Advanced Computer Graphics Proseminar

Univ.-Prof. Dr. Matthias Harders

Winter semester 2015

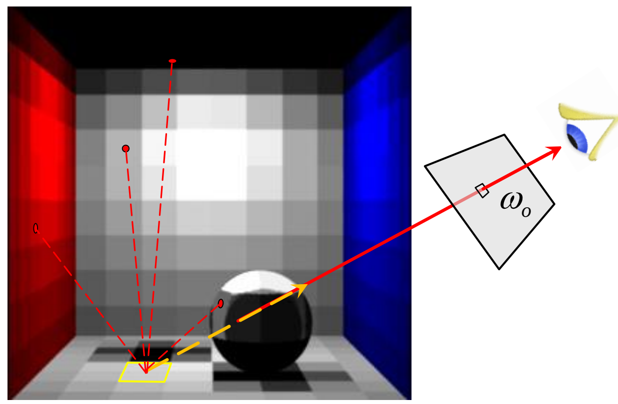


Radiosity

- Assumes purely diffuse surfaces (i.e. same apparent brightness from all viewing directions)
- Initially compute radiosity (i.e. radiant exitance) instead of radiance
- Numerical solution with finite element approach

$$B_i = E_i + \rho_i \sum_j B_j F_{ij}$$

Radiosity

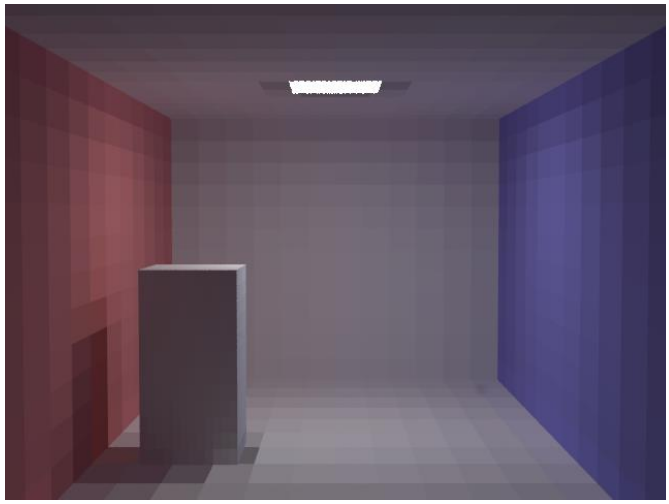


$$B_i = E_i + \rho_i \sum_j B_j F_{ij}$$

Algorithm

1. Discretize geometry into surface patches i
2. Compute form factors for all patch pairs
3. Solve system of linear equations for radiosities
4. Display solution via transformation of radiosities (e.g. tone mapping, gamma correction)

Example Code Rendering



Example Code – Main Features

- Renders Cornell box-type scene
- Geometries defined by rectangles
- All rectangles subdivided into equal number of patches
- Form factors computed for patch pairs via Monte-Carlo integration
- Each patch regularly sampled at $n \times n$ points
- Computes two images – with constant radiosity or bicubic interpolation

Main Structs

- Points, vectors, rays, colors: `Vector`
- Rays (primary and visibility): `Ray`
- Final rendered pixels: `Image`
- Scene geometry element: `Rectangle`

Main Functions

- Intersection of rays with geometry: `Intersect_Scene()`
- Calculate form factors: `Calculate_Form_Factors()`
- Numerical iteration for radiosity: `Calculate_Radiosity()`
- Bicubic interpolation: `bicubicInterpolate()`
- Calculate radiance from radiosity: `Radiance()`
- Setup camera, create and send primary rays: `main()`

Vector Operations

- Normalized vector (divide vector by its length)

$$\hat{\mathbf{v}} = \mathbf{v}/\|\mathbf{v}\| = \left(v_x, v_y, v_z\right)/\sqrt{v_x^2 + v_y^2 + v_z^2}$$

- Dot product (cosine of angle between unit vectors)

$$\mathbf{u} \cdot \mathbf{v} = \left(u_x, u_y, u_z\right) \cdot \left(v_x, v_y, v_z\right)^T = u_x \cdot v_x + u_y \cdot v_y + u_z \cdot v_z$$

$$= \|\mathbf{u}\| \cdot \|\mathbf{v}\| \cdot \cos\theta$$

- Cross product

$$\mathbf{u} \times \mathbf{v} = \left(u_x, u_y, u_z\right) \times \left(v_x, v_y, v_z\right)^T$$

