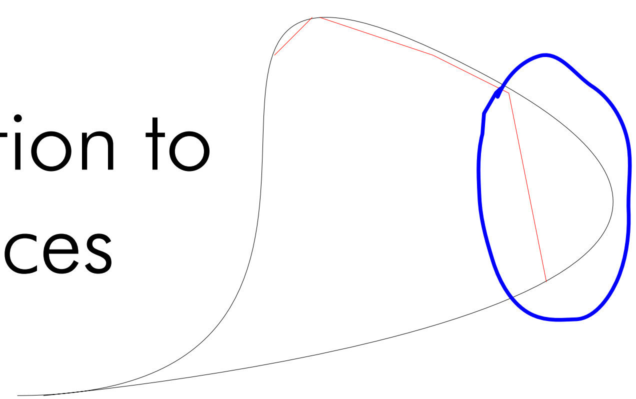
what we know so far?

surface modeling using
polygon mesh

↓

how can we represent a curve surface?

linear approximation to curves or surfaces



polygon meshs are hard to represent curved surfaces

2D representation of a curve surface

why using curves and curve surfaces?

1. more compact representation than polygons

why using curves and curve surfaces?

1. more compact representation than polygons

2. scalable geometric primitive

why using curves and curve surfaces?

1. more compact representation than polygons

2. scalable geometric primitive

3. smoother and more continuous primitives than lines and polygons

why using curves and curve surfaces?

1. more compact representation than polygons

2. scalable geometric primitive

3. smoother and more continuous primitives than lines and polygons

4. faster and simpler animation and collision detection

advantage

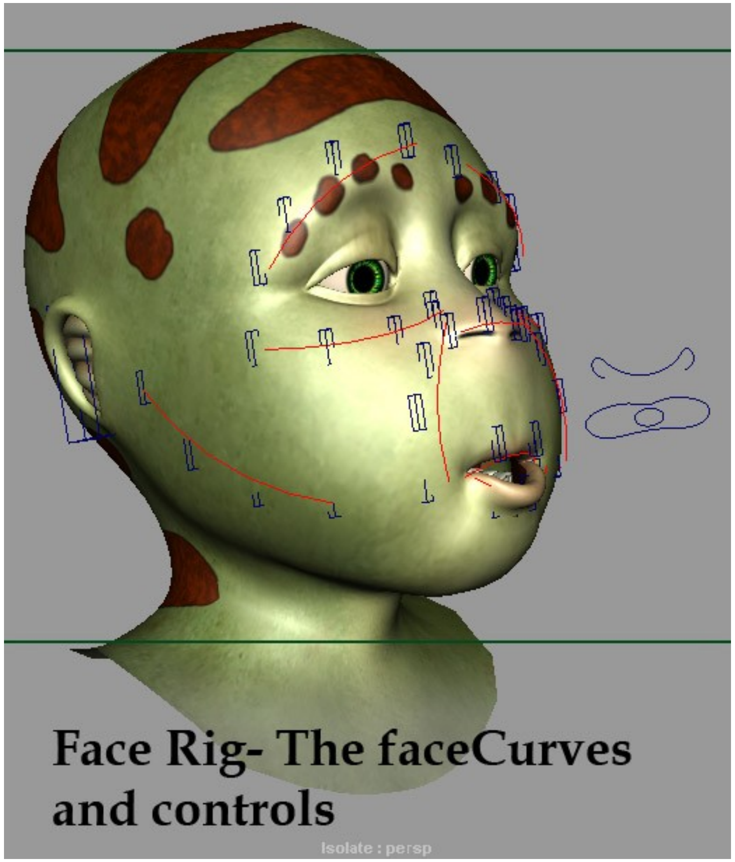
makes real-time CG applications:

> faster
> simpler to code
> last longer
(survive graphic HW generations)

where we use curves?

model **complex** object,
using **simple** pieces

DEMO + IMAGES
in Maya



what is a good curve representation?

. smooth and continuous
. allow local control of shape, so it is easy to create and edit
. stable, no oscillation
. easy to evaluate and render
. easy to compute derivatives

curve representation

1. Explicit

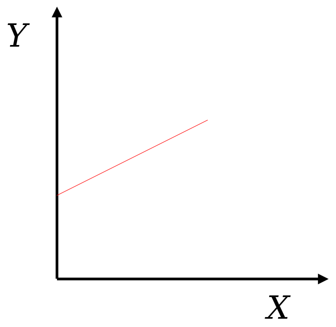
2. Implicit

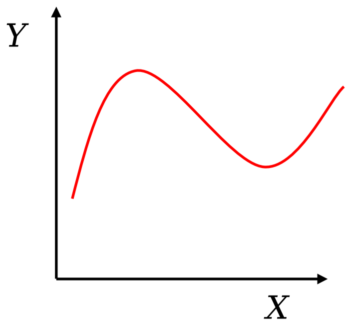
3. Parametric

curve representation

Explicit: y = f(x)

$$y = mx + b$$



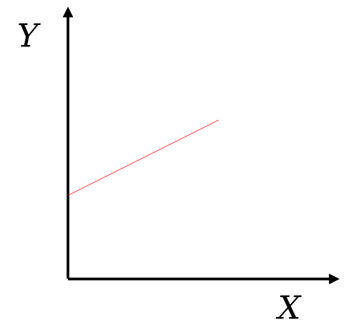


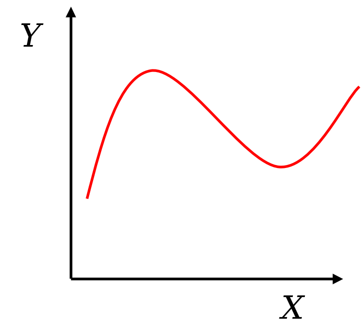
. easy to generate points

curve representation

Explicit: y = f(x)

$$y = mx + b$$





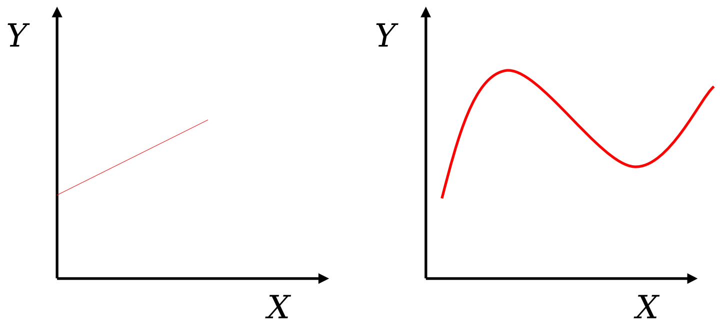
. easy to generate points

<u>big limitations</u>

1)

2)

curve representation



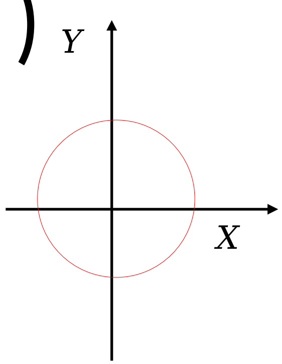
Explicit: y = f(x)

$$y = mx + b$$

. easy to generate points

<u>big limitations</u>

1)

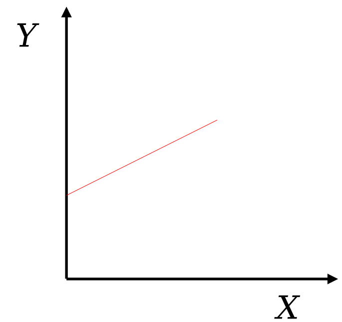


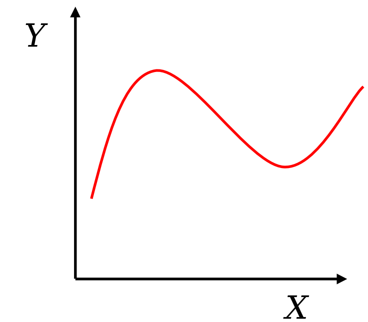
. must be represented by multiple **curve segments**

. is impossible to get multiple values of ***y*** for a unique ***x***

2)

curve representation





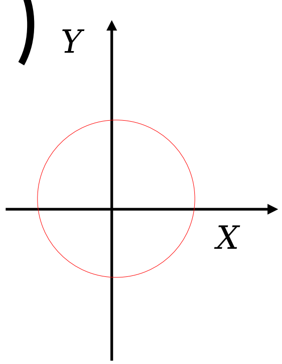
Explicit: y = f(x)

$$y = mx + b$$

. easy to generate points

<u>big limitations</u>

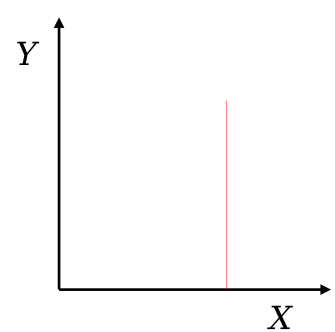
1)



. must be represented by multiple **curve segments**

. is impossible to get multiple values of ***y*** for a unique ***x***

2)



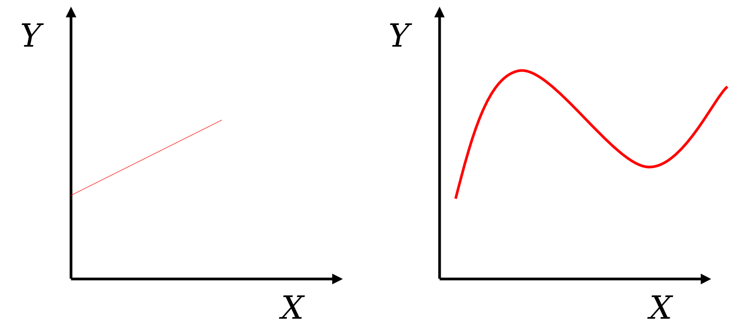
. **vertical lines** are very hard

. a slope of infinity is hard to represent, so vertical tangents are difficult to get

curve representation

Explicit: y = f(x)

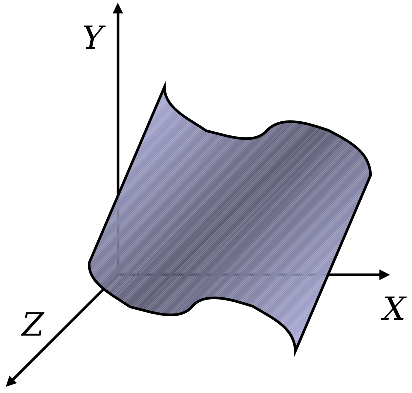
$y = mx + b$



. easy to generate points

in 3D:

$y = f(x)$ and $y = g(x)$



curve representation

Implicit: $f(x,y,z) = 0$

. easy to test if point on the curve

. normals are easy to compute

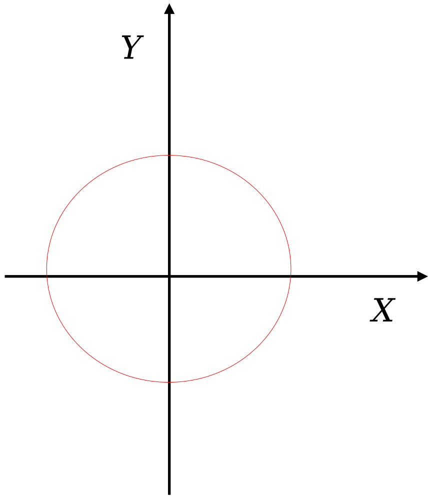
curve representation

Implicit: f(x,y,z) = 0

. easy to test if point on the curve

. normals are easy to compute

creating a circle $x^2+y^2-r^2=0$



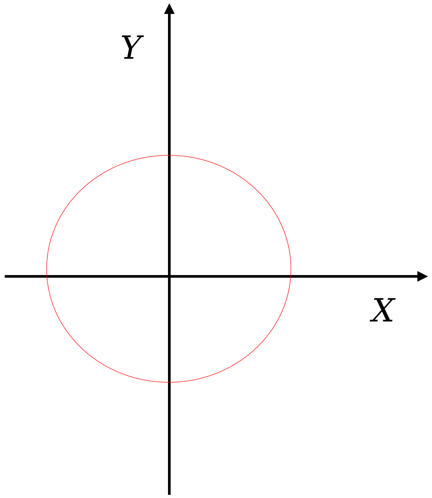
curve representation

Implicit: f(x,y,z) = 0

. easy to test if point on the curve

. normals are easy to compute

creating a circle $x^2+y^2-r^2=0$



<u>how do we model half circle?</u>

curve representation

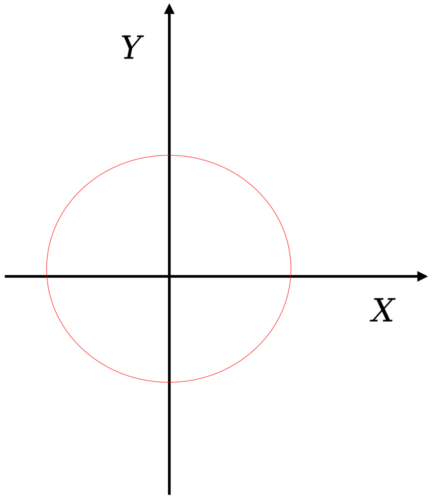
Implicit: f(x,y,z) = 0

. easy to test if point on the curve

. normals are easy to compute

creating a circle $x^2+y^2-r^2=0$

<u>how do we model half circle?</u>



add constraints $x \geqslant 0$

curve representation

Implicit: $f(x,y,z) = 0$

. easy to test if point on the curve

. normals are easy to compute

<u>limitations</u>

curve representation

Implicit: $f(x,y,z) = 0$

. easy to test if point on the curve
. normals are easy to compute

limitations

. constraints NOT included in implicit eq.

