Further Graphics

Global Illumination

Alex Benton, University of Cambridge – alex@bentonian.com

Supported in part by Google UK, Ltd

HENRIK WANN JENSEN - 2002

Anisotropic shading

Anisotropic shading occurs in nature when light reflects off a surface differently in one direction from another, as a function of the surface itself. The specular component is modified by the direction of the light.

http://www.blenderguru.com/videos/introduction-to-anisotropic-shading/

The *rendering equation*

Reflected light

BRDF

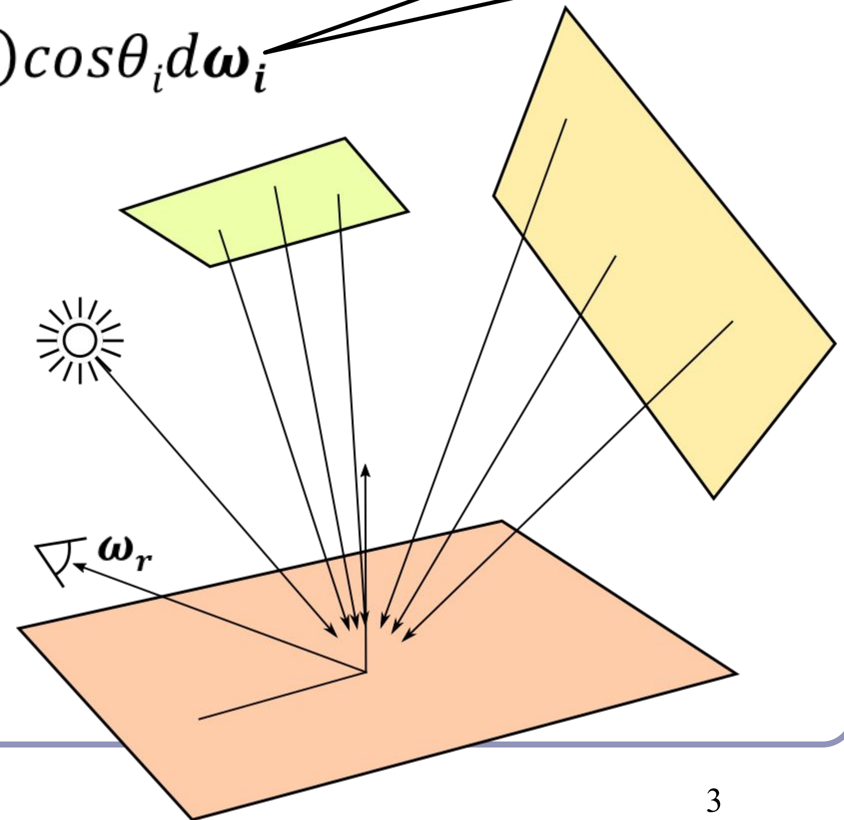
Incident light

Integral over the hemisphere of incident light

$$L_r(\boldsymbol{\omega_r}) = \int_{\Omega} \rho(\boldsymbol{\omega_i}, \boldsymbol{\omega_r}) L_i(\omega_i) cos\theta_i d\boldsymbol{\omega_i}$$

$$\boldsymbol{\omega_i} = [\phi_i, \theta_i]$$

Most rendering methods require solving an (approximation) of the rendering equation

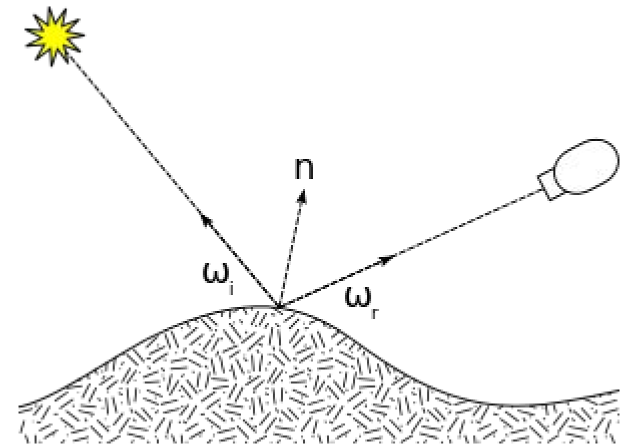


BRDF: Bidirectional Reflectance Distribution Function

Differential radiance of reflected light

$$\rho(\boldsymbol{\omega_i}, \boldsymbol{\omega_r}) = \frac{dL_r(\boldsymbol{\omega_r})}{dH_i(\boldsymbol{\omega_i})} = \frac{dL_r(\boldsymbol{\omega_r})}{L_i(\boldsymbol{\omega_i}) cos\theta_i d\boldsymbol{\omega_i}}$$

