# Bezier curves

Victor Eijkhout

Notes for CS 594 – Fall 2004

# Fonts

- Use of Bezier in fonts
- Bitmap vs outline (vector)
- Curve descriptions are scalable, smaller

# Fonts

- Use of Bezier in fonts
- Bitmap vs outline (vector)
- Curve descriptions are scalable, smaller
- More processing
- More intelligence in the rasterizer

# Requirements

- Description unique, simple to compute
- Easy to change shape (design)
- Well behaved: small change in parameter gives small change in shape
- Smoothly composable

# Two basic problems:

- Points known, smooth curve required
- Function known, approximation required

# Interpolation

# Lagrange interpolation

- Given points $(x_1, f_1) \dots (x_n, f_n)$, draw curve
- $n$ points: polynomial of degree $n-1$ (actually, order)

$$p(x) = p_{n-1}x^{n-1} + \cdots + p_1 x + p_0$$

- equations $p(x_i) = f_i$ to solve

- System of equations

$$\begin{aligned} p_{n-1}x_1^{n-1} + \cdots + p_1x_1 + p_0 &= f_1 \\ &\cdots \\ p_{n-1}x_n^{n-1} + \cdots + p_1x_n + p_0 &= f_n \end{aligned}$$

written as $X\bar{p} = \bar{f}$, where

$$X = (x_i^j), \quad \bar{p} = \begin{pmatrix} p_1 \\ \vdots \\ p_{n-1} \end{pmatrix}, \quad \bar{f} = \begin{pmatrix} f_1 - p_0 \\ \vdots \\ f_n - p_0 \end{pmatrix}$$

- not stable

# Lagrange polynomials

- $p^{(k)}$:

$$p^{(k)}(x) = c_k(x - x_1)\cdots(x - x_{k-1})\,(x - x_{k+1})\cdots(x - x_n)$$

where $c_k$ is chosen so that $p^{(k)}(x_k) = 1$.

# Lagrange polynomials

- $p^{(k)}$:

  $$p^{(k)}(x) = c_k(x-x_1)\cdots(x-x_{k-1})(x-x_{k+1})\cdots(x-x_n)$$

  where $c_k$ is chosen so that $p^{(k)}(x_k) = 1$.
- $p^{(i)}(x_j) = \delta_{ij}$, so

$$p(x) = \sum_i f_i p^{(i)}(x), \qquad p^{(i)}(x) = \prod_{j\neq i} \frac{x-x_j}{x_i-x_j} \tag{1}$$

# Hermite interpolation

- Dictate function values and derivatives
- Hermite polynomials

$$q^{(k)} = c_k(x-x_1)^2\cdots(x-x_{k-1})^2\cdot(x-x_k)\cdot(x-x_{k+1})^2\cdots(x-x_n)^2$$

where $c_k$ is chosen so that $q^{(k)'}(x_k) = 1$.

# Hermite interpolation

- Dictate function values and derivatives
- Hermite polynomials

$$q^{(k)} = c_k(x-x_1)^2 \cdots (x-x_{k-1})^2 \cdot (x-x_k) \cdot (x-x_{k+1})^2 \cdots (x-x_n)^2$$

  where $c_k$ is chosen so that $q^{(k)\prime}(x_k) = 1$.
- Combination of $p^{(k)}$ and $q^{(k)}$ polynomials: match values and derivatives

# Approximation

# Distance, convergence

- Approximate known curve
- cheaper computation, use uniform computations
- distance?

# Distance, convergence

- Approximate known curve
- cheaper computation, use uniform computations
- distance?
- Family of approximating curves $f_n$
- Does $|f - f_n| \to 0$?

