Global Illumination
Curves

Wolfgang Heidrich

Course News

Assignment 3 (project)

- Due today!!
- Demos in labs starting this Friday
- Demos are MANDATORY(!)

Reading

- Chapter 10 (ray tracing), except 10.8-10.10
- Chapter 14 (global illumination)

Direct Illumination

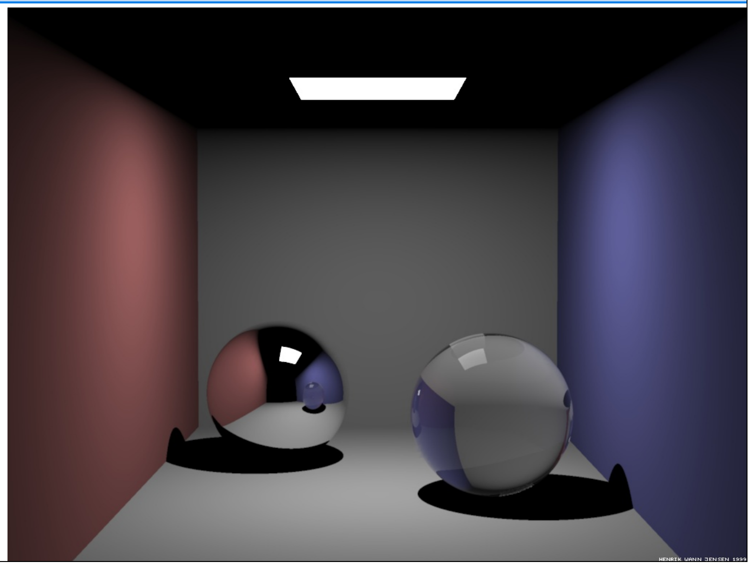


Image by
Henrik Wann Jensen

Global Illumination

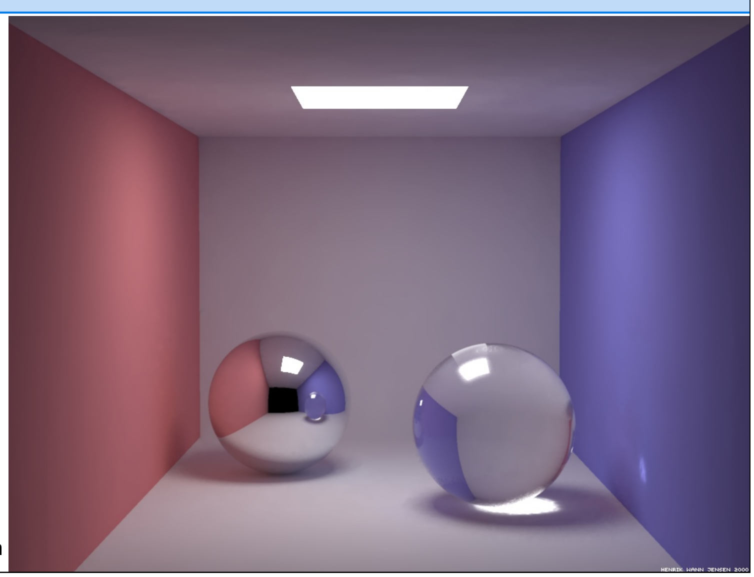


Image by
Henrik Wann Jensen

Rendering Equation

Equation guiding global illumination:

$$L_o(x,\omega_o) = L_e(x,\omega_o) + \int_\Omega \rho(x,\omega_i,\omega_0) L_i(\omega_i) d\omega_i$$

$$= L_e(x,\omega_o) + \int_\Omega \rho(x,\omega_i,\omega_0) L_o(R(x,\omega_i),-\omega_i) d\omega_i$$

Where

- ρ is the reflectance from ω_i to ω_o at point x
- L_o is the outgoing (I.e. reflected) **radiance** at point x in direction ω_i
  - *Radiance is a specific physical quantity describing the amount of light along a ray*
  - *Radiance is constant along a ray*
- L_e is the emitted radiance (=0 unless point x is on a light source)
- R is the "ray-tracing function". It describes what point is visible from x in direction ω_i