Differential irradiance of incoming light



Source: Wikipedia

BRDF is measured as a ratio of *reflected radiance* to *irradiance*

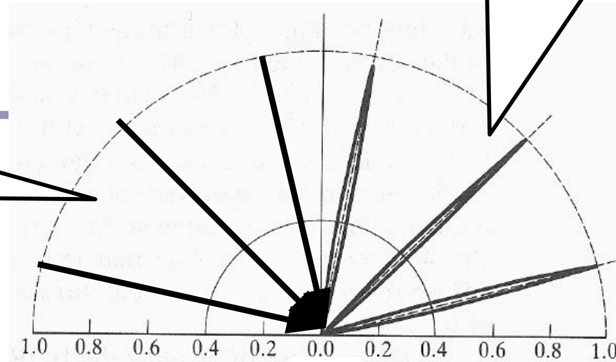
- Because it is difficult to measure $L_i(w_i)$, it's impractical to define BRDF simply as the ratio of $L_r(w_r)$ to $L_i(w_i)$

BRDF of various materials

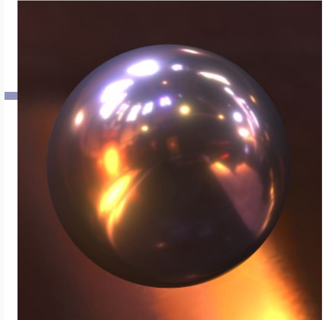
Incident light

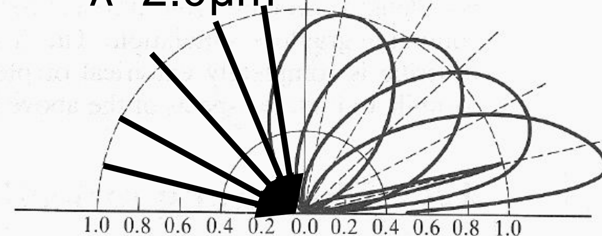
Reflected light

- These diagrams show the distribution of reflected light for the given incoming direction
- The material samples are close but not accurate matches for the diagrams

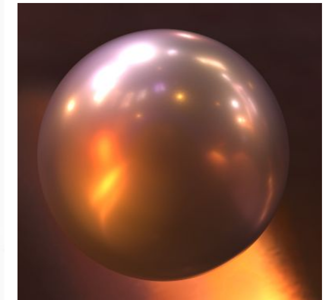


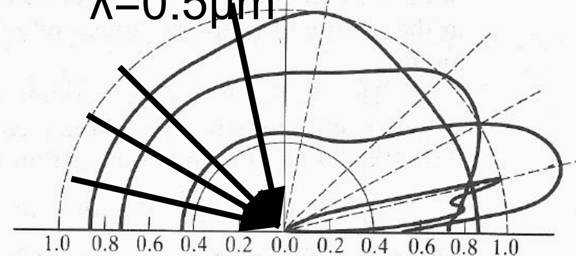
Aluminium; λ=2.0μm





Aluminium; λ=0.5μm

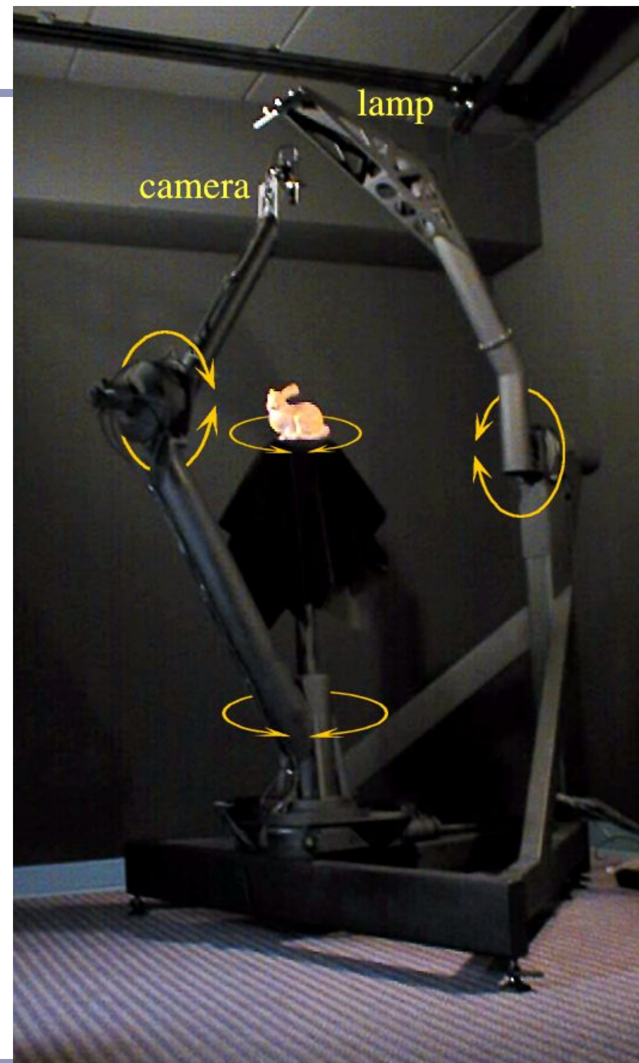




Magnesium alloy; λ=0.5μm

Measuring BRDF

- Gonio-Reflectometer
- BRDF measurement
  - point light source position (θ,ϕ)
  - light detector position (θ_o,ϕ_o)
- 4 directional degrees of freedom
- BRDF representation
  - m incident direction samples (θ,ϕ)
  - n outgoing direction samples (θ_o,ϕ_o)
  - $m*n$ reflectance values (large!!!)