. difficult to determine tangent direction, then it is hard to join curve segments

. one equation might have more than 1 solution

. hard to generate points

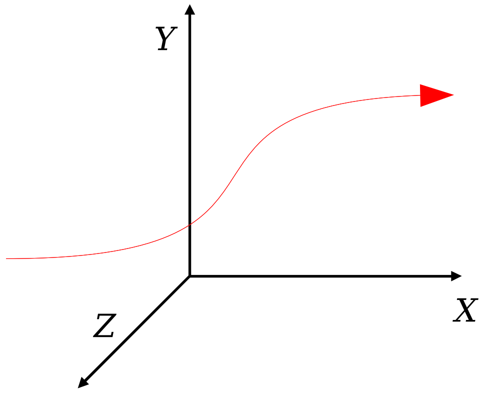
curve representation

Parametric: $(x,y,z) = (x(t), y(t), z(t))$

. separate equation for each spatial value

. easy to generate points

. replace the use of slopes (may be ∞) with parametric tangent vectors



curve representation

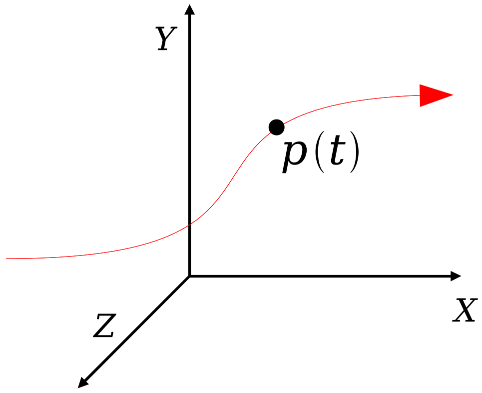
Parametric: $(x,y,z) = (x(t), y(t), z(t))$

. separate equation for each spatial value

. easy to generate points

. replace the use of slopes (may be ∞) with parametric tangent vectors

<u>how ?</u>



$$p(t)=[x(t),y(t),z(t)]^T$$

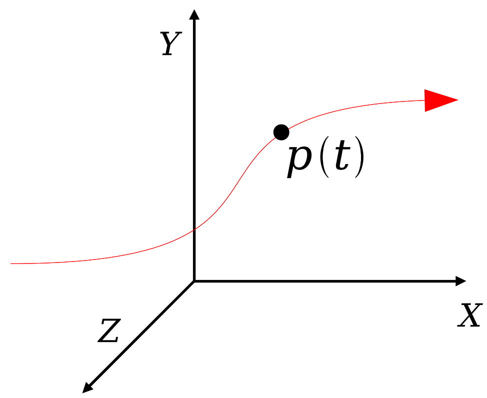
curve representation

Parametric: (x,y,z) = (x(t), y(t),z(t))

. separate equation for each spatial value
. easy to generate points
. replace the use of slopes (may be ∞) with parametric tangent vectors

<u>how ?</u>

A parametric curve describes points using some formula as a function of a parameter t



$$p(t)=[x(t),y(t),z(t)]^T$$

curve representation

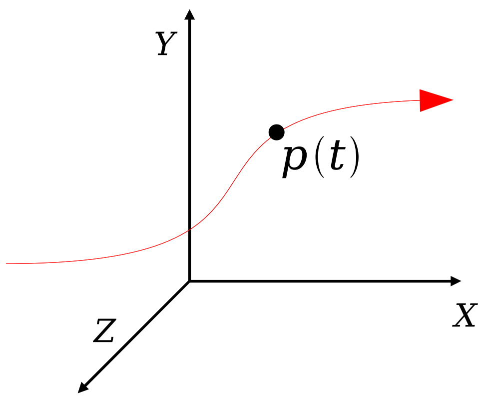
Parametric: (x,y,z) = (x(t), y(t),z(t))

. separate equation for each spatial value

. easy to generate points

. replace the use of slopes (may be ∞) with parametric tangent vectors

<u>how ?</u>



$$p(t) = [x(t), y(t), z(t)]^T$$

Each curve segment is given by 3 functions that are a polynomial:

$$x = x(t) \qquad y = y(t) \qquad z = z(t)$$

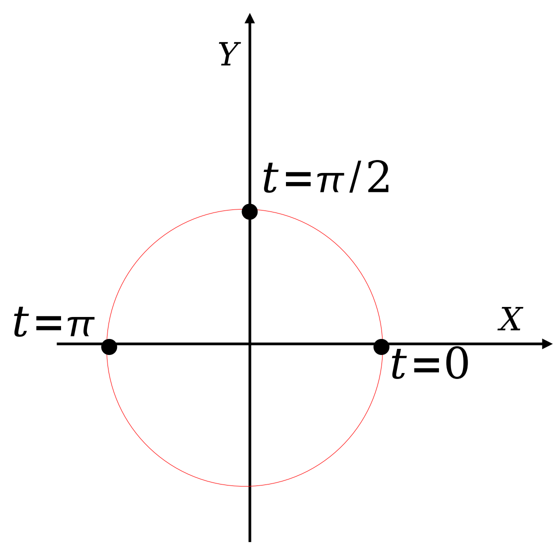
curve representation

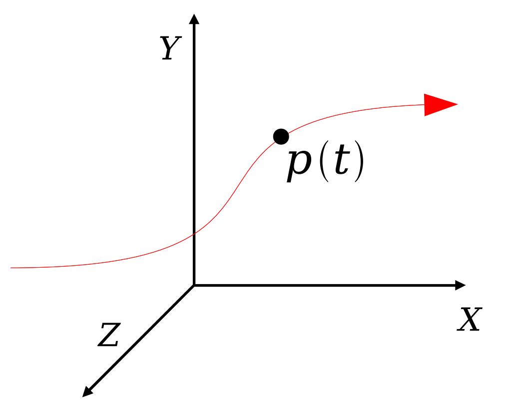
Parametric: (x,y,z) = (x(t), y(t),z(t))

. separate equation for each spatial value

. easy to generate points

. replace the use of slopes (may be ∞) with parametric tangent vectors





$$p(t)=[x(t),y(t),z(t)]^T$$

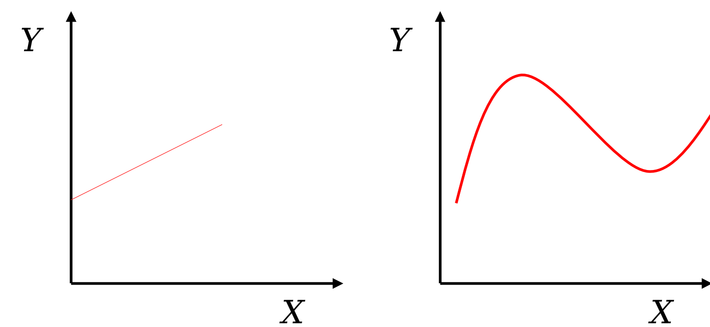
curve representation: summary

Explicit: y = f(x)

$y = mx + b$

. easy to generate points

. limitation: vertical lines, circles

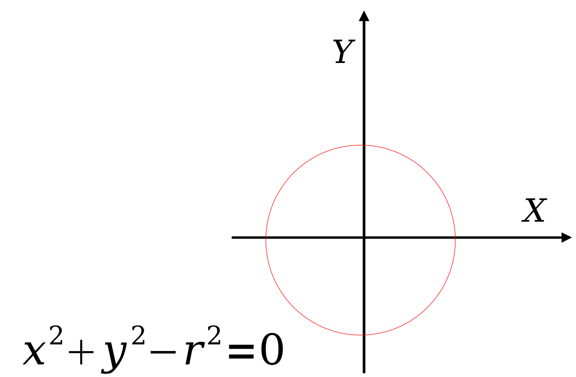


Implicit: f(x,y,z) = 0

. easy to test if point on the curve

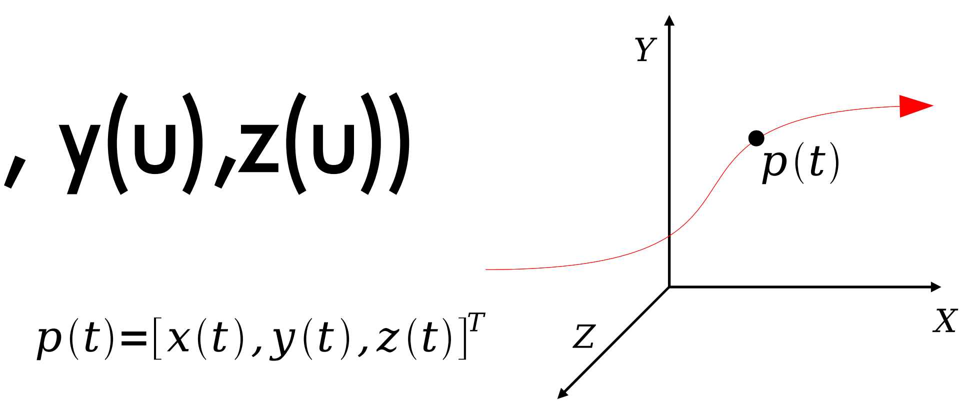
. normals are easy to compute

. hard to generate points



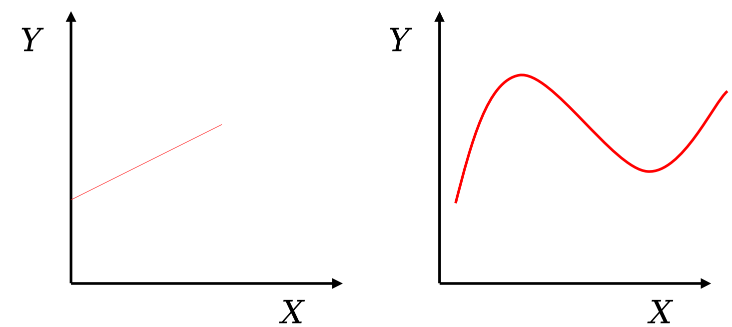
Parametric: (x,y,z) = (x(u), y(u),z(u))

. easy to generate points



curve representation: summary

Explicit: y = f(x)

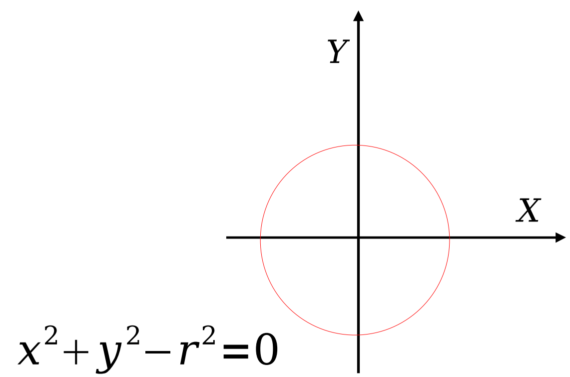


$y = mx + b$

. easy to generate points

. limitation: vertical lines, circles

Implicit: f(x,y,z) = 0



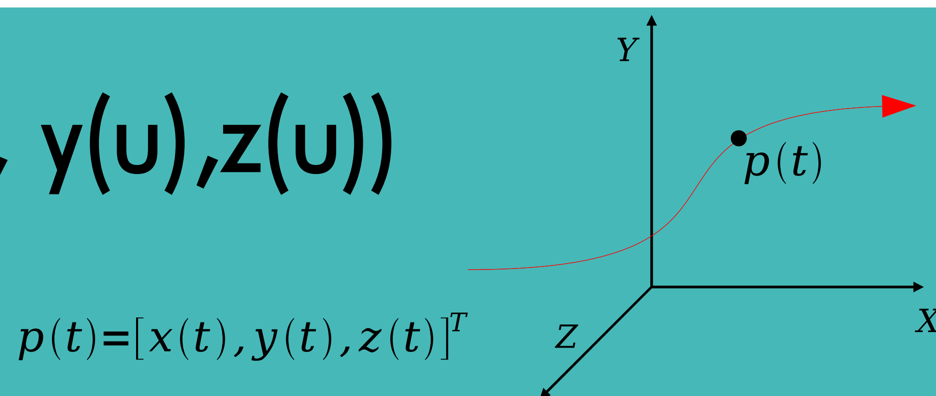
. easy to test if point on the curve

. normals are easy to compute

. hard to generate points

Parametric: (x,y,z) = (x(u), y(u),z(u))

. easy to generate points



parametric curves

use:

. move the viewer or object along a predefined path (changes in position and orientation)

. render hair

http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter23.html

Hubert Nguyen, William Donnely, Hair Animation and Rendering in the Nalu Demo, NVIDIA Corporation, Ch. 23, GPU Gems 2

http://www.youtube.com/watch?v=0RBqpQhj4X8

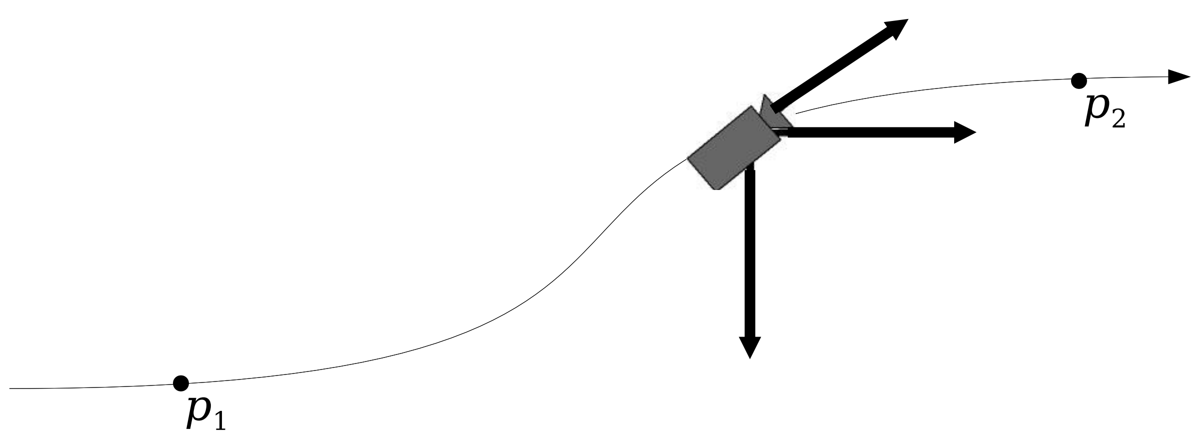
parametric curves

example:

. assume a camera should move between 2 points in one second.