Wolfgang Heidrich

Rendering Equation

Equation guiding global illumination:

$$L_o(x,\omega_o) = L_e(x,\omega_o) + \int_\Omega \rho(x,\omega_i,\omega_0) L_i(\omega_i) d\omega_i$$

$$= L_e(x,\omega_o) + \int_\Omega \rho(x,\omega_i,\omega_0) L_o(R(x,\omega_i),-\omega_i) d\omega_i$$

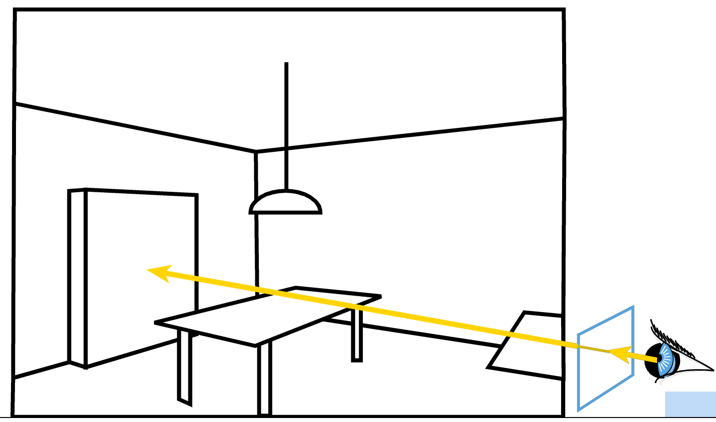
Note:

- The rendering equation is an **integral equation**
- This equation cannot be solved directly
  - *Ray-tracing function is complicated!*
  - *Similar to the problem we had computing illumination from area light sources!*

Wolfgang Heidrich

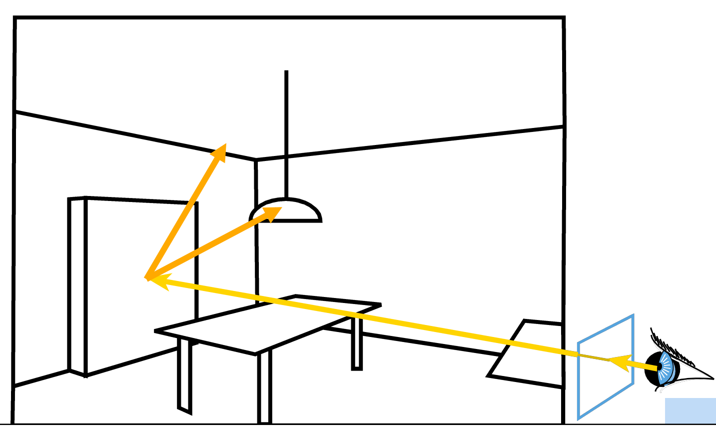
Ray Casting

- Cast a ray from the eye through each pixel
- The following few slides are from Fred Durand (MIT)



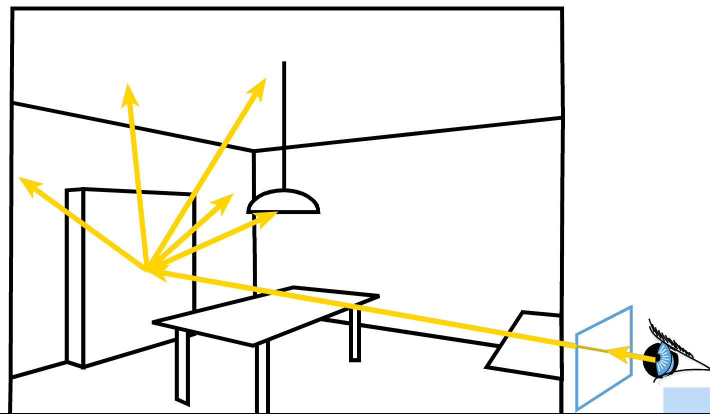
Ray Tracing

- Cast a ray from the eye through each pixel
- Trace secondary rays (light, reflection, refraction)