Stanford light gantry

Improving on the classic lighting implementations

- Soft shadows are expensive
- Shadows of transparent objects require further coding or hacks
- Lighting off reflective objects follows different shadow rules from normal lighting
- Hard to implement diffuse reflection (color bleeding, such as in the Cornell Box—notice how the sides of the inner cubes are shaded red and green.)

- Fundamentally, the ambient term is a hack and the diffuse term is only one step in what should be a recursive, self-reinforcing series.

The *Cornell Box* is a test for rendering Software, developed at Cornell University in 1984 by Don Greenberg. An actual box is built and photographed; an identical scene is then rendered in software and the two images are compared.

Ambient occlusion

- *Ambient illumination* is a blanket constant that we often add to every illuminated element in a scene, to (inaccurately) model the way that light scatters off all surfaces, illuminating areas not in direct lighting.
- *Ambient occlusion* is the technique of adding/removing ambient light when other objects are nearby and scattered light wouldn't reach the surface.
- Computing ambient occlusion is a form of *global illumination*, in which we compute the lighting of scene elements in the context of the scene as a whole.

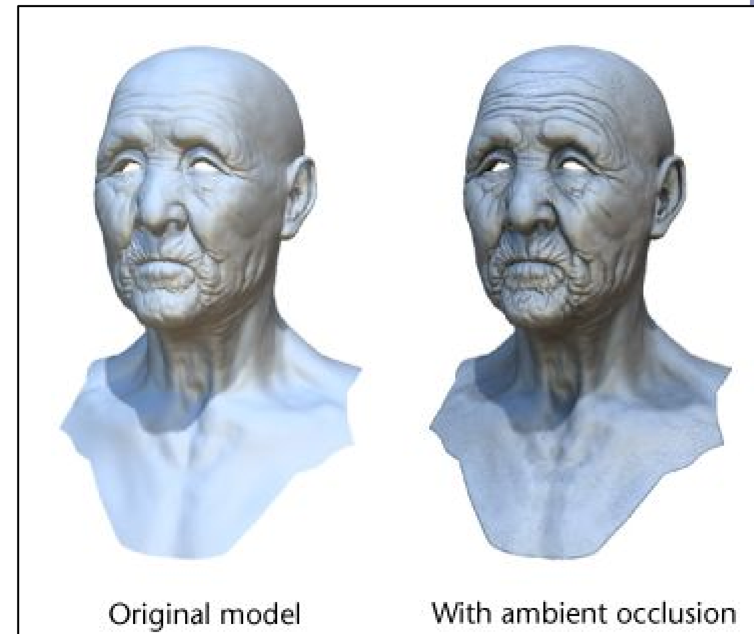


Image from "ZBrush® Character Creation: Advanced Digital Sculpting, Second Edition", by Scott Spencer, 2011

Ambient occlusion in action

Car photos from John Hable's presentation at GDC 2010, "Uncharted 2: HDR Lighting" (filmicgames.com/archives/6)

Ambient occlusion in action

Car photos from John Hable's presentation at GDC 2010, "Uncharted 2: HDR Lighting" (filmicgames.com/archives/6)

Ambient occlusion in action

Car photos from John Hable's presentation at GDC 2010, "Uncharted 2: HDR Lighting" (filmicgames.com/archives/6)

Ambient occlusion in action

Car photos from John Hable's presentation at GDC 2010, "Uncharted 2: HDR Lighting" (filmicgames.com/archives/6)