# Convergence

- Pointwise convergence

$$\forall_{x\in I,\epsilon}\exists_N\forall_{n\geq N} : |f_n(x) - f(x)| \leq \epsilon.$$

# Convergence

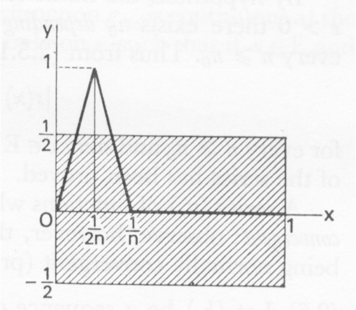
- Pointwise convergence

$$\forall_{x\in I,\epsilon}\exists_N\forall_{n\geq N} : |f_n(x) - f(x)| \leq \epsilon.$$

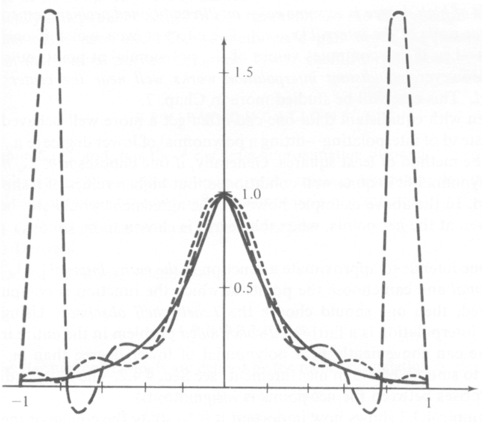
- Uniform convergence

$$\forall_{\epsilon}\exists_N\forall_{x\in I,n\geq N} : |f_n(x) - f(x)| \leq \epsilon.$$

# non-uniform convergence

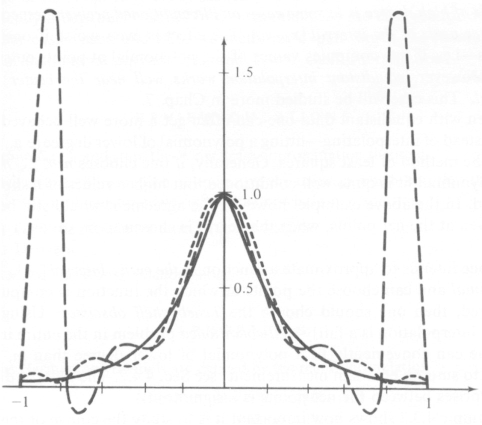


# Problems with high degree polynomials




- Better location of points
- Better polynomials

# Problems with high degree polynomials




- Better location of points
- Better polynomials
- Not use high degree polynomial

# Computations

# Lagrange polynomials

- $n$ terms of $n$ multiplications each:

$$p(x) = \sum_i f_i p^{(i)}(x), \qquad p^{(i)}(x) = \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}$$

- quadratic cost

# Lagrange polynomials

- $n$ terms of $n$ multiplications each:

$$p(x) = \sum_i f_i p^{(i)}(x), \qquad p^{(i)}(x) = \prod_{j\neq i} \frac{x - x_j}{x_i - x_j}$$

- quadratic cost
- Down to linear:

$$p(x) = \prod_i (x - t_i) \cdot \sum_i \frac{y_i}{x - t_i}, \qquad y_i = f_i / \prod_{j\neq i} (x_i - x_j),$$

with $y_i$ precomputed

# Lagrange polynomials

- $n$ terms of $n$ multiplications each:

$$p(x) = \sum_i f_i p^{(i)}(x), \qquad p^{(i)}(x) = \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}$$

- quadratic cost
- Down to linear:

$$p(x) = \prod_i (x - t_i) \cdot \sum_i \frac{y_i}{x - t_i}, \qquad y_i = f_i / \prod_{j \neq i} (x_i - x_j),$$

  with $y_i$ precomputed
- complicated story with derivatives

# Divided differences

# Definition

- $n$-th divided difference $[\tau_1, \ldots, \tau_{n+1}]g$ is leading coefficient of $n+1$-st order ($\Pi_{<n+1}$, degree $n$ or lower) polynomial that agrees with $g$ in $\tau_1, \ldots, \tau_{n+1}$