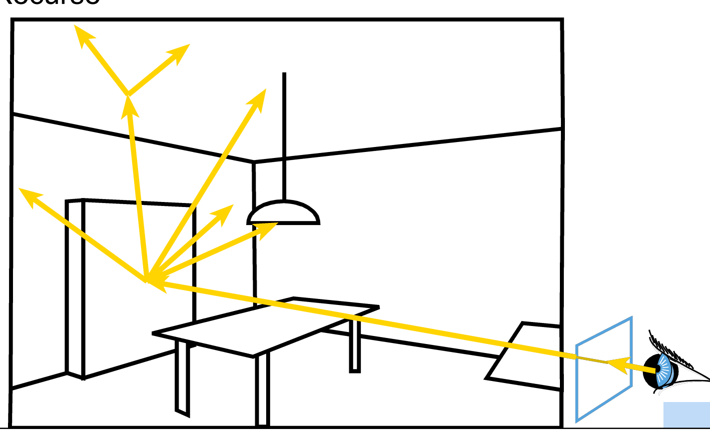
Monte Carlo Ray Tracing

- Cast a ray from the eye through each pixel
- Cast random rays from the visible point
  - *Accumulate radiance contribution*



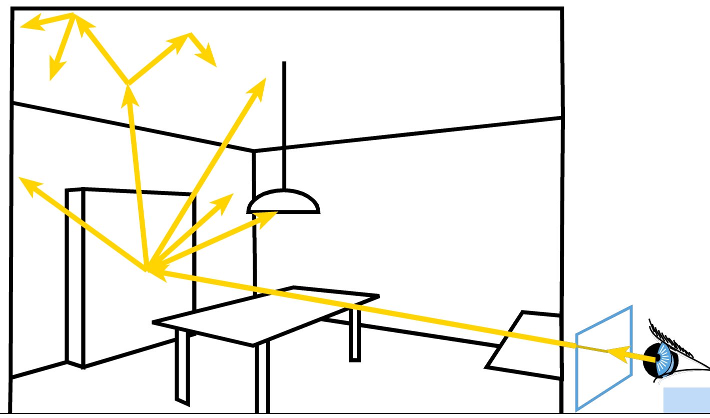
Monte Carlo Ray Tracing

- Cast a ray from the eye through each pixel
- Cast random rays from the visible point
- Recurse



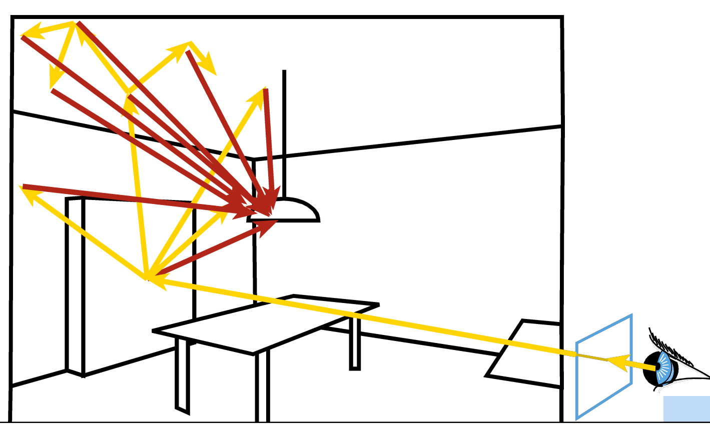
Monte Carlo

- Cast a ray from the eye through each pixel
- Cast random rays from the visible point
- Recurse



Monte Carlo

- Systematically sample primary light



Monte Carlo Path Tracing

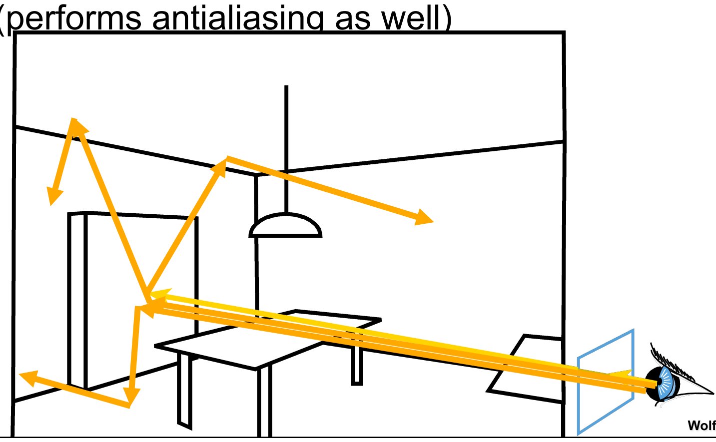
In practice:

- Do not branch at every intersection point
  - *This would have exponential complexity in the ray depth!*
- Instead:
  - *Shoot some number of primary rays through the pixel (10s-1000s, depending on scene!)*
  - *For each pixel and each intersection point, make a **single**, **random** decision in which direction to go next*

Wolfgang Heidrich

Monte Carlo Path Tracing

- Trace only one secondary ray per recursion
- But send many primary rays per pixel
- (performs antialiasing as well)



Wolfgang Heidrich

How to Sample?