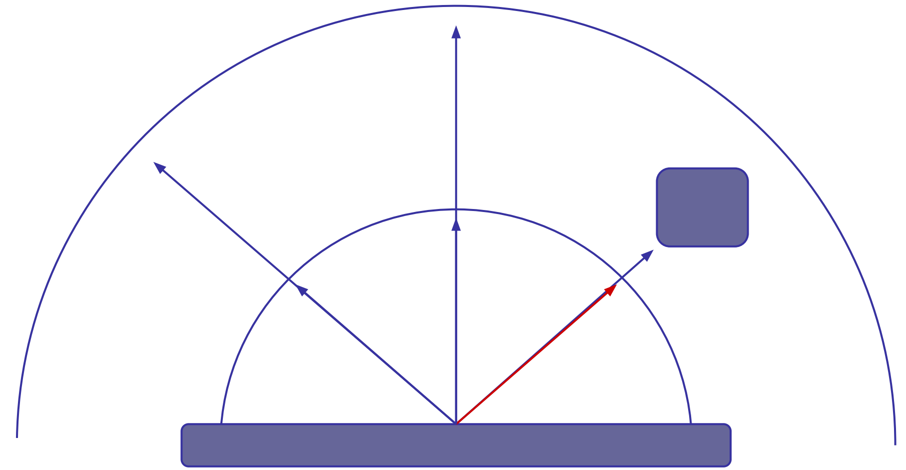
Ambient occlusion - Theory

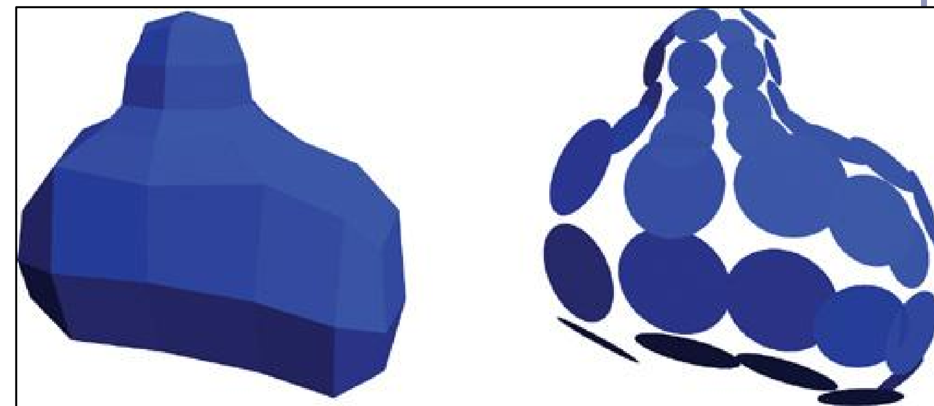
We can treat the background (the sky) as a vast ambient illumination source.

- For each vertex of a surface, compute how much background illumination reaches the vertex by computing how much sky it can 'see'
- Integrate occlusion A_p over the hemisphere around the normal at the vertex:

$$A_{\bar{p}} = \frac{1}{\pi}\int_{\Omega} V_{\bar{p},\hat{\omega}}(\hat{n}\cdot\hat{\omega})d\omega$$

- A_p occlusion at point p
- n normal at point p
- $V_{p,\omega}$ visibility from p in direction ω
- Ω integrate over area (hemisphere)





Bottom image credit: "GPU Gems 2", nVidia, 2005. Vertices mapped to illumination disks for hemispheric illumination mapping.

Ambient occlusion - Theory

- This approach is very flexible
- Also very expensive!
- To speed up computation, randomly sample rays cast out from each polygon or vertex (this is a *Monte-Carlo* method)
- Alternatively, render the scene from the point of view of each vertex and count the background pixels in the render
- Best used to pre-compute per-object "*occlusion maps*", texture maps of shadow to overlay onto each object
- But pre-computed maps fare poorly on animated models...

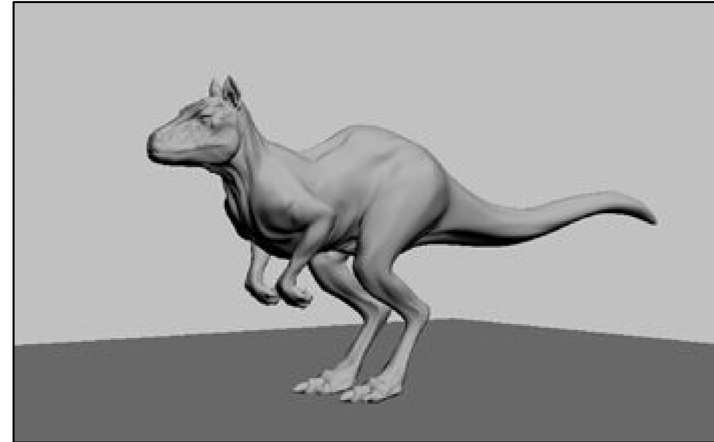


Image credit: "GPU Gems 1", nVidia, 2004.
Top: without AO. Bottom: with AO.

Screen Space Ambient Occlusion ("SSAO")

"True ambient occlusion is hard, let's go hacking."