. rendering 1 frame takes 50 ms

=> we will be able to render 20 frames in one second



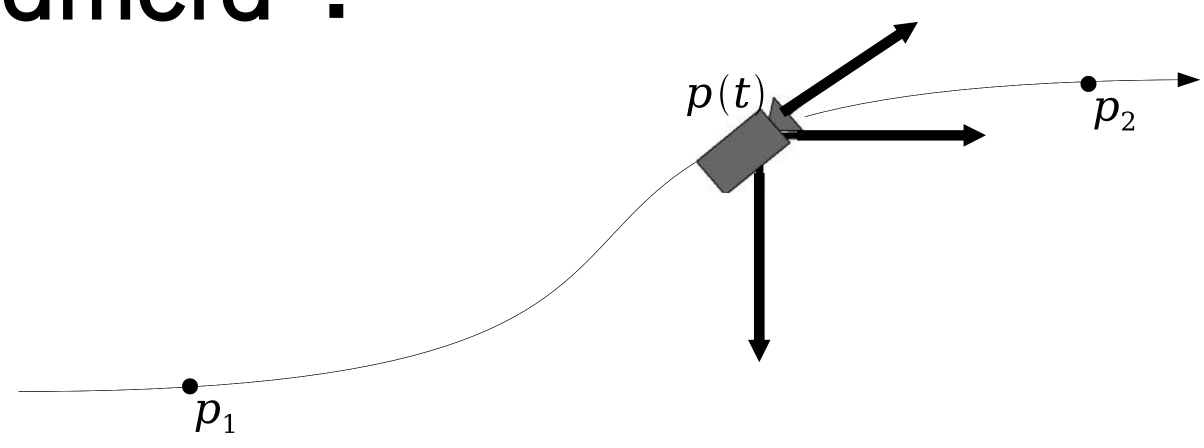
parametric curves

example:

. assume a camera should move between 2 points in one second.

. <u>NOW</u>, if 1 frame takes 25 ms, and we are able to render 40 frames in 1 second

$=>$ **to how many locations we need to move the camera ?**



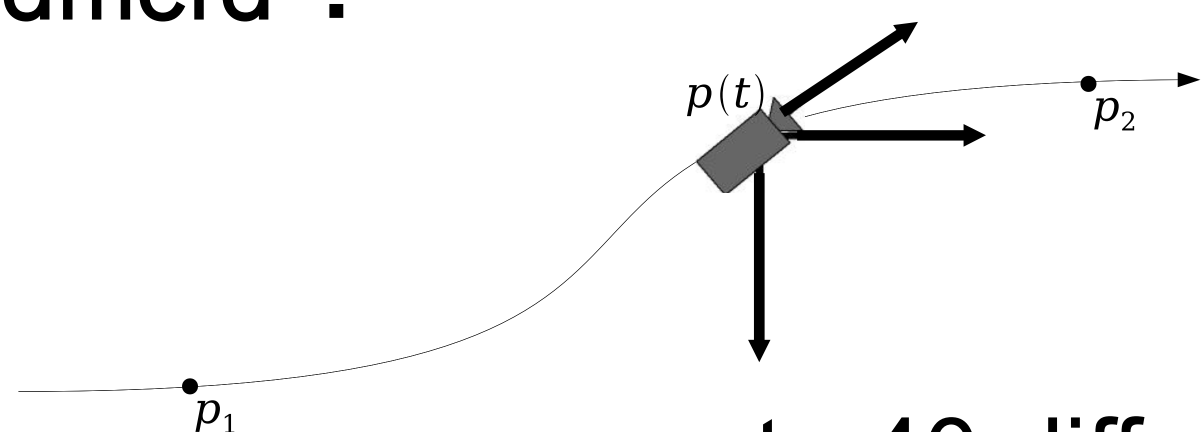
parametric curves

example:

. assume a camera should move between 2 points in one second.

. <u>NOW</u>, if 1 frame takes 25 ms, and we are able to render 40 frames in 1 second

$=>$ **to how many locations we need to move the camera ?**



to 40 different locations

we conclude

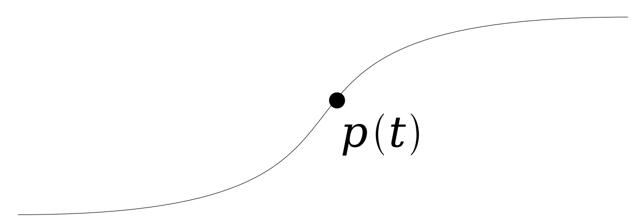
a parametric curve describes points using some formula as a function of a parameter t

parametric curves

we conclude

a parametric curve describes points using some formula as a function of a parameter t

$p(t)$ → returns a point for each value of t



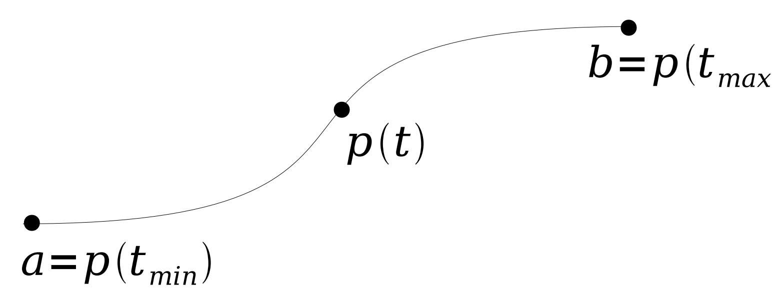
parametric curves

we conclude

a parametric curve describes points using some formula as a function of a parameter t

$p(t)$ → returns a point for each value of t

$t \in [a, b]$ → domain interval

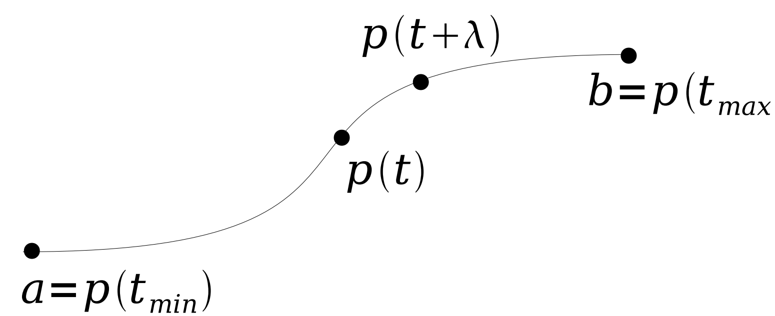


parametric curves

we conclude

a parametric curve describes points using some formula as a function of a parameter t

$p(t)$ → returns a point for each value of t

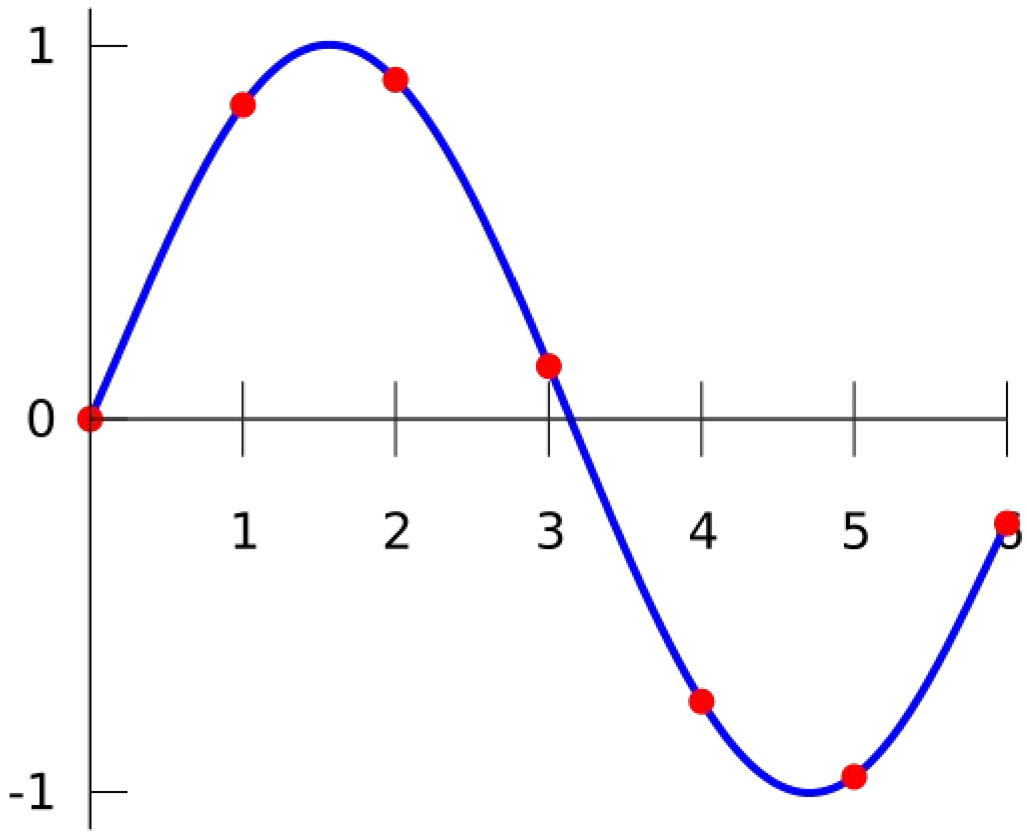


$t \in [a, b]$ → domain interval

$\lambda \Rightarrow 0$ then $p(t+\lambda) \Rightarrow p(t)$

If λ is a very small number, then $p(t)$ and $p(t+\lambda)$ are two points very close to each other

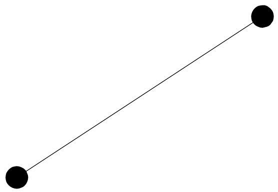
curves: polynomial interpolation



curves: polynomial interpolation

1st degree $y = ax + b$

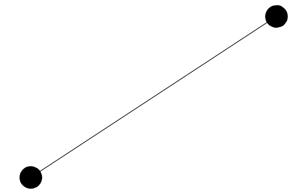
exact fit through 2 points



curves: polynomial interpolation

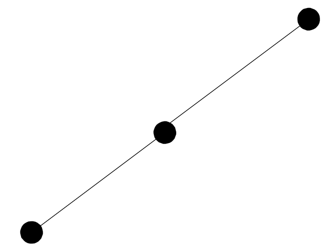
1st degree $y = ax + b$

exact fit through 2 points



2nd degree $y = ax^2 + bx + c$

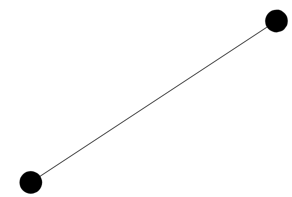
exact fit 3 points



curves: polynomial interpolation

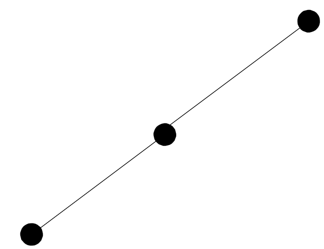
1st degree $y = ax + b$

exact fit through 2 points



2nd degree $y = ax^2 + bx + c$

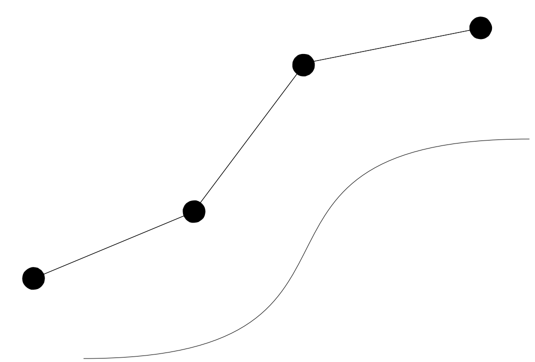
exact fit 3 points



3rd degree $y = ax^3 + bx^2 + cx + d$

exact fit 4 <u>**points**</u> or <u>**constraints**</u>

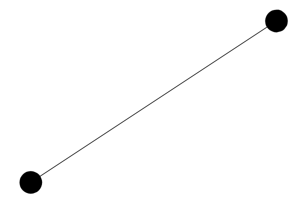
(constrain: point, curvature, angle)



curves: polynomial interpolation

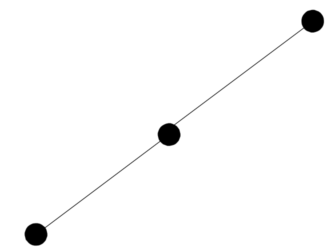
1st degree $y = ax + b$

exact fit through 2 points



2nd degree $y = ax^2 + bx + c$

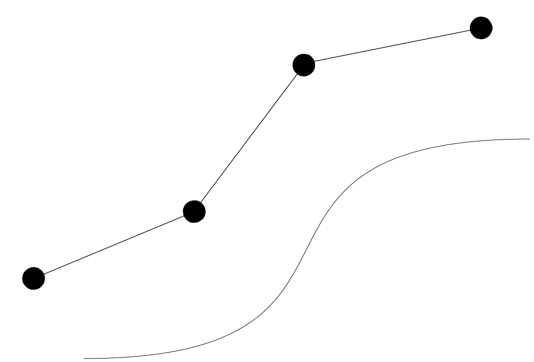
exact fit 3 points



3rd degree $y = ax^3 + bx^2 + cx + d$

exact fit 4 **points** or **constraints**

(constrain: point, curvature, angle)



n- degree … n+1 constraints

curves: polynomial interpolation

lower degree (eg. 2nd degree - quadratic)

. little flexibility to control the shape of the curve

. changing one control points affects <u>**all**</u> curve

. few degrees of freedom

curves: polynomial interpolation

lower degree (eg. 2^{nd} degree - quadratic)
. little flexibility to control the shape of the curve
. changing one control points affects **all** curve
. few degrees of freedom

high degree (eg. 4^{th} degree - quartic)
. required more computation
. too many degrees of freedom, then hard to control, high oscillatory.

which is the best approach?