Simple sampling strategy:

- At every point, choose between all possible reflection directions with equal probability
- This will produce very high variance/noise if the materials are specular or glossy
- Lots of rays are required to reduce noise!

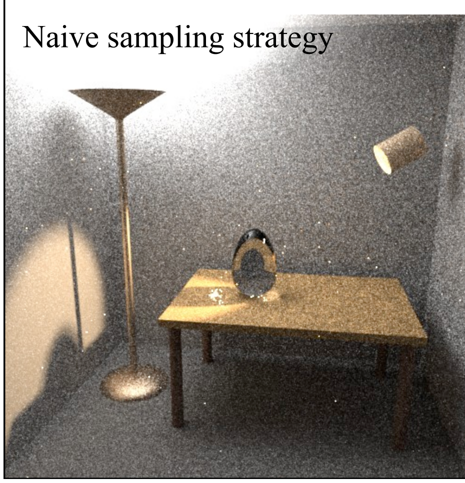
Better strategy: importance sampling

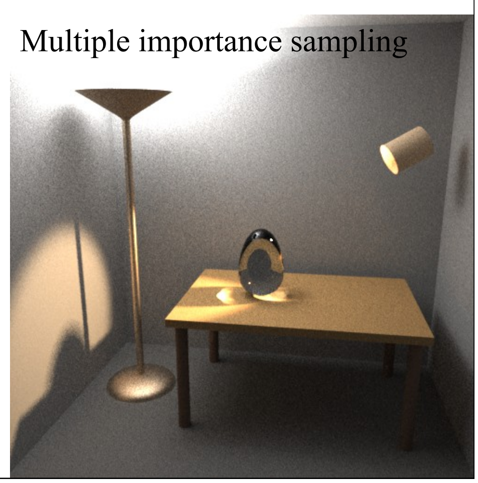
- Focus rays in areas where most of the reflected light contribution will be found
- For example: if the surface is a mirror, then only light from the mirror direction will contribute!
- Glossy materials: prefer rays near the mirror direction

Wolfgang Heidrich

How to Sample?

- Images by Veach & Guibas





How to Sample?

Sampling strategies are still an active research area!

- Recent years have seen drastic advances in performance
- Lots of excellent sampling strategies have been developed in statistics and machine learning
  - *Many are useful for graphics*

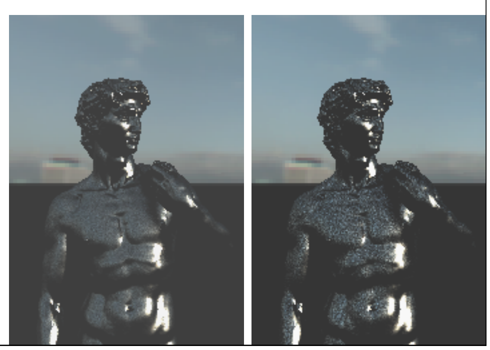
Wolfgang Heidrich

How to Sample?

Objective:

- Compute light transport in scenes using stochastic ray tracing
  - *Monte Carlo, Sequential Monte Carlo*
  - *Metropolis*

[Burke, Ghosh, Heidrich '05]
[Ghosh, Heidrich '06],
[Ghosh, Doucet, Heidrich '06]



How to Sample?

- E.g: importance sampling (left) vs. Sequential Monte Carlo (right)

More on Global Illumination

This was a (very) quick overview

- More details in CPSC 514 (Computer Graphics: Rendering)
- Not offered this year, but in 20011/12

Curves

Wolfgang Heidrich

Motivation

Geometric representations so far:

- Discrete geometry
  - *Triangles, line segments*
  - *Rendering pipeline, ray-tracing*
- Specific objects
  - *Spheres*
  - *Ray-tracing*

Want more general representations:

- Flexible like triangles
- But smooth!