- Approximate ambient occlusion by comparing z-buffer values in screen space!
- Open plane = unoccluded
- Closed 'valley' in depth buffer = shadowed by nearby geometry
- Multi-pass algorithm
- Runs entirely on the GPU

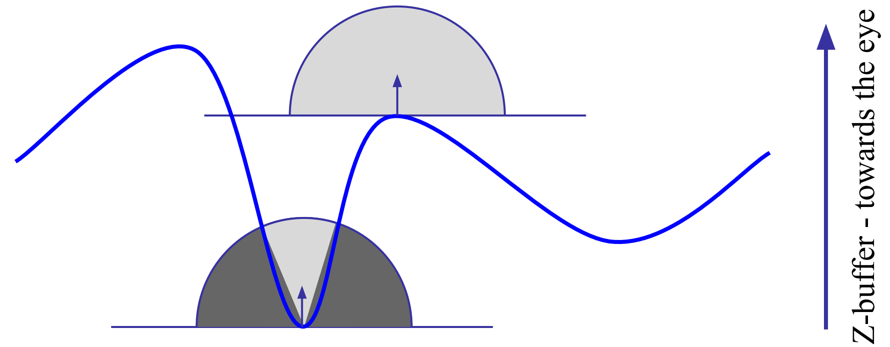
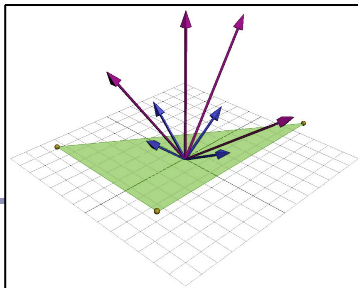


Image: CryEngine 2. M. Mittring, "Finding Next Gen – CryEngine 2.0, Chapter 8", SIGGRAPH 2007 Course 28

Screen Space Ambient Occlusion

1. For each visible point on a surface in the scene (ie., each pixel), take multiple samples (typically between 8 and 32) from nearby and map these samples back to screen space
2. Check if the depth sampled at each neighbor is nearer to, or further from, the scene sample point
3. If the neighbor is nearer than the scene sample point then there is some degree of occlusion
   a. Care must be taken not to occlude if the nearer neighbor is too much nearer than the scene sample point; this implies a separate object, much closer to the camera
4. Sum retained occlusions, weighting with an occlusion function



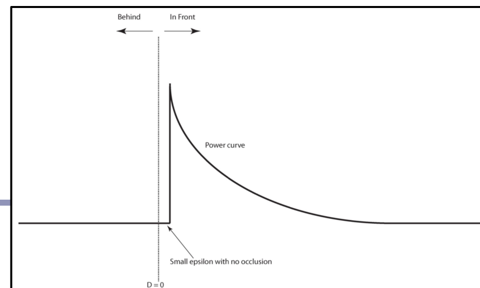


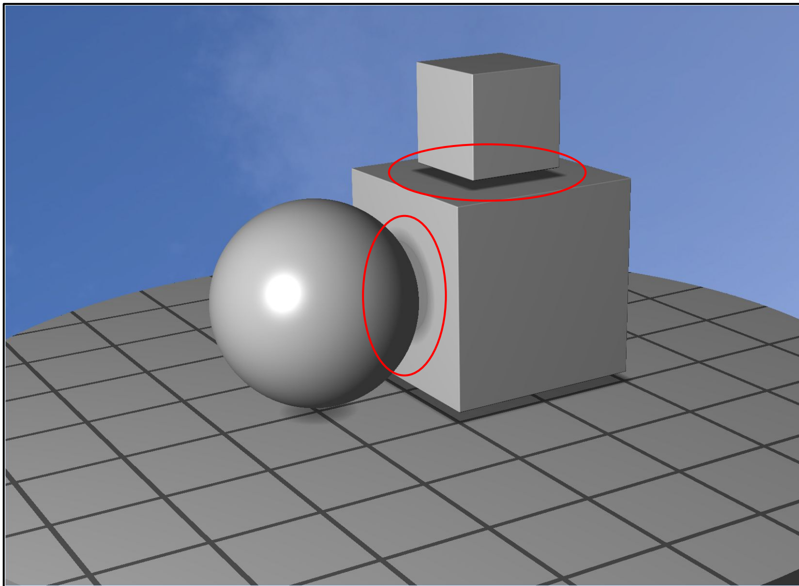
Image: StarCraft II. *Advances in Real-Time Rendering in 3D Graphics and Games* - Course notes, SIGGRAPH 2008

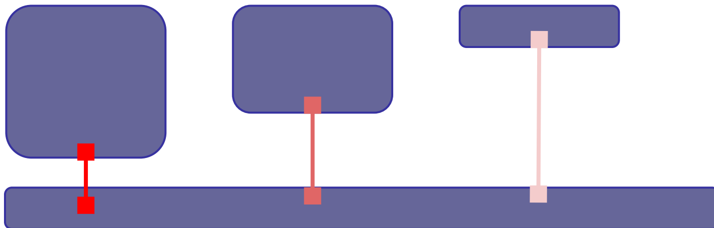
SSAO example- Uncharted 2

4) Low Pass Filter (significant blurring)

John Hable, GDC 2010, "Uncharted 2: HDR Lighting" (filmicgames.com/archives/6)

Ambient occlusion and Signed Distance Fields





In a nutshell, SSAO tries to estimate occlusion by asking, "how far is it to the nearest neighboring geometry?"