curves: polynomial interpolation

lower degree (eg. 2nd degree - quadratic)

. little flexibility to control the shape of the curve
. changing one control points affects <u>**all**</u> curve
. few degrees of freedom

high degree (eg. 4th degree - quartic)

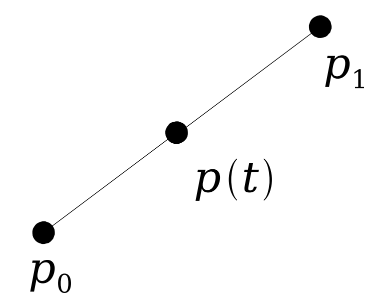
. required more computation
. too many degrees of freedom, then hard to control, high oscillatory.

cubic polynomial

Bézier curves

<u>linear interpolation</u>:
straight line between 2 points, p_0 and p_1

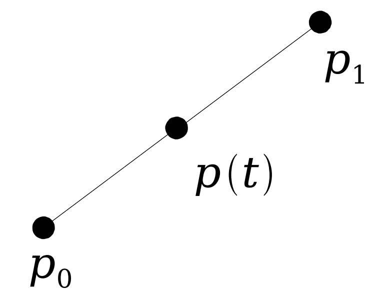
$$p(t) = p_0 + t(p_1 - p_0) \qquad t \in [0,1]$$
$$= (1-t)p_0 + t(p_1)$$



Bézier curves

<u>linear interpolation</u>:
straight line between 2 points, p_0 and p_1

$$p(t) = p_0 + t(p_1 - p_0) \qquad t \in [0,1]$$
$$= (1-t)p_0 + t(p_1)$$



$p(t)$: controls where on the line the point $p(t)$ will land

$p(0) = p_0$, $p(1) = p_1$ and $0 < t < 1$

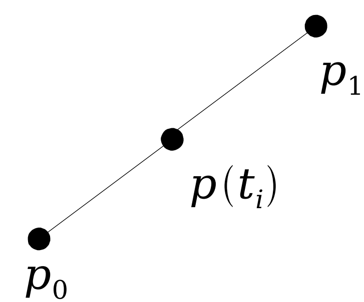
Bézier curves

<u>**example**</u>:

if you would like to move the camera from p_0 to p_1 linearly in 20 steps during 1 second which are the values for t ?

$$t_i = i/(20-1)$$

$$p(t_i) = (1-t_i)p_0 + t_i(p_1)$$



$$t_i \in [0,1]$$

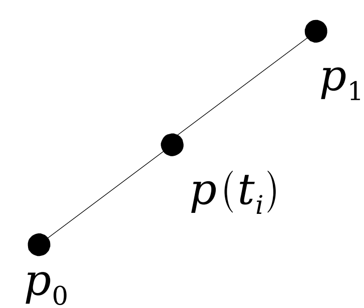
i: frame number

Bézier curves

example:

if you would like to move the camera from p_0 to p_1 linearly in 20 steps during 1 second which are the values for t ?

$$t_i = i/(20-1)$$



$$p(t_i) = (1-t_i)p_0 + t_i(p_1)$$

$$t_i \in [0,1]$$

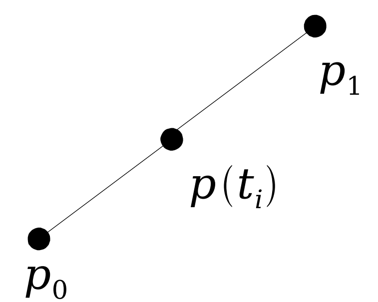
i: frame number

but, for more points on a path what happens?

Bézier curves

<u>example</u>:

if you would like to move the camera from p_0 to p_1 linearly in 20 steps during 1 second which are the values for t ?

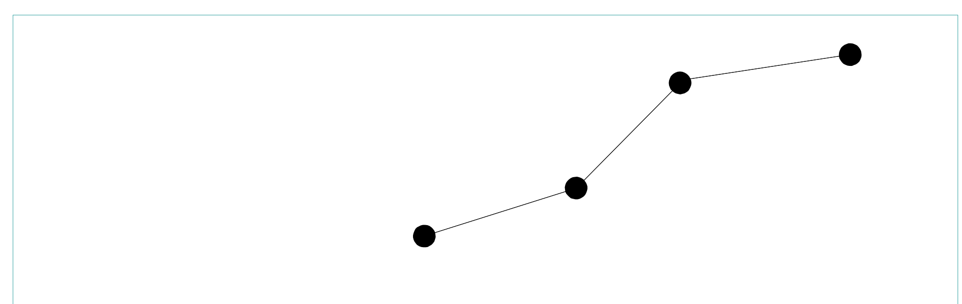


$$t_i = i/(20-1)$$

$$p(t_i) = (1-t_i)p_0 + t_i(p_1)$$

$$t_i \in [0,1]$$

i: frame number



Bézier curves

example:

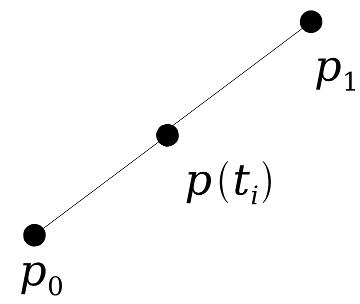
if you would like to move the camera from p_0 to p_1 linearly in 20 steps during 1 second which are the values for t ?