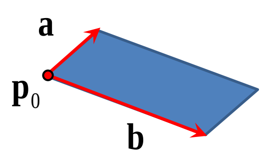
$$= \left(u_y \cdot v_z - u_z \cdot v_y, u_z \cdot v_x - u_x \cdot v_z, u_x \cdot v_y - u_y \cdot v_x\right)$$

Image

- Color values stored in 1D array (note: rows flipped, file format requires rows top to bottom)
- Output file written in *Plain PPM* format (RGB data, ranging from 0 to 255, as raw ASCII)
- RGB radiance data clamped to [0,1] range
- Gamma correction with gamma value of 2.2

Rectangle

- Given by corner point and two edges



Rectangle

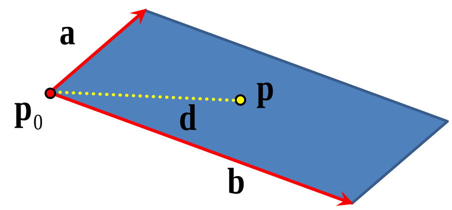
- Ray-rectangle intersection (note: does not check for parallel ray)

$$t = -\frac{(\mathbf{p} - \mathbf{r}) \cdot \mathbf{n}}{\mathbf{d} \cdot \mathbf{n}}$$

```
const double t = (p0 - ray.org).Dot(normal) /
                 ray.dir.Dot(normal);
if (t <= 0.00001)
    return 0.0;

Vector p = ray.org + ray.dir * t;
```

Rectangle

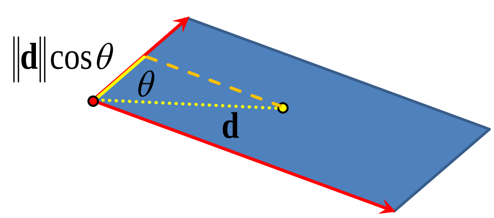
- Check if intersection inside rectangle based on angles to edges




```
Vector d = p - p0;
const double ddota = d.Dot(edge_a);
if (ddota < 0.0 || ddota > edge_a.LengthSquared())
    return 0.0;
const double ddotb = d.Dot(edge_b);
if (ddotb < 0.0 || ddotb > edge_b.LengthSquared())
    return 0.0;
```


Rectangle

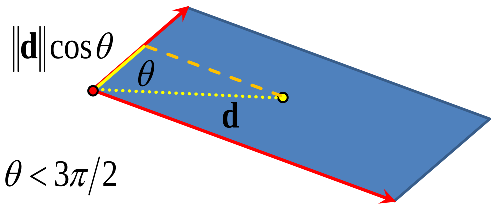
- Check if intersection inside rectangle based on angles to edges




```
Vector d = p - p0;
const double ddota = d.Dot(edge_a);
if (ddota < 0.0 || ddota > edge_a.LengthSquared())
    return 0.0;
const double ddotb = d.Dot(edge_b);
if (ddotb < 0.0 || ddotb > edge_b.LengthSquared())
    return 0.0;
```

Rectangle

- Check if intersection inside rectangle based on angles to edges

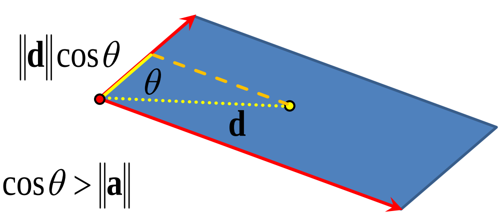


$$\|\mathbf{a}\| \cdot \|\mathbf{d}\|\cos\theta < 0 \Rightarrow \pi/2 < \theta < 3\pi/2$$

```
Vector d = p - p0;
const double ddota = d.Dot(edge_a);
if (ddota < 0.0 || ddota > edge_a.LengthSquared())
    return 0.0;
const double ddotb = d.Dot(edge_b);
if (ddotb < 0.0 || ddotb > edge_b.LengthSquared())
    return 0.0;
```