Curves&Surfaces as Parametric Functions

Curves&surfaces in arbitrary dimensions

- Curves: $\mathbf{x} = F(t); F : \mathrm{R} \mapsto \mathrm{R}^d$
- Surfaces: $\mathbf{x} = F(s,t); F : \mathrm{R}^2 \mapsto \mathrm{R}^d$

In practice:

- Restrict to specific class of functions
  - *e.g. polynomials of certain degree*

$$\mathbf{x} = \sum_{i=0}^{m} \mathbf{b}_i t^i$$

In 2D: $$\begin{pmatrix} x \\ y \end{pmatrix} = \sum_{i=0}^{m} \begin{pmatrix} b_{x,i} \\ b_{y,i} \end{pmatrix} t^i$$

Wolfgang Heidrich

Polynomial Curves

Advantages:

- Computationally easy to handle
  - $\mathbf{b}_0 \ldots \mathbf{b}_m$ *uniquely describe curve (finite storage, easy to represent)*

Disadvantages:

- Not all shapes representable
  - *Partially fix with piecewise functions (splines)*
- Still not very intuitive
  - *Fix: represent polynomials in different basis*
  - *For example: Bernstein polynomials*
  - *This is what is called a Bézier curve*

Wolfgang Heidrich

Polynomial Bases

Reminder

- The set of all polynomials of degree ≤ m over R forms a vector space with the common polynomial operations
  - *What are those operations?*
  - *Dimension of this space is m+1*
- One common basis for this space are the monomials

$$\{1, t, t^2, \ldots, t^m\}$$

- Problem: the relationship between this basis and a geometric shape is quite unintuitive
- Thus: use another basis later!

Wolfgang Heidrich

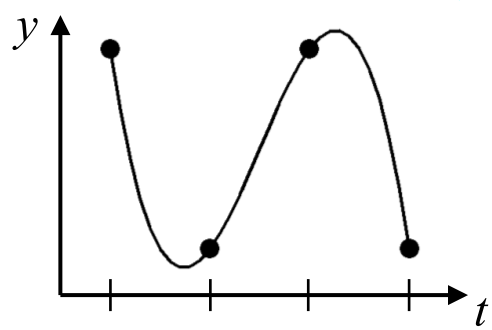
Interpolation

Find a polynomial y(t) such that $y(t_i)=y_i$

- *For 4 points t_i: need cubic polynomial*

$$y(t) = c_0 + c_1 t + c_2 t^2 + c_3 t^3$$

$$\underbrace{\begin{pmatrix} 1 & t & t^2 & t^3 \end{pmatrix}}_{\textbf{basis}} \underbrace{\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}}_{\textbf{coefficients}} = y(t)$$

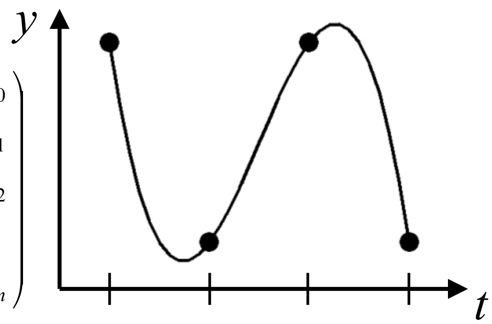


Wolfgang Heidrich

Interpolation

Find a polynomial y(t) such that $y(t_i)=y_i$

$$y(t) = c_0 + c_1 t + c_2 t^2 + c_3 t^3$$



$$\begin{pmatrix} 1 & t_0 & t_0^{\,2} & \dots & t_0^{\,n} \\ 1 & t_1 & t_1^{\,2} & \dots & t_1^{\,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & t_n & t_n^{\,2} & \dots & t_n^{\,n} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