$$t_i = i/(20-1)$$

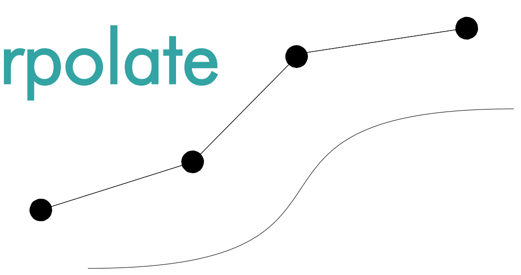
$$p(t_i) = (1-t_i)p_0 + t_i(p_1)$$

$$t_i \in [0,1]$$

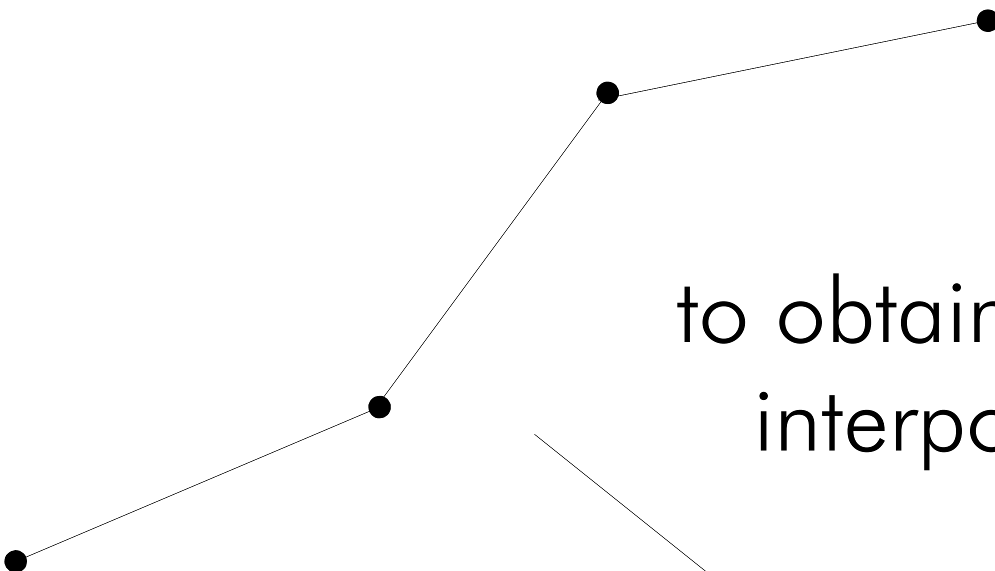
i: frame number



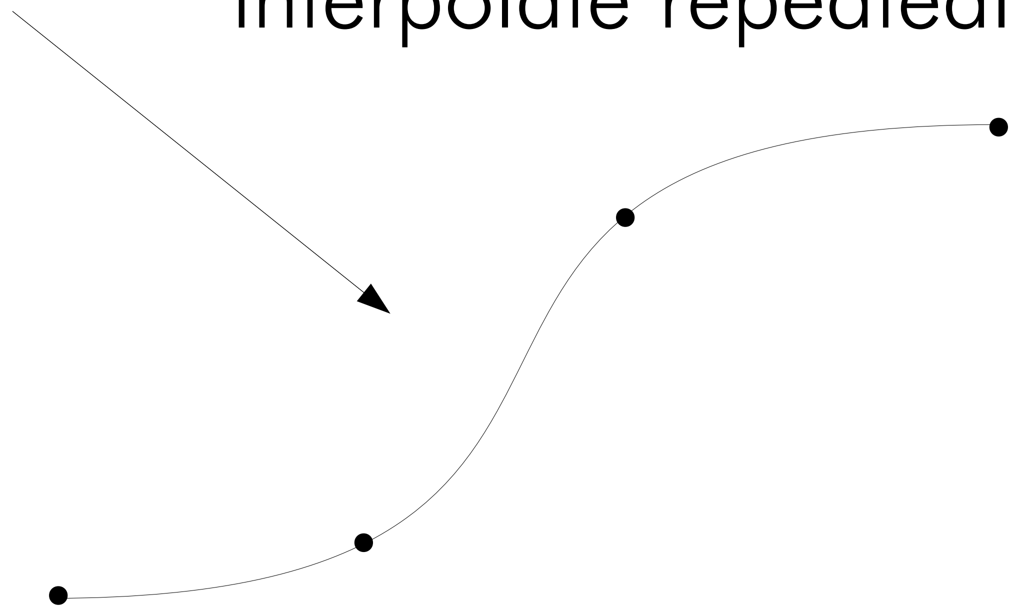
linearly interpolate repeatedly



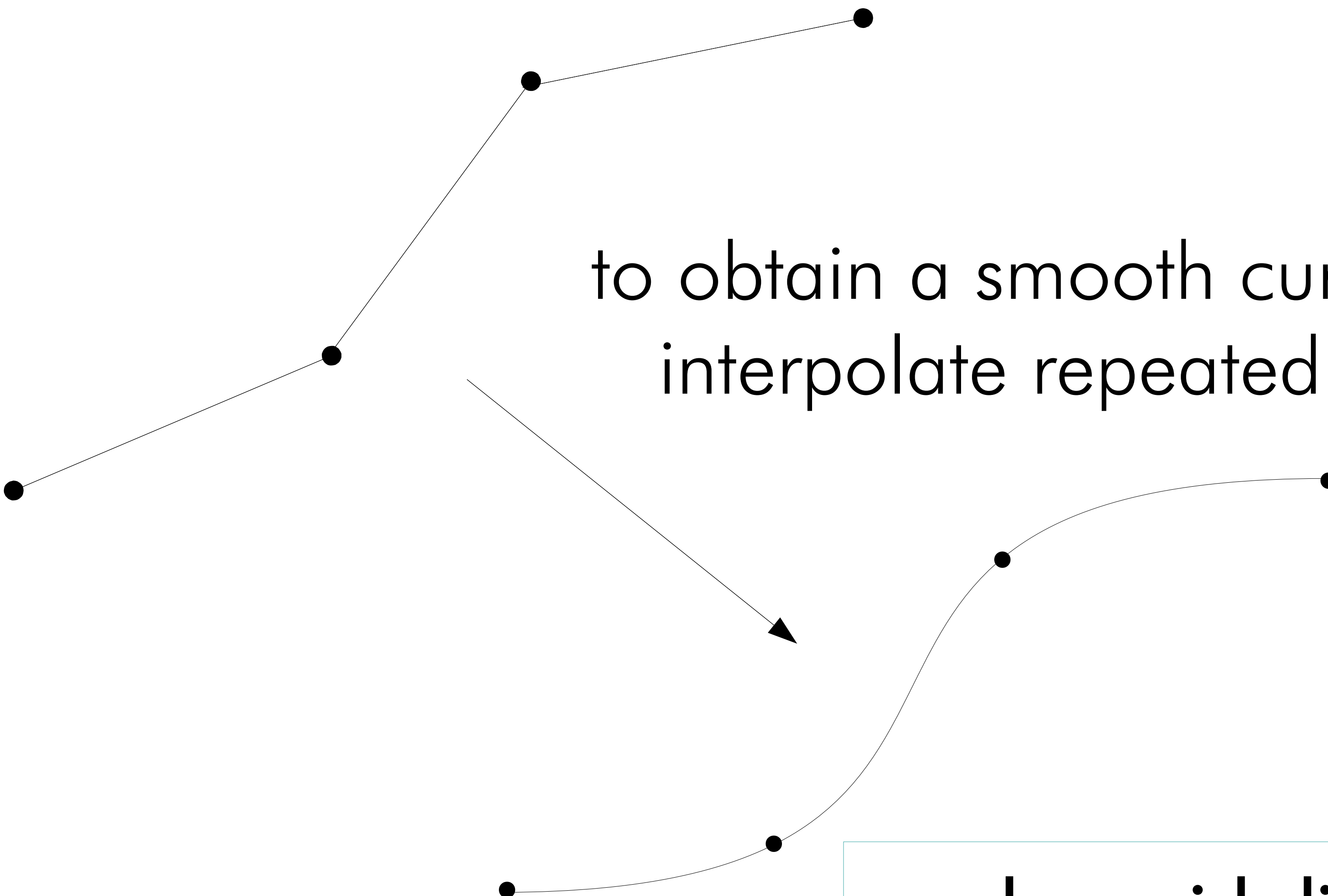
Bézier curves



to obtain a smooth curve:
interpolate repeatedly



Bézier curves

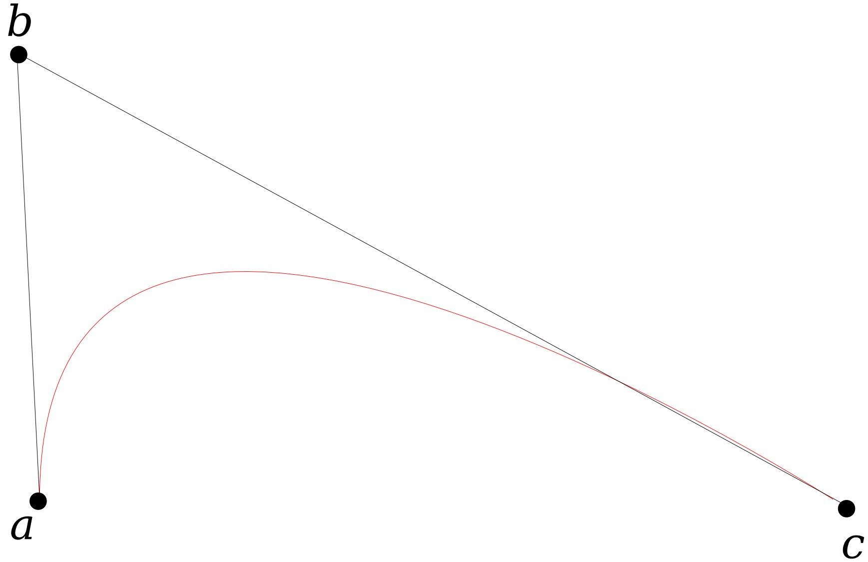


to obtain a smooth curve:
interpolate repeatedly

goal: avoid discontinuity
at the joints

Bézier curves

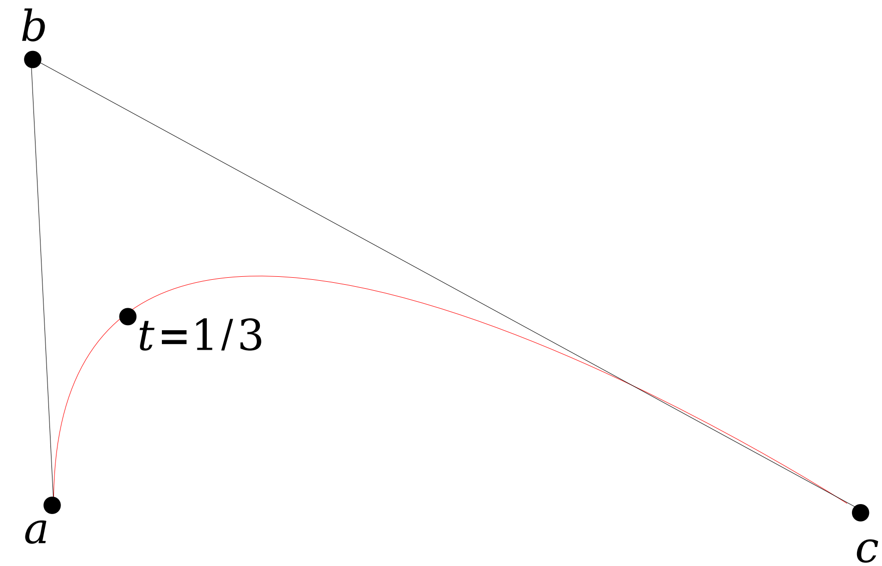
0) curve defined by 3 control points: a, b, c



Bézier curves

0) curve defined by 3 control points: a, b, c

1) we want to find the point on the curve for parameter $t = 1/3$

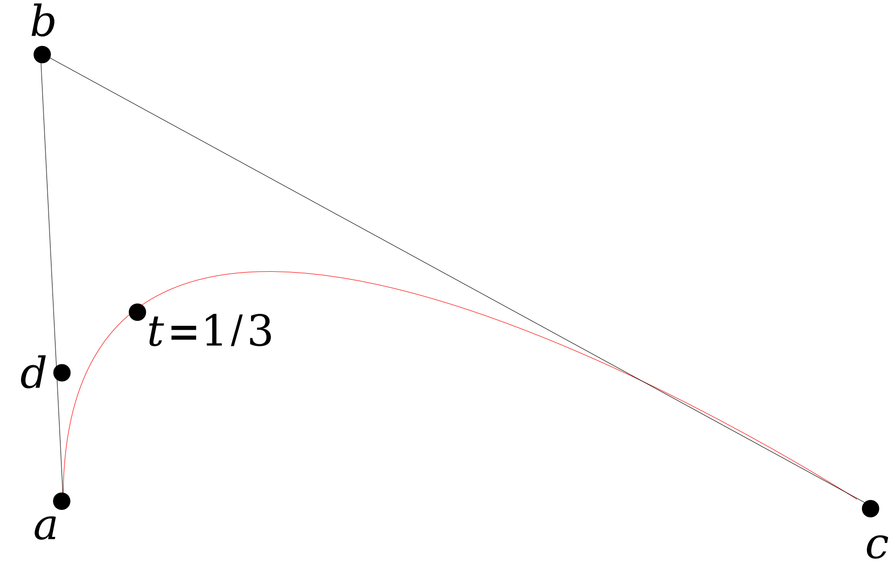


Bézier curves

0) curve defined by 3 control points: a, b, c

1) we want to find the point on the curve for parameter $t = 1/3$

2) linearly interpolation between a and b to get d



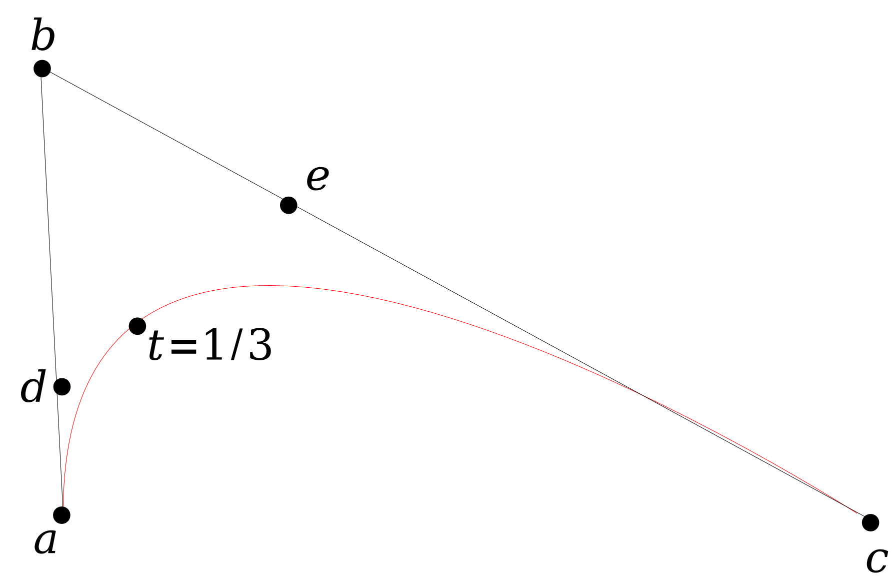
Bézier curves

0) curve defined by 3 control points: a, b, c

1) we want to find the point on the curve for parameter $t = 1/3$

2) linearly interpolation between a and b to get d

3) linearly interpolation between b and c to get e



Bézier curves

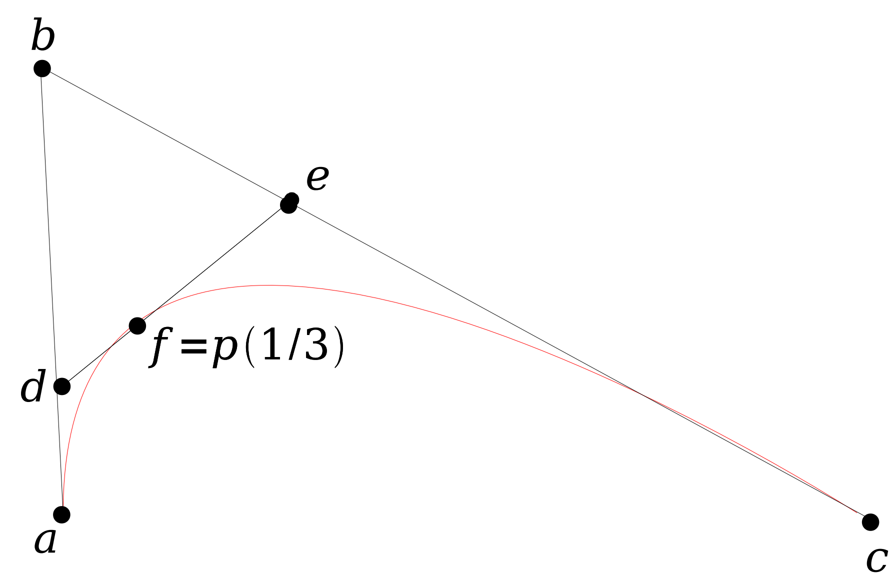
0) curve defined by 3 control points: a, b, c

1) we want to find the point on the curve for parameter $t = 1/3$

2) linearly interpolation between a and b to get d

3) linearly interpolation between b and c to get e

4) the point $p(1/3) = f$ is found by interpolating d and e



Bézier curves

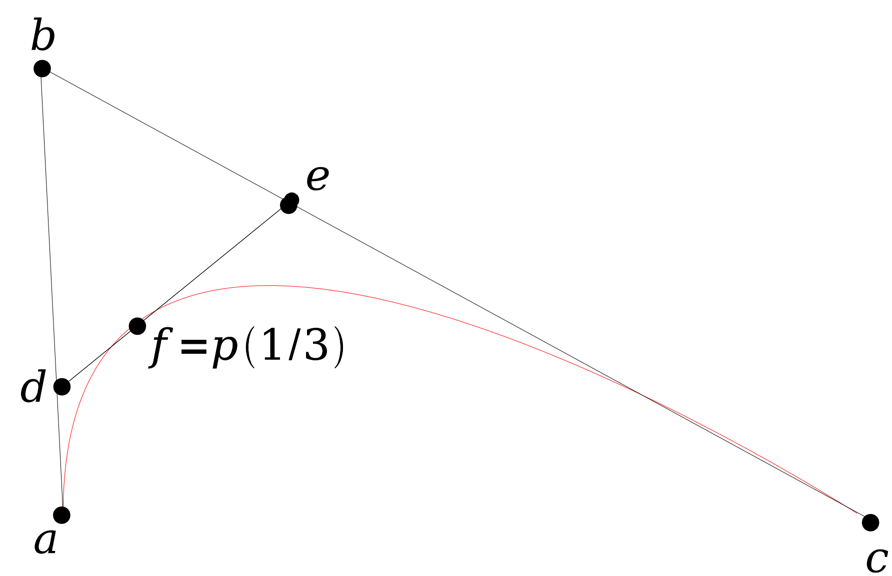
0) curve defined by 3 control points: a, b, c

1) we want to find the point on the curve for parameter $t = 1/3$

2) linearly interpolation between a and b to get d

3) linearly interpolation between b and c to get e

4) the point $p(1/3) = f$ is found by interpolating d and e



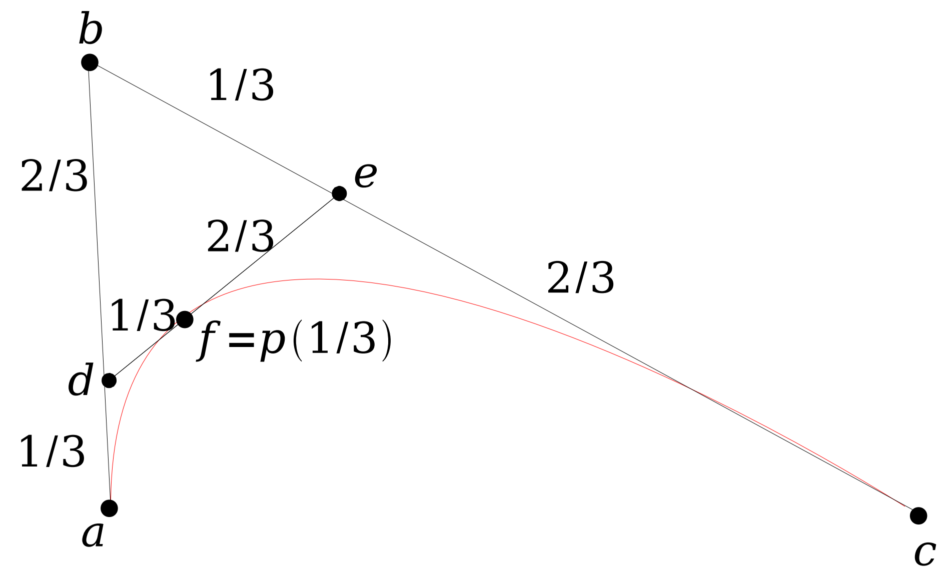
General: $p(t) = f$

Bézier curves

$$
\begin{aligned}
p(t) &= (1-t)d + te \\
&= (1-t)[(1-t)a + tb] + t[(1-t)b + tc] \\
&= (1-t)^2 a + 2(1-t)tb + t^2 c
\end{aligned}
$$

we obtain a **parabola**, the maximum degree of t is 2 (quadratic)