Rectangle

- Check if intersection inside rectangle based on angles to edges

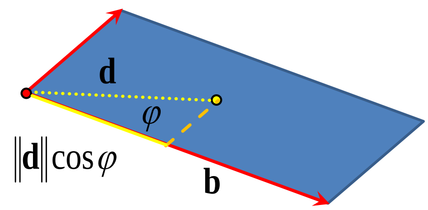


$$\|\mathbf{a}\| \cdot \|\mathbf{d}\|\cos\theta > \|\mathbf{a}\|^2 \Leftrightarrow \|\mathbf{d}\|\cos\theta > \|\mathbf{a}\|$$

```
Vector d = p - p0;
const double ddota = d.Dot(edge_a);
if (ddota < 0.0 || ddota > edge_a.LengthSquared())
    return 0.0;
const double ddotb = d.Dot(edge_b);
if (ddotb < 0.0 || ddotb > edge_b.LengthSquared())
    return 0.0;
```

Rectangle

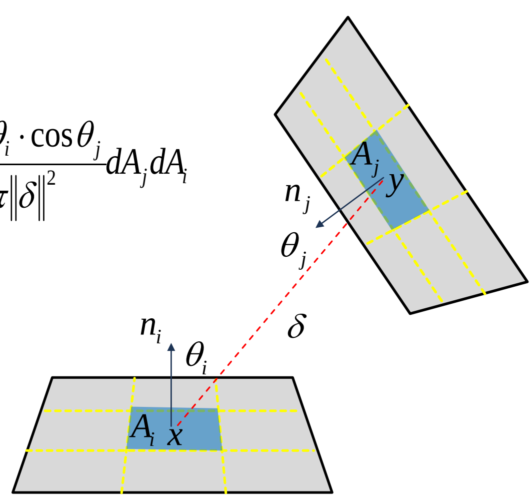
- Check if intersection inside rectangle based on angles to edges




```
Vector d = p - p0;
const double ddota = d.Dot(edge_a);
if (ddota < 0.0 || ddota > edge_a.LengthSquared())
    return 0.0;
const double ddotb = d.Dot(edge_b);
if (ddotb < 0.0 || ddotb > edge_b.LengthSquared())
    return 0.0;
```


Form Factor Calculation

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} V(x,y) \frac{\cos\theta_i \cdot \cos\theta_j}{\pi \|\delta\|^2} dA_j dA_i$$



Monte-Carlo Integration

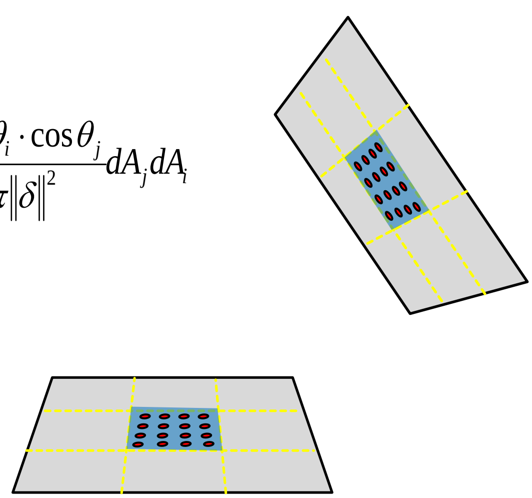
- Numerical approximation of value of definite integral
- Based on random sampling of function
- Random samples averaged and weighted by probabilities of sampling value

$$\iint f(x,y)dxdy \approx \frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i,y_i)}{p(x_i,y_i)}$$

(more details in later lecture)

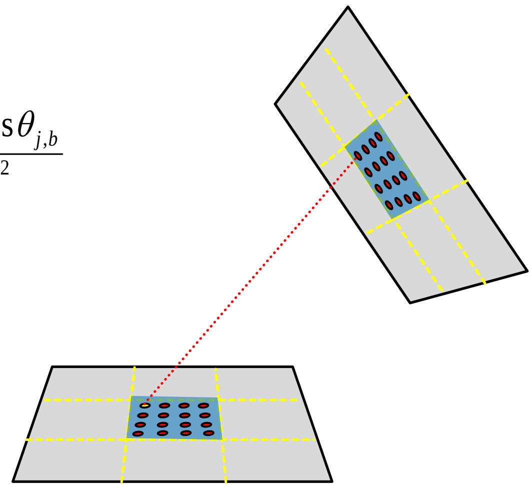
Form Factor Calculation

$$F_{ij} = \frac{1}{A_i}\int_{A_i}\int_{A_j} V(x,y)\frac{\cos\theta_i \cdot \cos\theta_j}{\pi\|\delta\|^2}dA_j dA_i$$