With signed distance fields, this question is almost trivial to answer.

```
float ambient(vec3 pt, vec3 normal) {
  return abs(getSdf(pt + 0.1 * normal)) / 0.1;
}
```

```
float ambient(vec3 pt, vec3 normal) {
  float a = 1;
  int step = 0;

  for (float t = 0.01; t <= 0.1; ) {
    float d = abs(getSdf(pt + t * normal));
    a = min(a, d / t);
    t += max(d, 0.01);
  }
  return a;
}
```

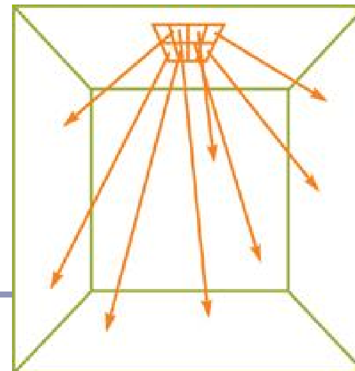
Radiosity

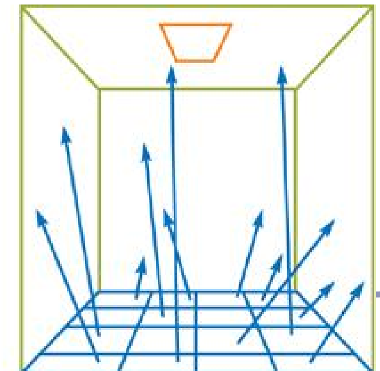
- *Radiosity* is an illumination method which simulates the global dispersion and reflection of diffuse light.
  - First developed for describing spectral heat transfer (1950s)
  - Adapted to graphics in the 1980s at Cornell University
- Radiosity is a finite-element approach to global illumination: it breaks the scene into many small elements (‘*patches*’) and calculates the energy transfer between them.

Images from Cornell University’s graphics group
http://www.graphics.cornell.edu/online/research/

Radiosity—algorithm

- Surfaces in the scene are divided into *patches*, small subsections of each polygon or object.
- For every pair of patches A, B, compute a *view factor* (also called a *form factor)* describing how much energy from patch A reaches patch B.
  - The further apart two patches are in space or orientation, the less light they shed on each other, giving lower view factors.
- Calculate the lighting of all directly-lit patches.
- Bounce the light from all lit patches to all those they light, carrying more light to patches with higher relative view factors. Repeating this step will distribute the total light across the scene, producing a global diffuse illumination model.





Radiosity—mathematical support

The 'radiosity' of a single patch is the amount of energy leaving the patch per discrete time interval.

This energy is the total light being emitted directly from the patch combined with the total light being reflected by the patch:

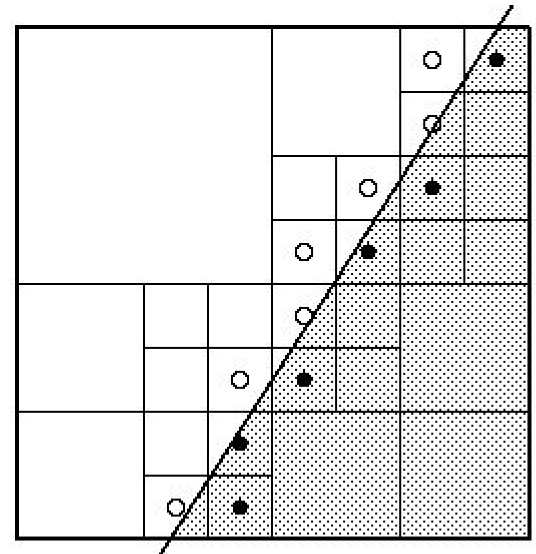
$$B_i = E_i + R_i \sum_{j=1}^{n} B_j F_{ij}$$

This forms a system of linear equations, where…

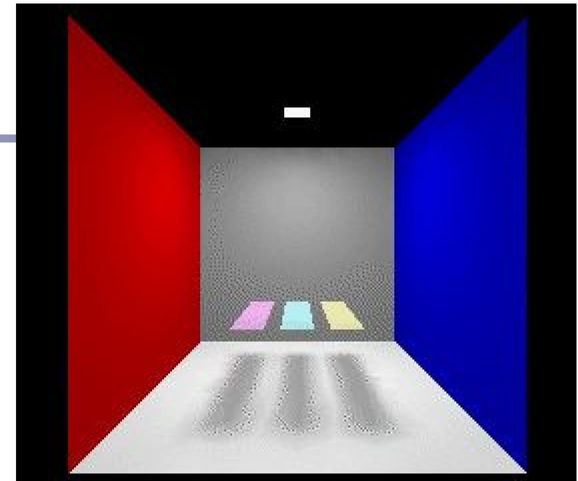
- B_i is the radiosity of patch i;
- B_j is the radiosity of each of the other patches ($j \neq i$)
- E_i is the emitted energy of the patch
- R_i is the reflectivity of the patch
- F_{ij} is the view factor of energy from patch i to patch j.