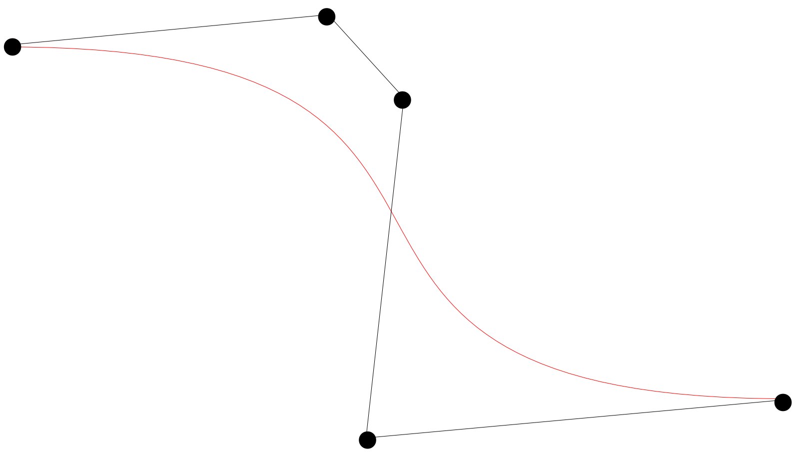
given $n+1$ control points, the degree of the curve is n



more control points gives the curve **more degrees of freedom**

Bézier curves

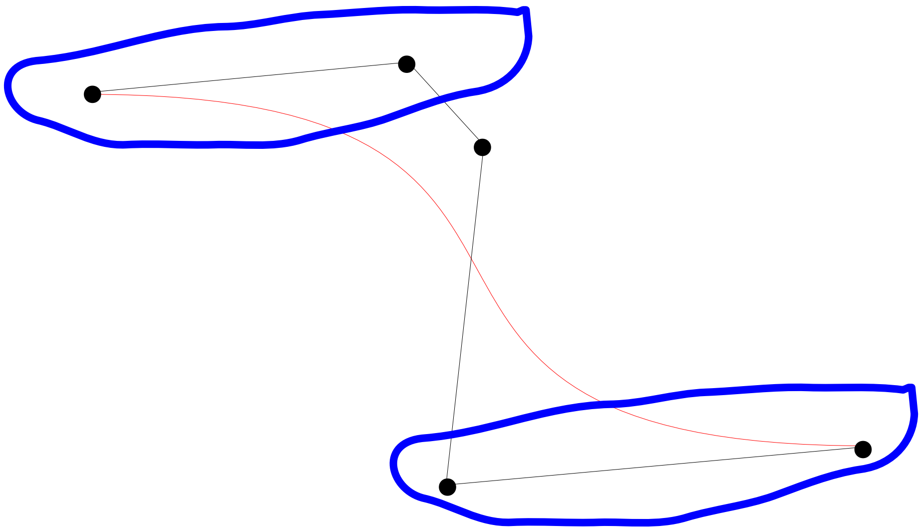
repeated linear interpolation from 5 control points, gives a 4^{th} degree curve (quartic)



Bézier curves

repeated linear interpolation from 5 control points, gives a 4th degree curve (quartic)

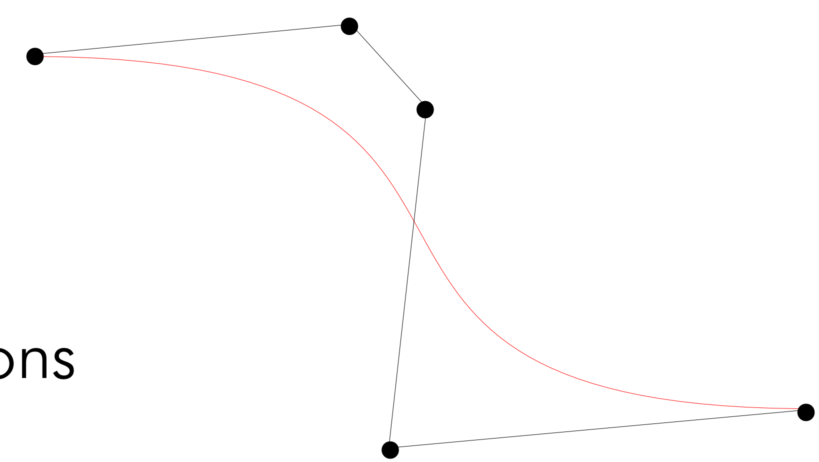
at the 1st point the curve is tangent to the line between the 1st and 2nd point. Same to the end of the curve



Bézier curves

repeated linear interpolation from 5 control points, gives a 4th degree curve (quartic)

at the 1st point the curve is tangent to the line between the 1st and 2nd point. Same to the end of the curve



$$p_i^k(t) = (1-t)\,p_i^{k-1}(t) + t\,p_{i+1}^{k-1}(t)$$

$k = 1 \ldots \mathrm{n}$ → # of linear interpolations

$i = 0 \ldots \mathrm{n}-k$ → # of control points

p_i^k → Intermediate control points

$p(t) = p_0^n(t)$ → describes a point on the curve

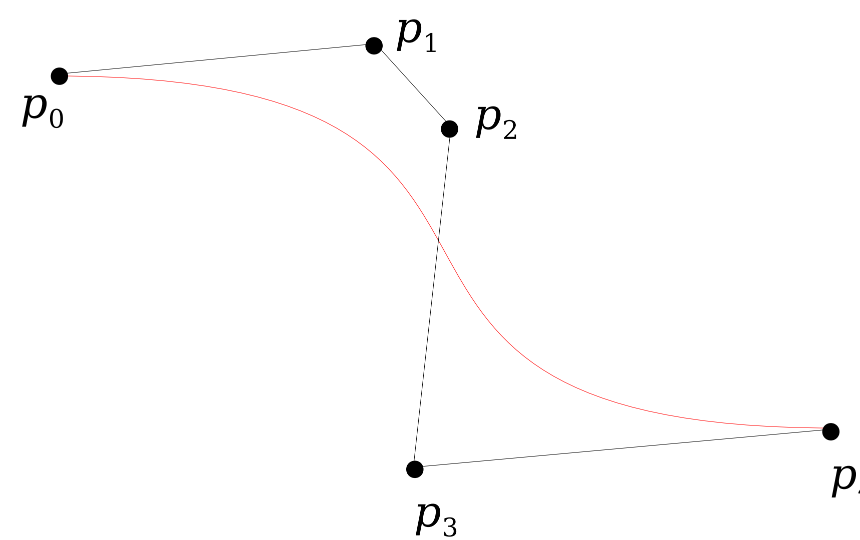
Bézier curves

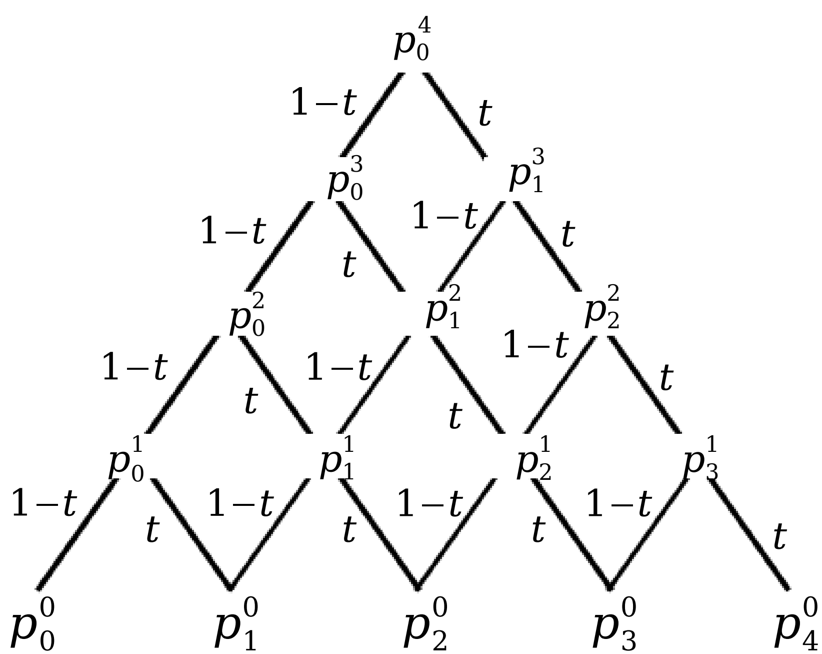
$$p_i^k(t) = (1-t)\,p_i^{k-1}(t) + t\,p_{i+1}^{k-1}(t)$$

For $k=1 \rightarrow p_0^1 = (1-t)\,p_0^0 + t\,p_1^0$

$p_1^1 = (1-t)\,p_1^0 + t\,p_2^0$

For $k=2 \rightarrow p_0^2 = (1-t)\,p_0^1 + t\,p_1^1$





read diagram from bottom to top (quartic, 5 control points)

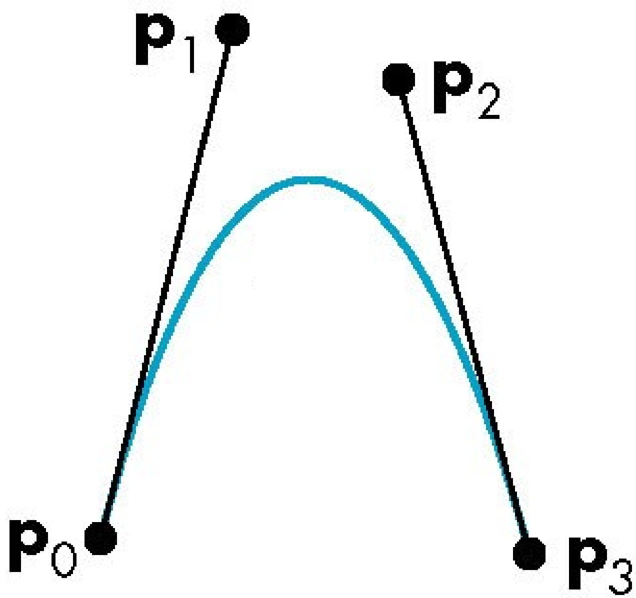
$k = 1 \ldots n$ # of linear interpolations

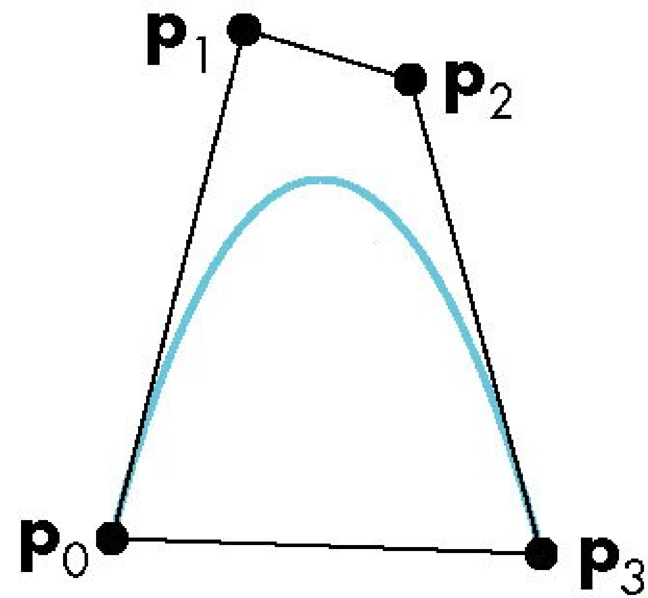
Bézier curves

. control points: $p_{0,}p_{1,}p_{2,}p_3$

. $p_{1,}p_2$ are used to calculate the tangent

. the curve only pass through the end points



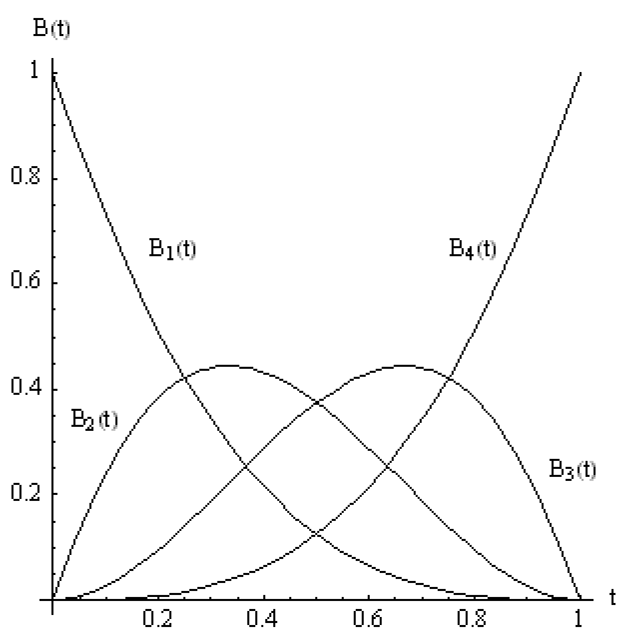


all points of curve inside
<u>**convex hull**</u> of control points

Bézier curve: cubic polynomial

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

$$p(t)=\begin{pmatrix}1 & t & t^2 & t^3\end{pmatrix}\begin{pmatrix}1 & 0 & 0 & 0\\ -3 & 3 & 0 & 0\\ 3 & -6 & 3 & 0\\ -1 & 3 & -3 & 1\end{pmatrix}\begin{pmatrix}p_0\\ p_1\\ p_2\\ p_3\end{pmatrix}$$



cubic blending function $n=3$

Bézier curve: cubic polynomial

some interesting properties:

. you can directly rotate the control points and then compute the curve, instead of computing points on a Bezier and then rotating <u>(MUCH FASTER)</u>

. uses DOT PRODUCT instead of SCALAR operations

Bézier curve: cubic polynomial

downside:

. curve dosen't pass through all the control points

which can be a possible solution?

Bézier curve: cubic polynomial

downside:

. curve dosen't pass through all the control points

which can be a possible solution?

. use a lower degree curve between each pair of subsequent control points.

. check if the piecewise interpolation has high enough degree of continuity.