# Definition

- $n$-th divided difference $[\tau_1, \ldots, \tau_{n+1}]g$ is leading coefficient of $n+1$-st order ($\Pi_{<n+1}$, degree $n$ or lower) polynomial that agrees with $g$ in $\tau_1, \ldots, \tau_{n+1}$
- Zeroeth divided difference: constant polynomial, match $g(\tau_1) = g_1$: $[\tau_1]g = g_1$

# Definition

- $n$-th divided difference $[\tau_1, \ldots, \tau_{n+1}]g$ is leading coefficient of $n+1$-st order ($\Pi_{<n+1}$, degree $n$ or lower) polynomial that agrees with $g$ in $\tau_1, \ldots, \tau_{n+1}$
- Zeroeth divided difference: constant polynomial, match $g(\tau_1) = g_1$: $[\tau_1]g = g_1$
- First divided difference: linear function, leading coefficient is slope
$$[\tau_1, \tau_2]g = \frac{g(\tau_2) - g(\tau_1)}{\tau_2 - \tau_1} = \frac{[\tau_2]g - [\tau_1]g}{\tau_2 - \tau_1}$$
- recurrence?

# Relation

- Let $p_{k+1} \in \prod_{<k+1}$ agree with $g$ in $\tau_1 \ldots \tau_{k+1}$, and $p_k \in \prod_{<k}$ with $g$ in $\tau_1 \ldots \tau_k$, then

$$p_{k+1}(x) - p_k(x) = [\tau_1, \ldots, \tau_{k+1}]g \prod_{i=1}^{k} (x - \tau_i). \qquad (2)$$

- $p_{k+1} - p_k$ is zero in $t_i$ for $i \leq k$:

$$p_{k+1} - p_k = C \prod_{i=1}^{k} (x - \tau_i).$$

- therefore $C = [\tau_1, \ldots, \tau_{k+1}]g$.

# Relation

- Let $p_{k+1} \in \prod_{<k+1}$ agree with $g$ in $\tau_1 \ldots \tau_{k+1}$, and $p_k \in \prod_{<k}$ with $g$ in $\tau_1 \ldots \tau_k$, then

$$p_{k+1}(x) - p_k(x) = [\tau_1, \ldots, \tau_{k+1}]g \prod_{i=1}^{k}(x - \tau_i). \qquad (2)$$

- Proof. $p_k$ is of a lower order, so

$$p_{k+1} - p_k = [\tau_1, \ldots, \tau_{k+1}]gx^k + cx^{k-1} + \cdots .$$

- $p_{k+1} - p_k$ is zero in $t_i$ for $i \leq k$:

$$p_{k+1} - p_k = C \prod_{i=1}^{k}(x - \tau_i).$$

- therefore $C = [\tau_1, \ldots, \tau_{k+1}]g$.

- Repeat this:

$$p_{k+1}(x) = \sum_{m=1}^{k+1} [\tau_1, \ldots, \tau_m]g \prod_{i=1}^{m-1} (x - \tau_i), \qquad (3)$$

- can be evaluated as

$$\begin{aligned} p_{k+1}(x) &= [\tau_1, \ldots, \tau_{k+1}]g \prod^{k} (x - \tau_i) + [\tau_1, \ldots, \tau_k]g \prod^{k-1} (x - \tau_i) \\ &= [\tau_1, \ldots, \tau_{k+1}]g(x - \tau_k)\bigl(c_k + [\tau_1, \ldots, \tau_k]g(x - \tau_{k-1})\bigl(c_{k-1} + \cdots \end{aligned}$$

where $c_k = [\tau_1, \ldots, \tau_k]g / [\tau_1, \ldots, \tau_{k+1}]g$

- Horner's rule

# Construction of divided differences

- Recursive:

$$[\tau_1,\ldots,\tau_{n+1}]g = ([\tau_1,\ldots,\tau_n]g - [\tau_2,\ldots,\tau_{n+1}]g)\,/(\tau_1 - \tau_{n+1}).$$