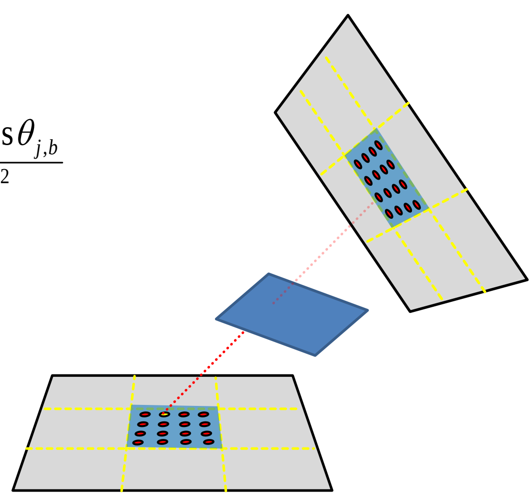
Form Factor Calculation

$$V(x_a, y_b)\frac{\cos\theta_{i,a} \cdot \cos\theta_{j,b}}{\pi\left\|\delta_{a,b}\right\|^2}$$



Form Factor Calculation

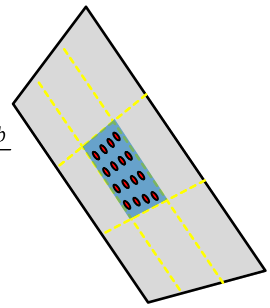
$$V(x_a, y_b)\frac{\cos\theta_{i,a} \cdot \cos\theta_{j,b}}{\pi\left\|\delta_{a,b}\right\|^2}$$

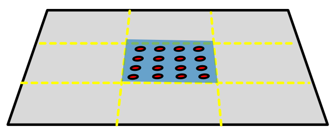


Form Factor Calculation

$$\hat{F}_{ij} \approx \frac{1}{n_a \cdot n_b} \cdot \sum_{a,b} \frac{1}{A_i A_j} V(x_a, y_b) \frac{\cos\theta_{i,a} \cdot \cos\theta_{j,b}}{\pi \left\| \delta_{a,b} \right\|^2}$$

$$F_{ij} \approx \frac{1}{A_i} \hat{F}_{ij} \qquad F_{ji} \approx \frac{1}{A_j} \hat{F}_{ij}$$





Form Factor Calculation

```
if (Intersect_Scene(Ray(xi, ij), &t, &id, &normal) &&
    id != j)
{
    continue;
}

const double d0 = normal_i.Dot(ij);
const double d1 = normal_j.Dot(-1.0 * ij);

if (d0 > 0.0 && d1 > 0.0)
{
    const double K = d0 * d1 /
                 (M_PI * (xj - xi).LengthSquared());
    F += K / pdf;
}
```

Radiosity Calculation

- Iterative numerical Gauss-Seidel solver

$$B^{(k+1)} = E + RFB^{(k)}$$

$$\mathbf{B}_i^{(k+1)} = \mathbf{E}_i + \sum_{j=1}^{i-1} \begin{pmatrix} \rho_{i,R} \\ \rho_{i,G} \\ \rho_{i,B} \end{pmatrix} \otimes F_{ij} \mathbf{B}_j^{(k+1)} + \sum_{j=i+1}^{n} \begin{pmatrix} \rho_{i,R} \\ \rho_{i,G} \\ \rho_{i,B} \end{pmatrix} \otimes F_{ij} \mathbf{B}_j^{(k)}$$