Radiosity—form factors

- Finding form factors can be done procedurally or dynamically
  - Can subdivide every surface into small patches of similar size
  - Can dynamically subdivide wherever the 1st derivative of calculated intensity rises above some threshold.
- Computing cost for a general radiosity solution goes up as the square of the number of patches, so try to keep patches down.
  - Subdividing a large flat white wall could be a waste.
- Patches should ideally closely align with lines of shadow.



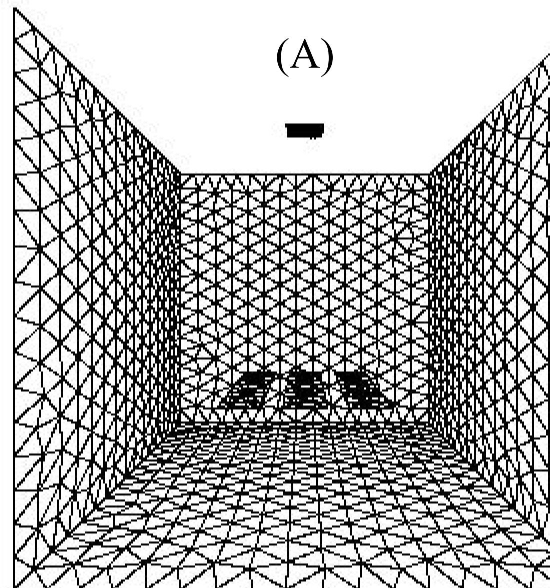
Radiosity—implementation

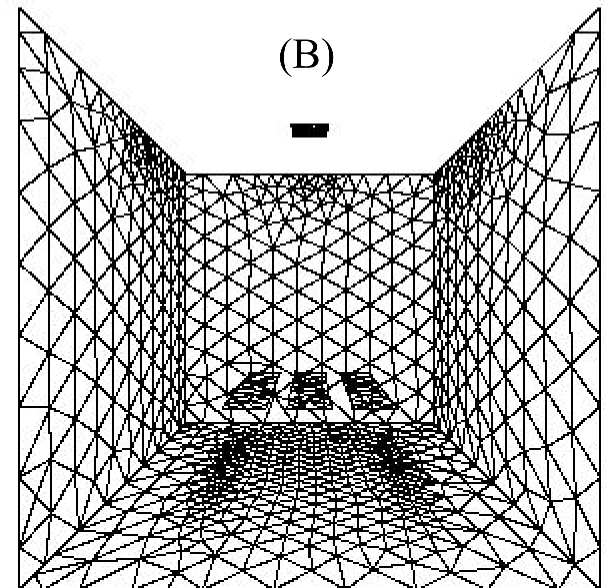


(A) Simple patch triangulation

(B) Adaptive patch generation: the floor and walls of the room are dynamically subdivided to produce more patches where shadow detail is higher.

Images from "Automatic generation of node spacing function", IBM (1998) http://www.trl.ibm.com/projects/meshing/nsp/nspE.htm