- Proof. Three polynomials given:
  $p_n^{(1)} \in \prod_{<n}$ agrees with $g$ on $\tau_1 \ldots \tau_n$;
  $p_n^{(2)} \in \prod_{<n}$ agrees with $g$ on $\tau_2 \ldots \tau_{n+1}$;
  $p_{n-1} \in \prod_{<n-1}$ agrees with $g$ on $\tau_2 \ldots \tau_n$.
- then

$$\begin{array}{rcl} p_n^{(1)} - p_{n-1} & = & [\tau_1,\ldots,\tau_n]g \prod_{j=2}^{n}(x-\tau_j) \\ p_n^{(2)} - p_{n-1} & = & [\tau_2,\ldots,\tau_{n+1}]g \prod_{j=2}^{n}(x-\tau_j) \end{array}$$

- $p_{n+1}$ agrees with $g$ on $\tau_1 \ldots \tau_{n+1}$:

$$\begin{aligned} p_{n+1} - p^{(1)} &= [\tau_1, \ldots, \tau_{n+1}]g \prod_{j=1}^{n} (x - \tau_j) \\ p_{n+1} - p^{(2)} &= [\tau_1, \ldots, \tau_{n+1}]g \prod_{j=2}^{n+1} (x - \tau_j) \end{aligned}$$

- Subtracting gives for $p_n^{(1)} - p_n^{(2)}$:

$$([\tau_1, \ldots, \tau_n]g - [\tau_2, \ldots, \tau_{n+1}]g) \prod_{j=2}^{n} (x - \tau_j) = [\tau_1, \ldots, \tau_{n+1}]g \left( \prod_{j=2}^{n+1} - \prod_{j=1}^{n} \right) (x - \tau_j)$$

- Fill in $\tau_2 \ldots \tau_n$: zero
  With $x = \tau_1$

$$([\tau_1, \ldots, \tau_n]g - [\tau_2, \ldots, \tau_{n+1}]g) \prod_{j=2}^{n} (\tau_1 - \tau_j) = [\tau_1, \ldots, \tau_{n+1}]g \prod_{j=2}^{n+1} (\tau_1 - \tau_j)$$

# Parametric curves

# Functions are not enough

- no unique mapping $x \mapsto y$: circle
- implicit $f(x, y) = 0$ trouble with half circle
- parametric:

$$P = P(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}.$$

- parametric interpolation: $P = tP_2 + (1 - t)P_1$: $P(0) = P_1$, $P(1) = P_2$;

$$P(t) = (\cos 2\pi t, \sin 2\pi t)$$

# Cubics

- Flexible enough: 4 degrees of freedom location and direction in two points (Hermite strategy)
- Higher degrees harder to control

# Formal description

- Matrix/Vector description: $Q(t) = C \cdot T$
- coefficient matrix

$$C = \begin{pmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \end{pmatrix}, \qquad T = \begin{pmatrix} 1 \\ t \\ t^2 \\ t^3 \end{pmatrix}$$

- Direction:

$$\frac{dQ(t)}{dt} = C \cdot \frac{dT}{dt} = C \cdot \begin{pmatrix} 0 \\ 1 \\ 2t \\ 3t^2 \end{pmatrix}$$

- Hermite example: $P_1 = Q(0)$, $R_1 = Q'(0)$, $P_2 = Q(1)$, and $R_2 = Q'(1)$

$$C \cdot \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 1 & 3 \end{pmatrix} = [P_1, R_1, P_2, R_2],$$