Vandermonde matrix

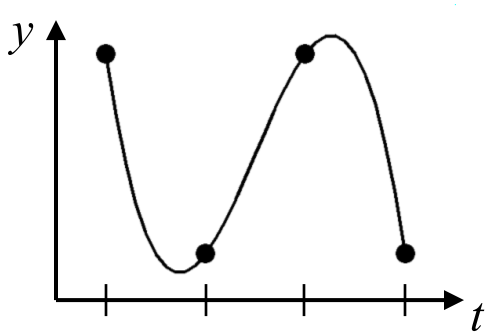
Wolfgang Heidrich

Interpolation

Find a polynomial y(t) such that $y(t_i)=y_i$

Example:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \\ 1 & 3 & 9 & 27 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 3 \\ 1 \\ 3 \\ 1 \end{pmatrix}$$



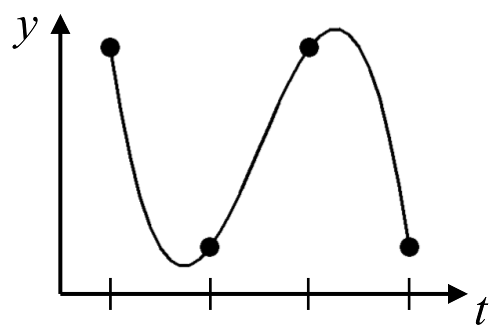
Wolfgang Heidrich

Interpolation

Find a polynomial y(t) such that $y(t_i)=y_i$

Example:

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 3 \\ -20/3 \\ 6 \\ -1/3 \end{pmatrix}$$

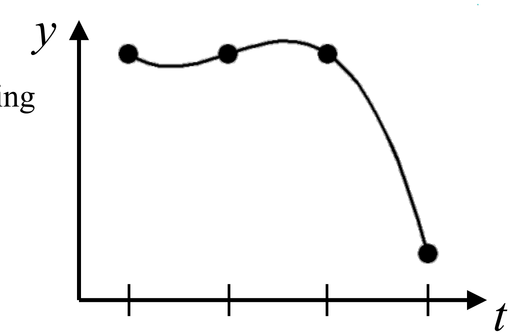


Wolfgang Heidrich

Interpolation

Find a polynomial y(t) such that $y(t_i)=y_i$

Intuitive control of curve using control points!



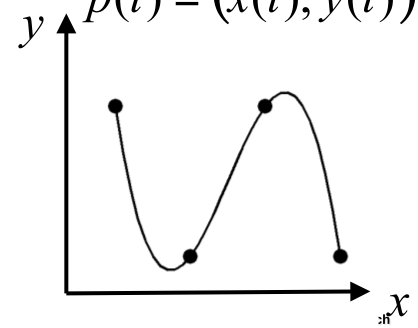
Wolfgang Heidrich

Interpolation

Parametric setting:

- Perform interpolation separately for x, y, (and z in 3D)
- Assign arbitrary parameter values to control points
  - *I.e. choose t_i, such that $f(t_i) = (x_i, y_i)$*
  - *This choice **will** affect the curve shape!*

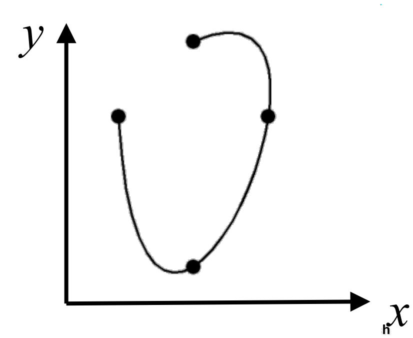
$$p(t) = \big(x(t), y(t)\big)$$



Interpolation

Parametric setting:

- Perform interpolation separately for x, y, (and z in 3D)
- Assign arbitrary parameter values to control points
  - *I.e. choose t_i, such that $f(t_i) = (x_i, y_i)$*
  - *This choice **will** affect the curve shape!*



Generalized Vandermonde Matrices

Assume different basis functions $f_i(t)$

$$y(t) = \sum_i c_i f_i(t)$$

$$\begin{pmatrix} f_0(t_0) & f_1(t_0) & f_2(t_0) & \ldots & f_n(t_0) \\ f_0(t_1) & f_1(t_1) & f_2(t_1) & \ldots & f_n(t_1) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ f_0(t_n) & f_1(t_n) & f_2(t_n) & \ldots & f_n(t_n) \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