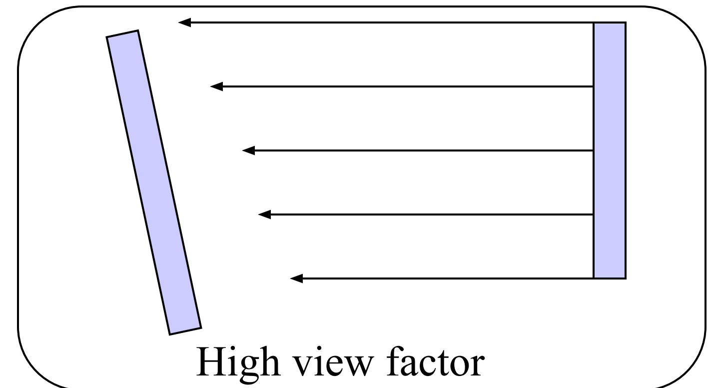


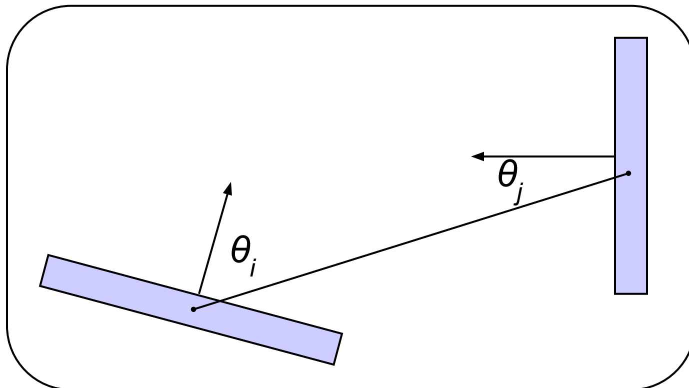
Radiosity—view factors

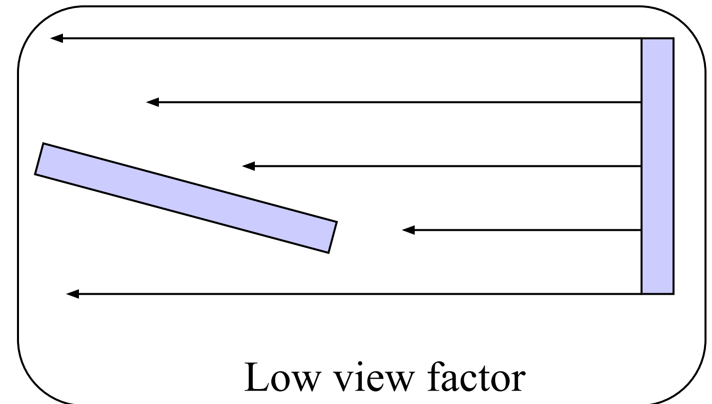
One equation for the view factor between patches i, j is:

$$F_{i \rightarrow j} = \frac{\cos\theta_i \cos\theta_j}{\pi r^2} V(i, j)$$

…where θ_i is the angle between the normal of patch i and the line to patch j, r is the distance and $V(i,j)$ is the visibility from i to j (0 for occluded, 1 for clear line of sight.)

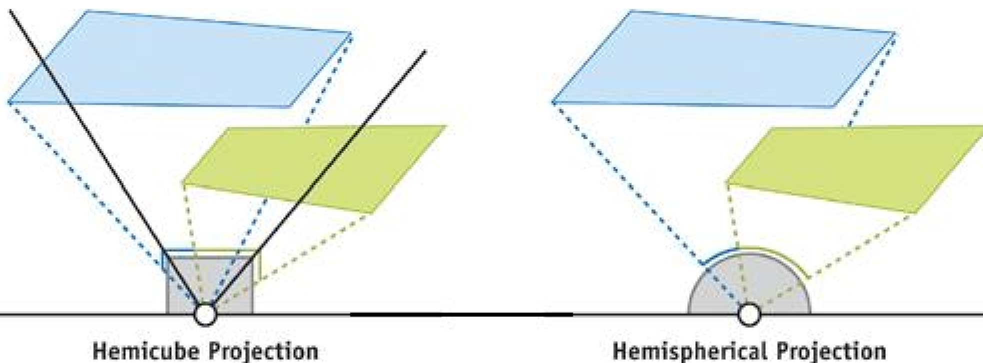

High view factor




Low view factor

Radiosity—calculating visibility

- Calculating *V(i,j)* can be slow.
- One method is the *hemicube*, in which each form factor is encased in a half-cube. The scene is then ‘rendered’ from the point of view of the patch, through the walls of the hemicube; $V(i,j)$ is computed for each patch based on which patches it can see (and at what percentage) in its hemicube.
- A purer method, but more computationally expensive, uses hemispheres.



Note: This method can be accelerated using modern graphics hardware to render the scene. The scene is ‘rendered’ with flat lighting, setting the ‘color’ of each object to be a pointer to the object in memory.

Radiosity gallery

Image from *A Two Pass Solution to the Rendering Equation: a Synthesis of Ray Tracing and Radiosity Methods*,
John R. Wallace, Michael F. Cohen and Donald P. Greenberg
(Cornell University, 1987)

Image from
GPU Gems II, nVidia

Teapot (wikipedia)

References

Shirley and Marschner, "Fundamentals of Computer Graphics", Chapter 24 (2009)

Anisotropic surface:

- A. Watt, 3D Computer Graphics - Chapter 7: Simulating light-object interaction: local reflection models
- Eurographics 2016 tutorial - D. Guarnera, G. C. Guarnera, A. Ghosh, C. Denk, and M. Glencross - BRDF Representation and Acquisition

Ambient occlusion and SSAO:

- "GPU Gems 2", nVidia, 2005. Vertices mapped to illumination. http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter14.html
- Mittring, M. 2007. *Finding Next Gen – CryEngine 2.0,* Chapter 8, SIGGRAPH 2007 Course 28 – Advanced Real-Time Rendering in 3D Graphics and Games http://developer.amd.com/wordpress/media/2012/10/Chapter8-Mittring-Finding_NextGen_CryEngine2.pdf
- John Hable's presentation at GDC 2010, "Uncharted 2: HDR Lighting" (filmicgames.com/archives/6)

Radiosity:

- http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter39.html
- http://www.graphics.cornell.edu/online/research/
- Wallace, J. R., K. A. Elmquist, and E. A. Haines. 1989, "A Ray Tracing Algorithm for Progressive Radiosity." In Computer Graphics (Proceedings of SIGGRAPH 89) 23(4), pp. 315–324.
- Buss, "3-D Computer Graphics: A Mathematical Introduction with OpenGL" (Chapter XI), Cambridge University Press (2003)