# Blending functions

- Description $C = G \cdot M$ in terms of basis polynomials and geometry matrix:

$$Q(t) = G{\cdot}M{\cdot}T = \begin{pmatrix} g_{11} & g_{12} & g_{13} & g_{14} \\ g_{21} & g_{22} & g_{23} & g_{24} \\ g_{31} & g_{32} & g_{33} & g_{34} \end{pmatrix} \cdot \begin{pmatrix} m_{11} & \ldots & m_{14} \\ \vdots & & \vdots \\ m_{41} & \ldots & m_{44} \end{pmatrix} {\cdot}T \tag{4}$$

If we introduce new basis polynomials $\pi_i(t) = M_{i*} \cdot T$, then we see that $Q_x = G_{11}\pi_1 + G_{12}\pi_2 + G_{13}\pi_3 + G_{14}\pi_4$, $Q_y = G_{21}\pi_1 + \cdots$, et cetera.

# Computing the basis matrix $M$

Hermite case: geometric constraints $[Q(0), Q'(0), Q(1), Q'(1)]$
From $Q = G \cdot M \cdot T$:

$$Q(t) = G_H \cdot M_H \cdot T(t), \qquad Q'(t) = G_H \cdot M_H \cdot T'(t).$$

Applying to $t = 0$ and $t = 1$:

$$Q_H \equiv [Q(0), Q'(0), Q(1), Q'(1)] = G_H \cdot M_H \cdot T_H$$

where

$$T_H = [T(0), T'(0), T(1), T'(1)] = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 1 & 3 \end{pmatrix}$$

But now $G_H = Q_H$. It now follows that

$$M_H = T_H^{-1} = \begin{pmatrix} 1 & 0 & -3 & 2 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 3 & -2 \\ 0 & 0 & -1 & 1 \end{pmatrix}$$

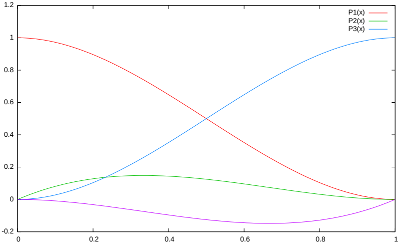
- $Q = G \cdot M \cdot T$ with

$$M_H = T_H^{-1} = \begin{pmatrix} 1 & 0 & -3 & 2 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 3 & -2 \\ 0 & 0 & -1 & 1 \end{pmatrix}.$$

- Writing this out, we find the cubic Hermite polynomials

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

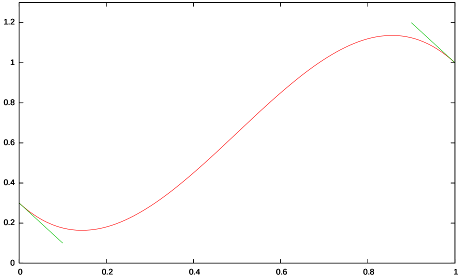
# Hermite basis functions




```
#
# 4 cubic Hermite polynomials
#
set terminal pdf
set xrange [0:1]
set yrange [-.2:1.2]
P1(x) = 2*x**3-3*x**2+1
P2(x) = x**3-2*x**2+x
P3(x) = -2*x**3+3*x**2
P4(x) = x**3-x**2
plot P1(x), P2(x), P3(x), P4(x) title
```

# Example of Hermite interpolation

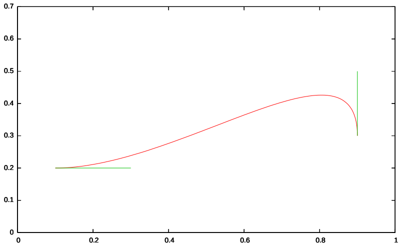
Curve $.3P_1 - 2P_2 + P_3 - 2P4$:
through $(0, .3)$ and $(1, 1)$, with slope $-2$ in both $x = 0, 1$.