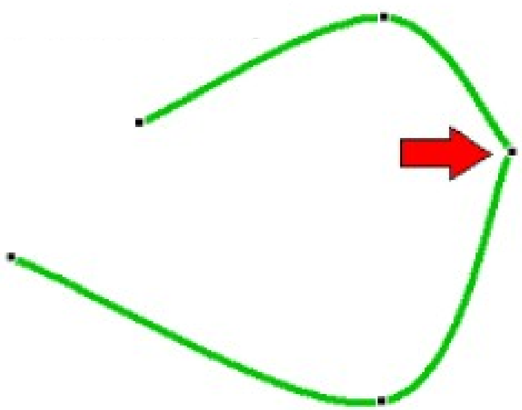
piecewise polynomials

join curves or curve segments **nicely**

piecewise polynomials

join curves or curve segments **nicely**

C^0 continuity

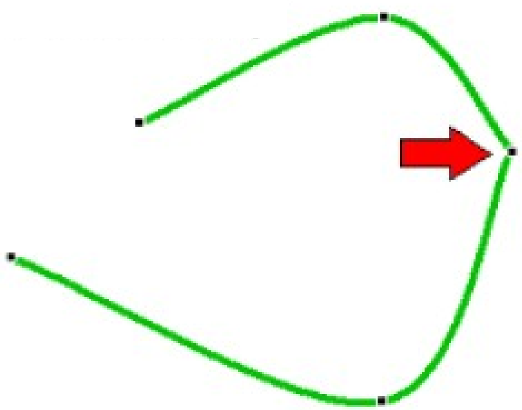


continuous in
position

piecewise polynomials

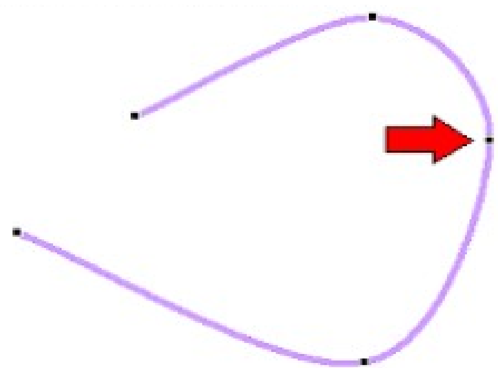
join curves or curve segments **nicely**

C^0 continuity



continuous in **position**

$C^0 \wedge C^1$ continuity

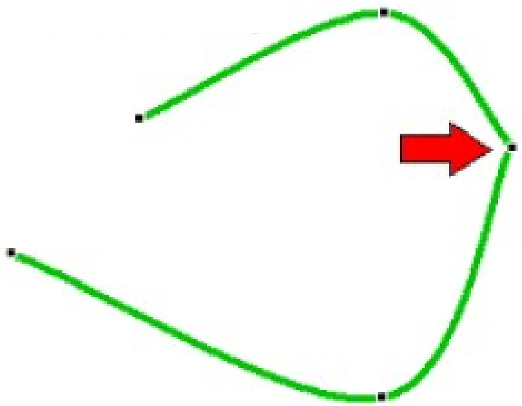


continuous in **position** and **tangent vector**

piecewise polynomials

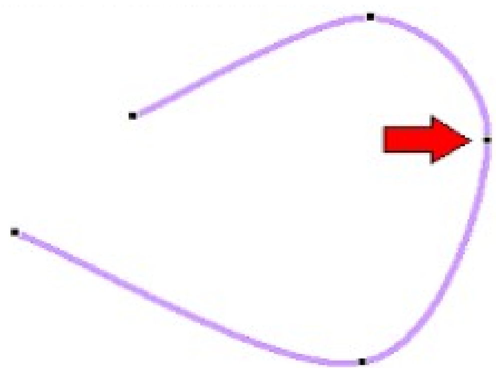
join curves or curve segments **nicely**

C^0 continuity



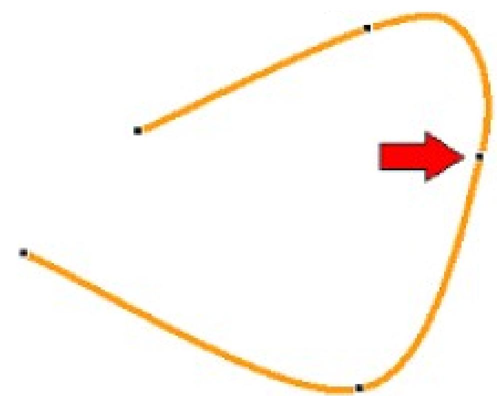
continuous in
position

$C^0 \wedge C^1$ continuity



continuous in
position and
tangent vector

$C^0 \wedge C^1 \wedge C^2$ continuity

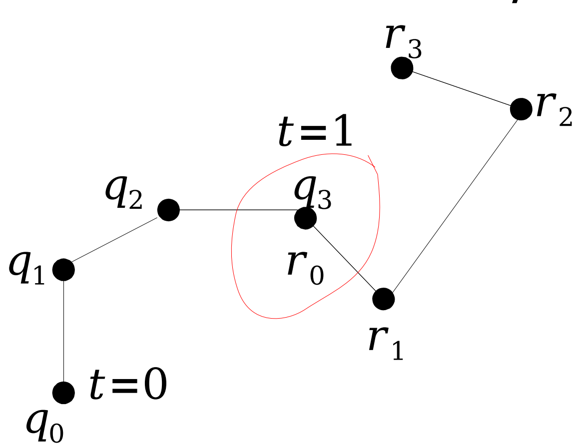


continuous in
position,
tangent vector
and **curvature**

piecewise polynomials

sudden jerk at the join

C^0 continuity



continuous in **position**

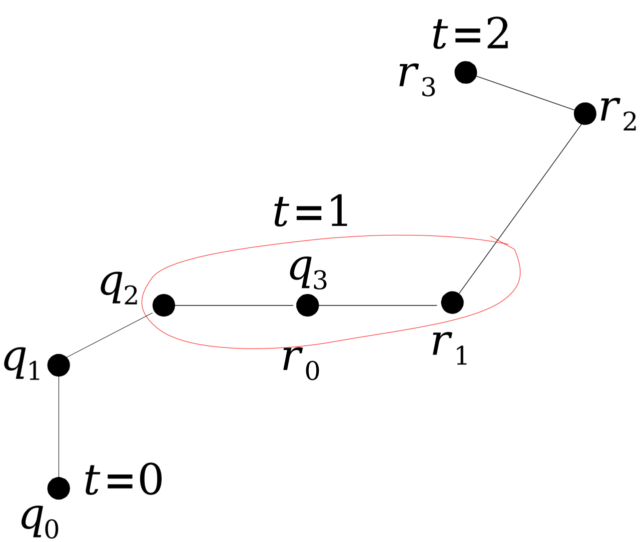
the segment should join at the same point, so linear interpolation fulfills this condition

$$\boxed{q_3 = r_0}$$

piecewise polynomials

tangents at the join parallel and equal in length

G^1 continuity



continuous in **position** and **tangent vector**

must be parallel and have the same direction, nothing about the length

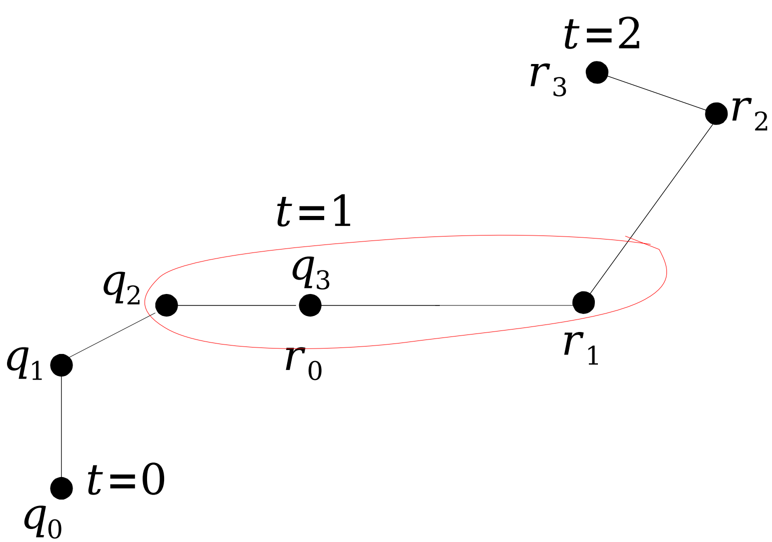
$$\boxed{(r_1 - r_0) = c(q_3 - q_2) \quad \text{for} \quad c > 0}$$

$$c = (t_2 - t_1)/(t_1 - t_0)$$

piecewise polynomials

tangents at the join parallel and double in length

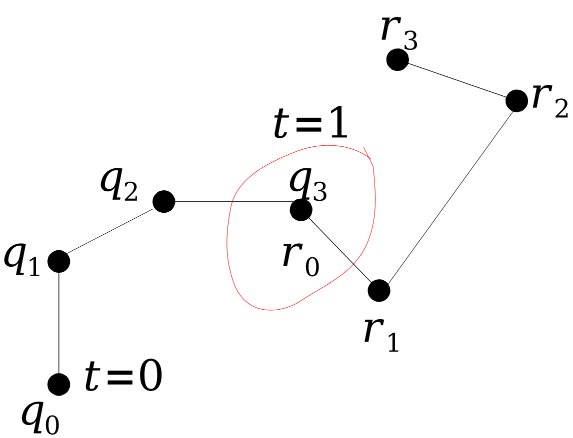
C^1 continuity



continuous in **position** and **tangent vector,** stronger than G^1

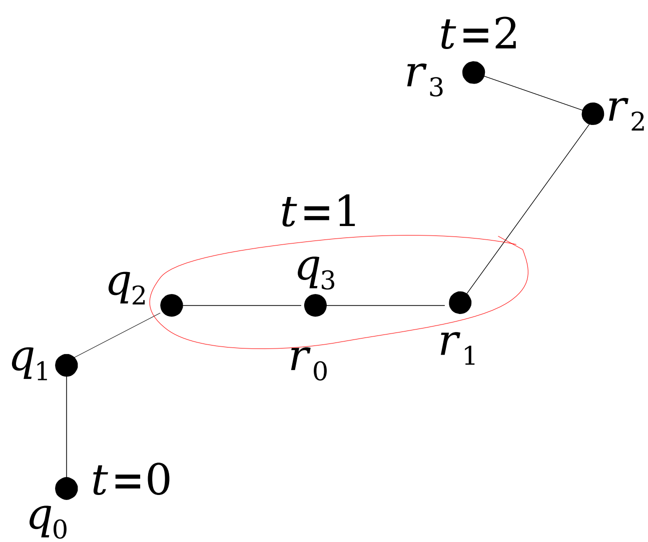
piecewise polynomials

C^0 continuity



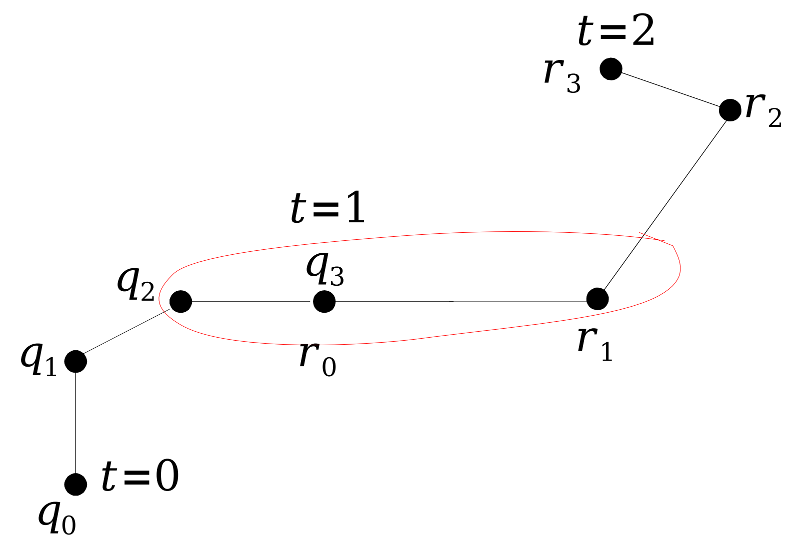
continuous in **position**

G^1 continuity



continuous in **position** and **tangent vector**

C^1 continuity



continuous in **position** and **tangent vector,** stronger than G_1

other curves

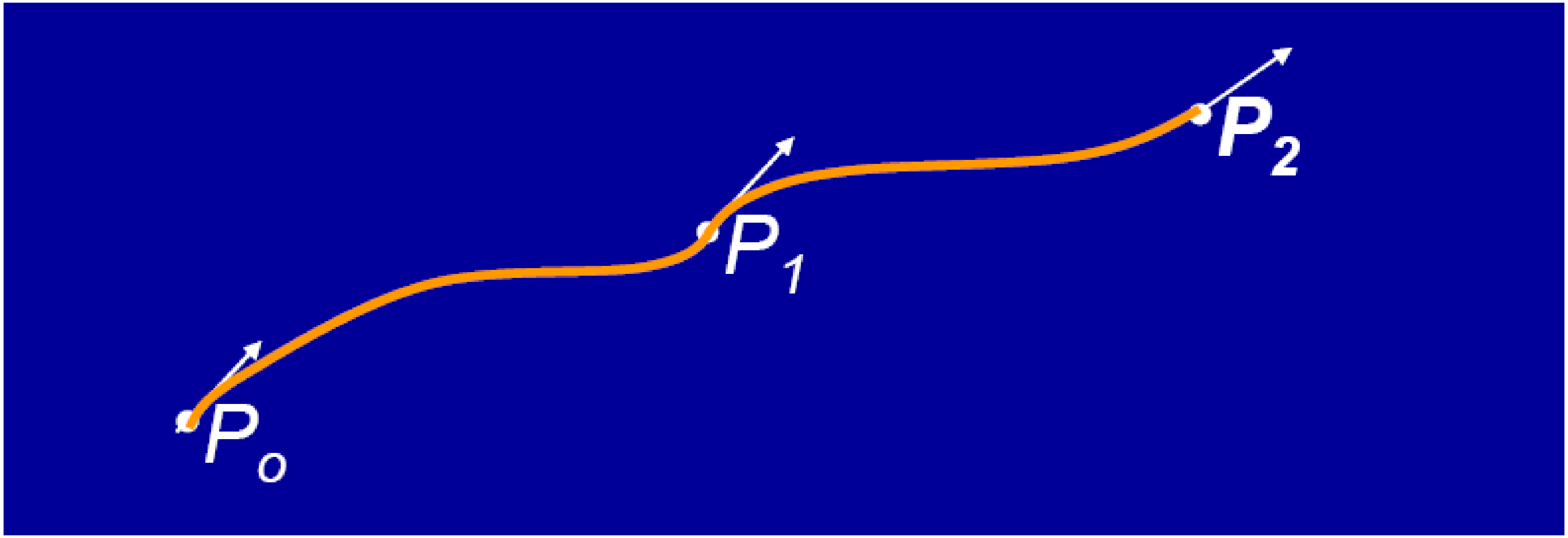
. Hermite Splines
. Catmull-Rom Splines
. Natural Cubic Splines
. B-Splines
. NURBS

other curves

- Hermite Splines
- Catmull-Rom Splines
- Natural Cubic Splines
- B-Splines
- NURBS