Radiosity Calculation

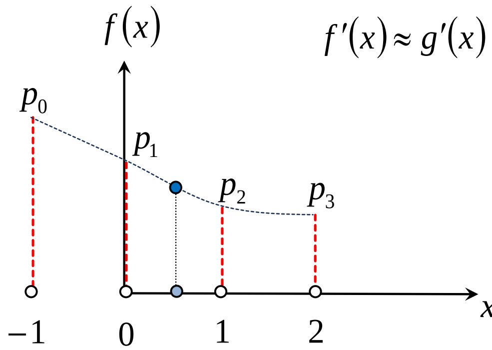
```
for (int j = 0; j < n; j ++)
{
    for (int ja = 0; ja < recs[j].a_num; ja ++)
    {
        for (int jb = 0; jb < recs[j].b_num; jb ++)
        {
            const double Fij = form_factor[patch_i *
                                   patch_num + patch_j];

            if (Fij > 0.0)
                B = B + Fij * recs[j].patch[ja *
                                  recs[j].b_num + jb];
        }
    }
}
B = recs[i].color.MultComponents(B) + recs[i].emission;
recs[i].patch[ia * recs[i].b_num + ib] = B;
```

Radiosity Interpolation

- Cubic interpolation

$$f(x) \approx g(x) = ax^3 + bx^2 + cx + d$$

$$f'(x) \approx g'(x) = 3ax^2 + 2bx + c$$



Radiosity Interpolation

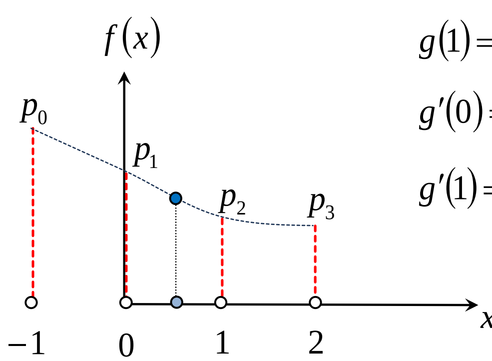
- Cubic interpolation

$$g(0) = d$$

$$g(1) = a + b + c + d$$

$$g'(0) = c$$

$$g'(1) = 3a + 2b + c$$



Radiosity Interpolation

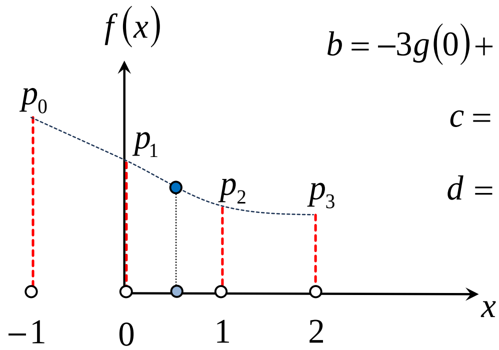
- Cubic interpolation

$$a = 2g(0) - 2g(1) + g'(0) + g'(1)$$

$$b = -3g(0) + 3g(1) - 2g'(0) - g'(1)$$

$$c = g'(0)$$

$$d = g(0)$$



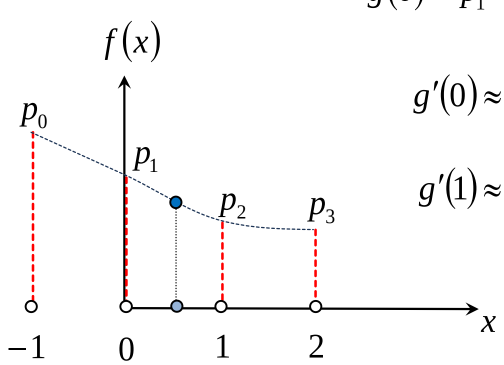
Radiosity Interpolation

- Cubic interpolation

$$g(0) = p_1 \qquad g(1) = p_2$$

$$g'(0) \approx \frac{p_2 - p_0}{2}$$

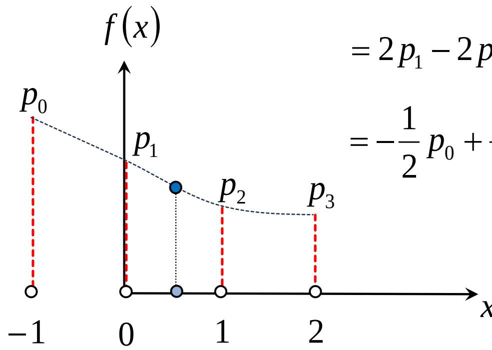
$$g'(1) \approx \frac{p_3 - p_1}{2}$$



Radiosity Interpolation

- Cubic interpolation

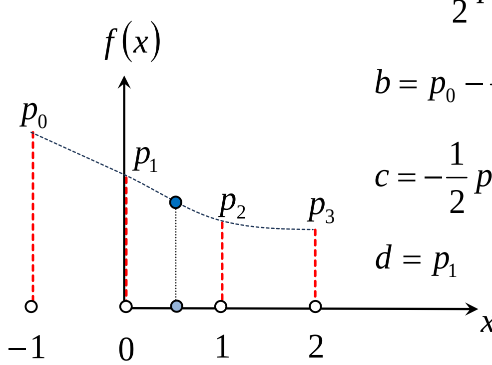
$$a = 2g(0) - 2g(1) + g'(0) + g'(1)$$
$$= 2p_1 - 2p_2 + \frac{p_2 - p_0}{2} + \frac{p_3 - p_1}{2}$$
$$= -\frac{1}{2}p_0 + \frac{3}{2}p_1 - \frac{3}{2}p_2 + \frac{1}{2}p_3$$