# Hermite parametric curves

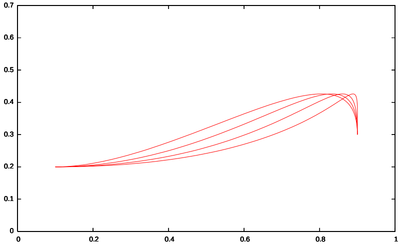
Both coordinates parametric:

$$Q = \begin{pmatrix} .1 \\ .2 \end{pmatrix} P_1 + \begin{pmatrix} 1 \\ 0 \end{pmatrix} P_2 + \begin{pmatrix} .9 \\ .3 \end{pmatrix} P_3 + \begin{pmatrix} 0 \\ -1 \end{pmatrix} P_4.$$




```
#
# Parametric Hermite curve
#
set terminal pdf
set parametric
set multiplot
set xrange [0:1]
set yrange [0:.7]
P1(t) = 2*t**3-3*t**2+1
P2(t) = t**3-2*t**2+t
P3(t) = -2*t**3+3*t**2
P4(t) = t**3-t**2
p1x  = .1 ; p1y  = .2
p1dx =  1 ; p1dy =  0
```

# Smoothness

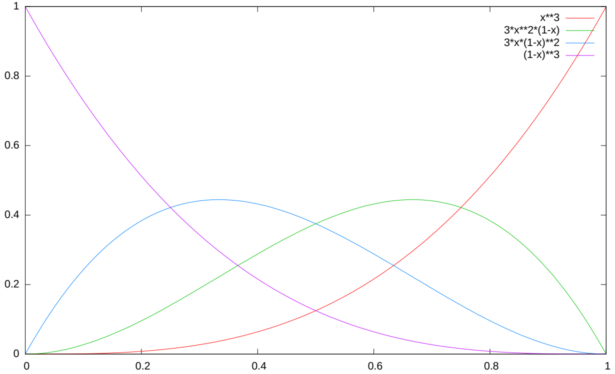



```
set terminal pdf
set parametric
set multiplot
set dummy t
set xrange [0:1]
set yrange [0:.7]
P1(t) = 2*t**3-3*t**2+1
P2(t) = t**3-2*t**2+t
P3(t) = -2*t**3+3*t**2
P4(t) = t**3-t**2
p1x = .1  ; p1y = .2
p2x  = .9 ; p2y = .3
p2dx = 0  ; p2dy = -1
# direction 1:
p1dx = 1 ; p1dy = 0
plot [t=0:1] \
  p1x*P1(t)+p1dx*P2(t)+p2x*P3(t)+p2dx
  p1y*P1(t)+p1dy*P2(t)+p2y*P3(t)+p2dy
  title ""
```

# Splines

# Bernstein polynomials

- $z(t) = (1-t)^3 z_1 + 3(1-t)^2 t z_2 + 3(1-t)t^2 z_3 + t^3 z_4$



# Derivation from Hermite basis

- Hermite direction vectors $R_1, R_2$,
  replace by control points $P'_1, P'_2$:

$$R_1 = 3(P'_1 - P1), \quad R_2 = 3(P_2 - P'_2)$$

- Geometry matrix

$$G_B = [P_1, P'_1, P'_2, P_2] \quad G_H = [P_1, R_1, P_2, R_2] = [P_1, P'_1, P'_2, P_2]M_{BH}$$

with

$$M_{BH} = \begin{pmatrix} 1 & -3 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & -3 \\ 0 & 0 & 1 & 3 \end{pmatrix}$$

- Define

$$M_B = M_{BH} \cdot M_H = \begin{pmatrix} 1 & -3 & 3 & -1 \\ 0 & 3 & -6 & 3 \\ 0 & 0 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

  the Bezier curves:

$$Q(t) = G_H \cdot M_H \cdot T(t) = G_B \cdot M_{BH} \cdot M_H \cdot T(t) = G_B \cdot M_B \cdot T(t)$$