Hermite Splines or Cubic splines

simpler to control than Bezier

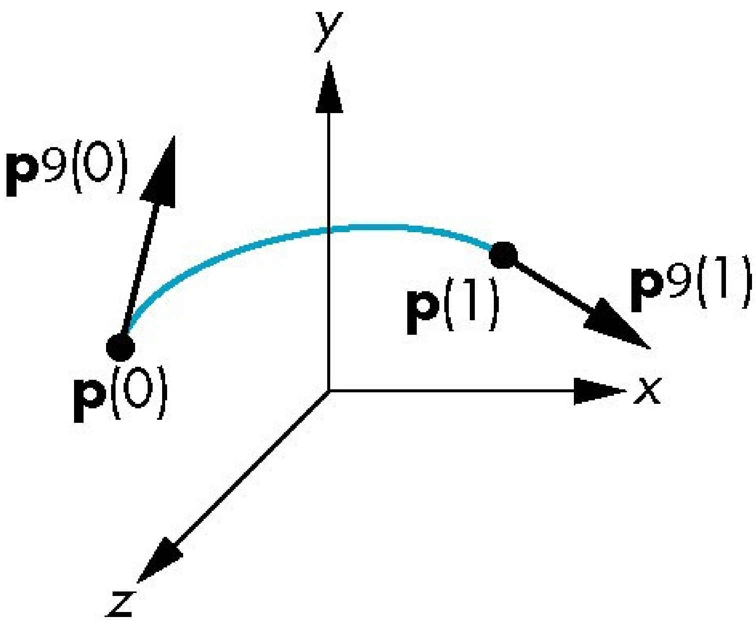


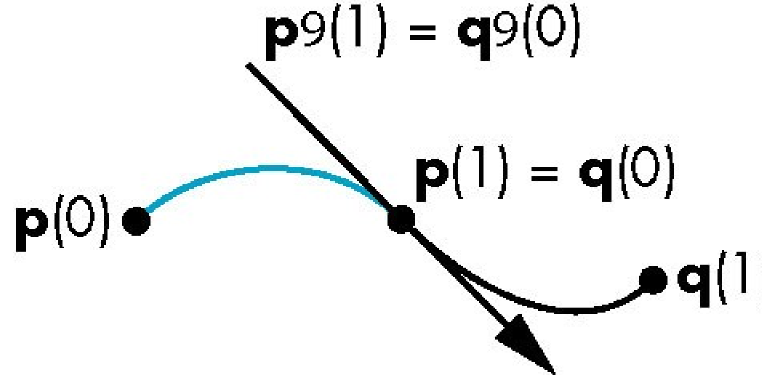
defined by:

. starting and end points

. and starting and end tangents

Hermite Splines



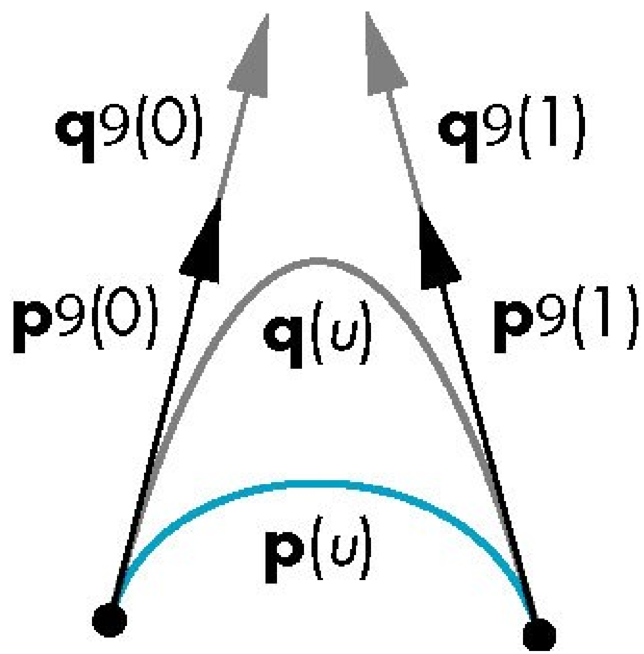


$$p(t)=(2t^3-3t^2+1)p_0+(t^3-2t^2+t)m_0+(t^3-t^2)m_1+(-2t^3+3t^2)p_1$$

$$p(0)=p_{0,}\, p(1)=p_1$$

$$(\partial p/\partial t)(0)=m_{0,}(\partial p/\partial t)(0)=m_{1,}$$

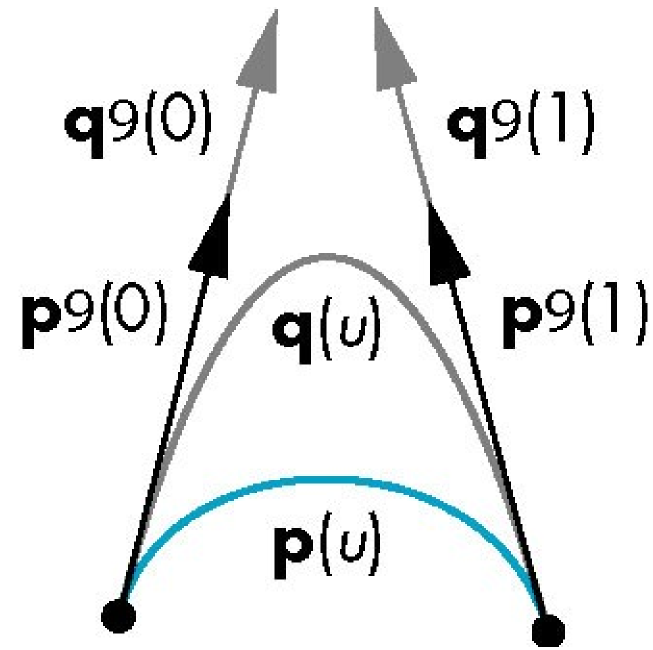
Hermite Splines or Cubic splines



why Hermit Splines are cubic interpolation?

$$p(t) = (2t^3 - 3t^2 + 1)p_0 + (t^3 - 2t^2 + t)m_0 + (t^3 - t^2)m_1 + (-2t^3 + 3t^2)p_1$$

Hermite Splines or Cubic splines

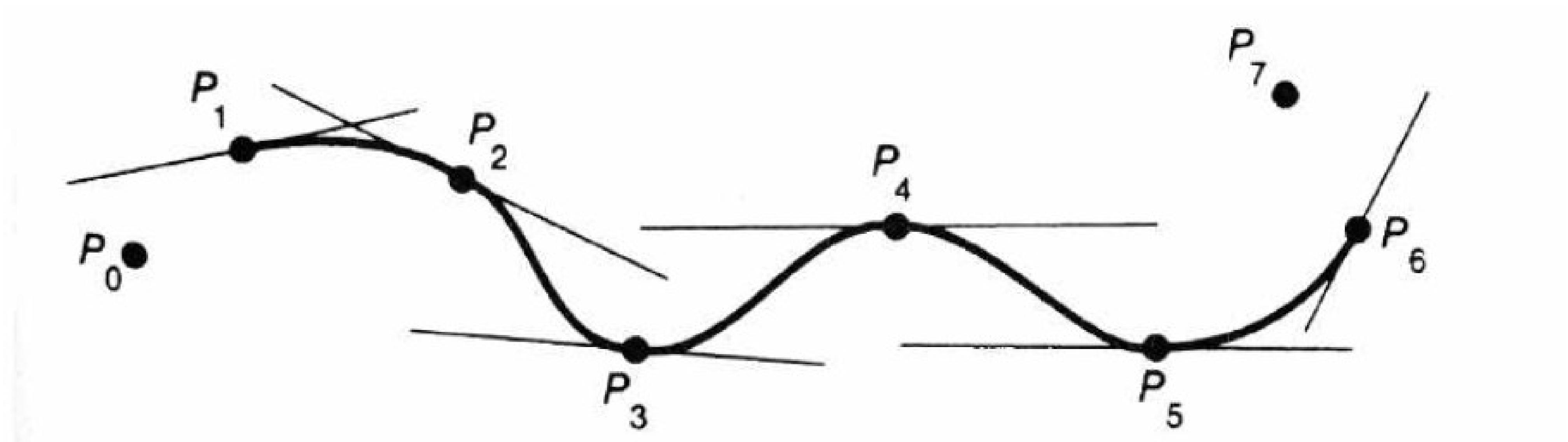


why Hermit Splines are cubic interpolation?

because the highest exponent on the blending function is t^3

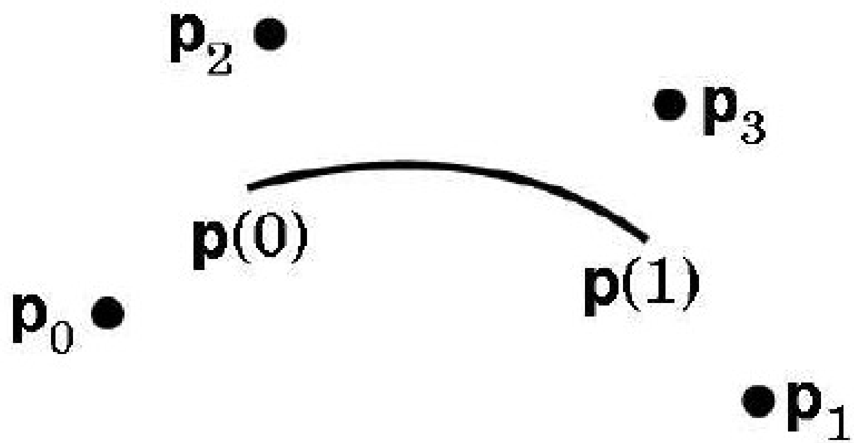
$$p(t)=(2\mathrm{t}^3-3\mathrm{t}^2+1)p_0+(t^3-2\mathrm{t}^2+t)m_0+(t^3-t^2)m_1+(-2\mathrm{t}^3+3\mathrm{t}^2)p_1$$

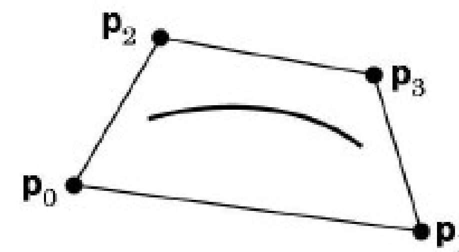
Catmull-Rom Splines



. the spline passes through all of the control points
. C^1 continuous, there are no discontinuities in the tangent direction and magnitude

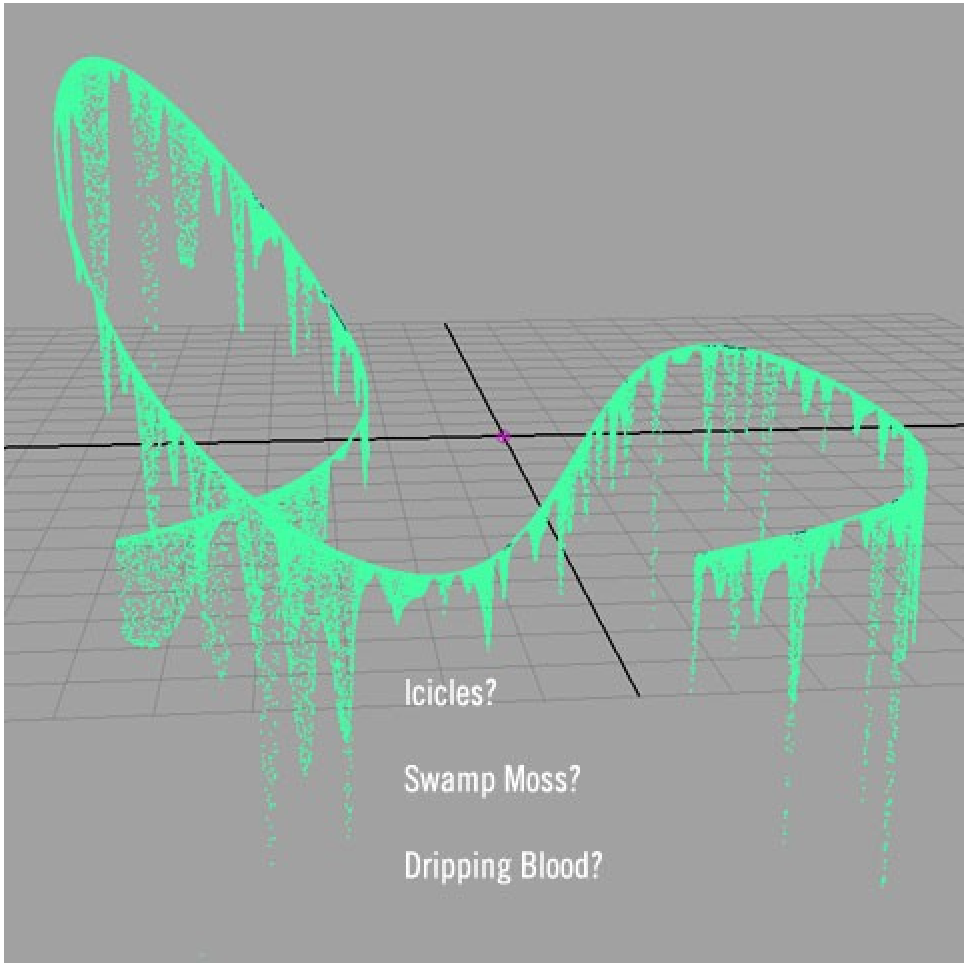
B-Splines





. no interpolation
. the curve passes near the control points (use interactive placement, it is hard to know where the curve will go)
. C^2 continuous to compensate the loss of interpolation

example effect + curve



http://www.digitalartform.com/archives/images/dripDemo.jpg