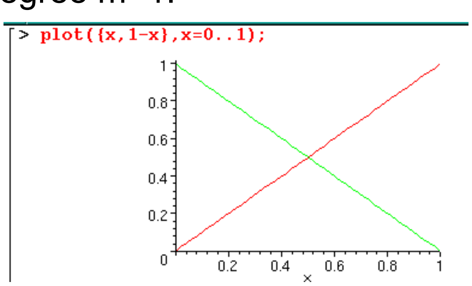
Wolfgang Heidrich

Other Bases for Polynomials

Bernstein Polynomials

$$B_i^m(t) := \binom{m}{i} t^i (1-t)^{m-i}; i = 0..m; t \in [0,1]$$

- Graph for degree m=1:

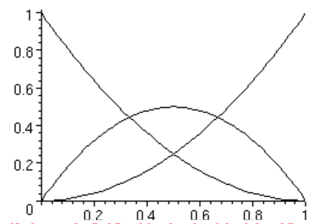


Wolfgang Heidrich

Bernstein Polynomials

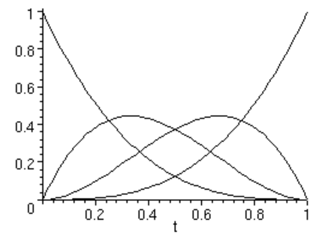
- Graph for m=2:

```
> plot({seq(binomial(2,i)*t^i*(1-t)^(2-i),i=0..2)},
  t=0..1,color=black);
```



- Graph for m=3:

```
> plot({seq(binomial(3,i)*t^i*(1-t)^(3-i),i=0..3)},
  t=0..1,color=black);
```



Bernstein Polynomials

$$B_i^m(t) := \binom{m}{i} t^i (1-t)^{m-i}; i = 0..m; t \in [0,1]$$

Properties:

- $B_i^m(t)$ is a polynomial of degree m
- $B_i^m(t) \geq 0$ for $t \in [0,1]; B_0^m(0) = 1; B_i^m(0) = 0$ for $i \neq 0$
- $B_i^m(t) = B_{m-i}^m(1-t)$
- $B_i^m(t)$ has exactly one maximum in the interval 0..1. It is at t=i/m (proof: compute derivative…)
- W/o proof: all (m+1) functions B_i^m are linearly independent
  - *Thus they form a basis for all polynomials of degree ≤ m*

Wolfgang Heidrich

Bernstein Polynomials

More properties

- $$\sum_{i=0}^{m} B_i^m(t) = (t + (1-t))^m \equiv 1$$

- $$B_i^m(t) = t \cdot B_{i-1}^{m-1}(t) + (1-t) \cdot B_i^{m-1}(t)$$

- Both are quite important a fast evaluation algorithm of Bézier curves (de Casteljau algorithm)

Wolfgang Heidrich

Bézier Curves

Definition:

- A Bézier curve is a polynomial curve that uses the Bernstein polynomials as a basis

$$F(t) = \sum_{i=0}^{m} \mathbf{b}_i B_i^m(t)$$

- The b_i are called control points of the Bézier curve
- The control polygon is obtained by connecting the control points with line segments

Advantage of Bézier curves:

- The control points and control polygon have clear geometric meaning and are intuitive to use

Wolfgang Heidrich

Properties of Bézier Curves (Pierre Bézier, Renault, about 1960)

Easy to see:

- The endpoints b_0 and b_m of the control polygon are interpolated and the corresponding parameter values are t=0 and t=1

More properties:

- The Bézier curve is tangential to the control polygon in the endpoints
- The curve completely lies within the convex hull of the control points
- The curve is *affine invariant*
- There is a fast, recursive evaluation algorithm

Wolfgang Heidrich

Bézier Curve Properties

$$F(t) = \sum_{i=0}^{m} \mathbf{b}_i B_i^m(t)$$

Recall:

- Bernstein polynomials have values between 0 and 1 for $t \in [0,1]$, and

$$\sum_{i=0}^{m} B_i^m(t) \equiv 1$$

  - *Therefore: every point on Bézier curve is convex combination of control points*
  - *Therefore: Bézier curve lies completely within convex hull of control points*

Wolfgang Heidrich

De Casteljau Algorithm

Also recall:

- Recursive formula for Bernstein polynomials:

$$B_i^m(t) = t \cdot B_{i-1}^{m-1}(t) + (1-t) \cdot B_i^{m-1}(t)$$

Plug into Bézier curve definition:

$$F(t) = \sum_{i=0}^{m} \mathbf{b}_i \left(t \cdot B_{i-1}^{m-1}(t) + (1-t) \cdot B_i^{m-1}(t) \right)$$

$$= t \cdot \sum_{i=1}^{m} \mathbf{b}_i B_{i-1}^{m-1}(t) + (1-t) \cdot \sum_{i=0}^{m-1} \mathbf{b}_i B_i^{m-1}(t)$$

Wolfgang Heidrich

De Casteljau Algorithm

Consequence:

- Every point $F(t_0)$ on a Bézier curve of degree m is the convex combination of two points $G(t_0)$ and $H(t_0)$ that lie on Bézier curves of degree m-1.
- The control points of $G(t)$ are the <u>first</u> m control points of $F(t)$
- The control points of $H(t)$ are the <u>last</u> m control points of $F(t)$

Wolfgang Heidrich

De Casteljau Algorithm

Recursion:

- Every point on a Bézier curve can be generated through successive convex combinations of the degree 0 Bézier curves
- Degree 0 Bézier curves are the control points!

$$F(t) = \sum_{i=0}^{0} \mathbf{b}_i B_i^0(t) = \mathbf{b}_i \cdot 1 \equiv \mathbf{b}_i$$

De Casteljau Algorithm

After working out the math we get:

$$F(t) = \mathbf{b}_0^m(t) \text{ ; where}$$

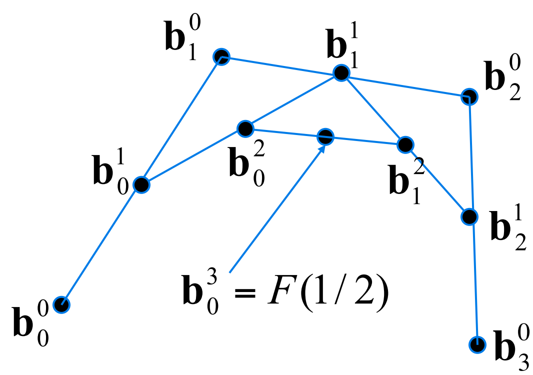
$$\mathbf{b}_i^0(t) := \mathbf{b}_i(t); \quad i = 0 \ldots m$$

$$\mathbf{b}_i^l(t) := (1-t) \cdot \mathbf{b}_i^{l-1}(t) + t \cdot \mathbf{b}_{i+1}^{l-1}(t)$$

De Casteljau Algorithm

Graphical Interpretation:

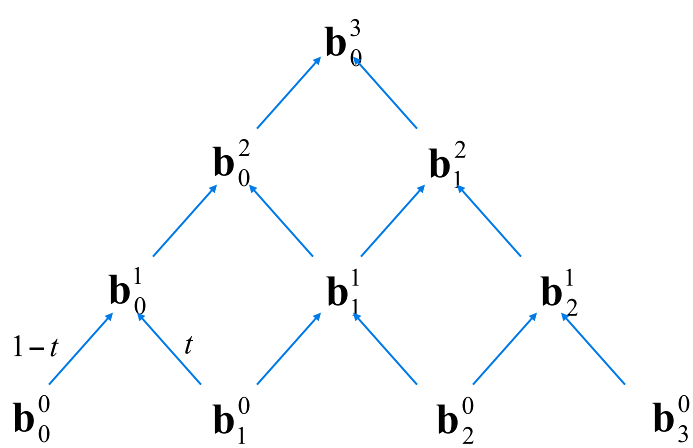
- Determine point $F(1/2)$ for the cubic Bézier curve given by the following four points:



Wolfgang Heidrich

De Casteljau Algorithm

Evaluation scheme (cubic case):



Wolfgang Heidrich

More on Curves & Surfaces

This was a (very) quick overview

- More details in CPSC 424 (Geometric Modeling)
- Offered every other year
- Taught by Alla Sheffer in 2012/13

Wolfgang Heidrich

Coming Up

Monday:

- Examples of recent graphics research

Wednesday:

- Course summary
- Q&A (bring your questions…)

Wolfgang Heidrich