- Per component: $Q_x(t) = g_{11}B_1(t) + g_{12}B_2(t) + \cdots$ where

$$\begin{aligned} B_1(t) &= 1 - 3t + 3t^2 - t^3 &&= (1-t)^3 \\ B_2(t) &= 3t - 6t^2 + 3t^3 &&= 3t(1-t)^2 \\ B_3(t) &= 3t^2 - 3t^3 &&= 3t^2(1-t) \\ B_4(t) &= &&= t^3 \end{aligned}$$

# Enclosing hull

- Bernshtein polynomials

$$z(t) = (1-t)^3 z_1 + 3(1-t)^2 t z_2 + 3(1-t)t^2 z_3 + t^3 z_4$$

- With all $z_i \equiv 1$:

$$z(t) = (t + (1-t))^3 \equiv 1$$

- $\Rightarrow$ Bezier curve components are weighted averages
- $\Rightarrow$ curve containes in convex hull of control points

# Calculation of Bezier curves

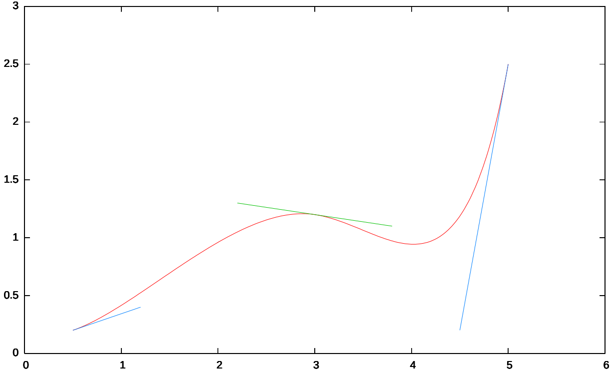
- The simple way: $Q(t) = G \cdot M \cdot T(t)$
  - 2 multiplications to form the terms $t^2$ and $t^3$ in $T$;
  - 16 multiplications and 12 additions forming $M \cdot T$;
  - 12 multiplications and 9 additions forming $G \cdot (M \cdot T)$.
- Improvement: store $\tilde{M} = G \cdot M$:
  - two multiplications for $T$;
  - 12 multiplications and 9 additions for forming $\tilde{M} \cdot T$.

# Calculation with divided differences

- $Q(t) = G \cdot M \cdot T(t)$, $x$ component:
$$x(t) = \sum_k c_k t^{k-1} \qquad c_k = \sum_j G_{1j} M_{jk}.$$
- recall
$$M_B = \begin{pmatrix} 1 & -3 & 3 & -1 \\ 0 & 3 & -6 & 3 \\ 0 & 0 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$
- writing $g_j \equiv G_{1j}$:
$$c_1 = g_1, \quad c_2 = 3(g_2 - g_1), \quad c_3 = 3(g_3 - 2g_2 + g_1), \quad c_4 = g_4 - 3g_3 + 3g_2 - g_1.$$
- These are divided differences:
$$\begin{array}{lcl} [2,1]g & = & g_2 - g_1, \\ [3,2,1]g & = & [3,2]g - [2,1]g = (g_3 - g_2) - (g_2 - g_1) \\ & = & g_3 - 2g_2 + g_1 \\ [4,3,2,1] & = & [4,3,2]g - [3,2,1]g = (g_4 - 2g_3 + g_2) - (g_3 - 2g_2 + g_1) \\ & = & g_4 - 3g_3 + 3g_2 - g_1 \end{array}$$

# Practical use of splines

# Piecewise curves



# Spline drawing

- Divided difference algorithm is good for single points

## Spline drawing

- Divided difference algorithm is good for single points
- Whole curve: compute step by step $Q(n\delta)$, $n = 0, 1, \ldots$

# Spline drawing

- Divided difference algorithm is good for single points
- Whole curve: compute step by step $Q(n\delta)$, $n = 0, 1, \ldots$
- Other tricks: recursive bisection; use line drawing when curve is flat enough
- (flatness test through control points for Bezier)