Radiosity Interpolation

- Cubic interpolation

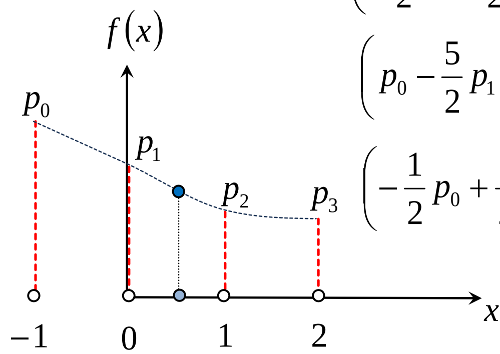
$$a = -\frac{1}{2}p_0 + \frac{3}{2}p_1 - \frac{3}{2}p_2 + \frac{1}{2}p_3$$
$$b = p_0 - \frac{5}{2}p_1 - 2p_2 - \frac{1}{2}p_3$$
$$c = -\frac{1}{2}p_0 + \frac{1}{2}p_2$$
$$d = p_1$$



Radiosity Interpolation

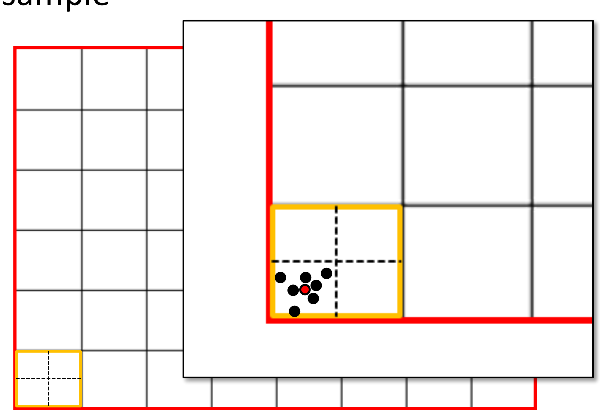
- Cubic interpolation

$$g(x)=\left(-\frac{1}{2}p_0+\frac{3}{2}p_1-\frac{3}{2}p_2+\frac{1}{2}p_3\right)x^3+\left(p_0-\frac{5}{2}p_1-2p_2-\frac{1}{2}p_3\right)x^2+\left(-\frac{1}{2}p_0+\frac{1}{2}p_2\right)x+p_1$$



Screen Pixel Supersampling

- 2×2 subsampling per pixel and n random samples per subsample



Programming Assignment 1

- Change geometric base element from rectangles (& rectangular patches) to triangles (& triangular patches)
- Change scene description to triangle meshes
- Implement ray-triangle intersection test

Proseminar Schedule

Date	Topic	Remark
12.10.	Introduction	
19.10.	Theory – Radiometry	Radiosity example code
26.10.	*(no proseminar - Nationalfeiertag)*	
2.11.	*(no proseminar - Allerseelen)*	
9.11.	Discussion of Radiosity code	Programming assignment 1
16.11.	Programming support and advice	
23.11.	Programming support and advice	Path Tracer example
30.11.	Discussion of Path Tracer code	Programming assignment 2, *Hand-in PA1*
7.12.	Programming support and advice	
14.12.	Programming support and advice	*Project proposal (21.12. Hand-in PA2)*
	Christmas break	
14.1.	Geometric Modelling	
21.1.	Procedural Modelling	
28.1.	Programming support and advice	
4.2.	Project presentation	*Submission final project*