Ambien Occlusion

Kadi Bouatouch
IRISA
Email: kadi@irisa.fr

Summary

1. Lighting
2. Definition
3. Computing the ambient occlusion
4. Ambient occlusion fields
5. Dynamic ambient occlusion

Lighting: Ambient Light Sources

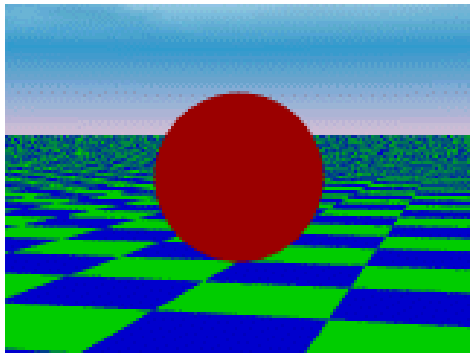
- Objects not directly lit are typically still visible
  - e.g., the ceiling in this room, undersides of desks
- This is the result of *indirect illumination* from emitters, bouncing off intermediate surfaces
- Too expensive to calculate (in real time), so we use a hack called an *ambient light source*
  - No spatial or directional characteristics; illuminates all surfaces equally
  - Amount reflected depends on surface properties

Lighting: Ambient Light Sources

- For each sampled wavelength (R, G, B), the ambient light reflected from a surface depends on
  - The surface properties, $k_{ambient}$
  - The intensity, $I_{ambient,}$ of the ambient light source (constant for all points on all surfaces)
    - $I_{reflected} = k_{ambient} I_{ambient}$

Lighting: Ambient Light Sources

- A scene lit only with an ambient light source:



Light Position Not Important

Viewer Position Not Important

Surface Angle Not Important

Lighting: Ambient Term

- Represents reflection of all indirect illumination

This is a total hack (avoids complexity of global illumination)!

Definiton

The radiance at point x and in direction ωr is:

$$L(\mathbf{x},\vec{\omega}_e) = \int_{\vec{\omega}\in\Omega} L(\mathbf{x},\vec{\omega}) f_r(\mathbf{x},\vec{\omega}_e,\vec{\omega}) V_{\mathbf{X}}(\vec{\omega})(\vec{\omega}\cdot\vec{n}) d\omega$$

If the BRDF fr is Lambertian and we ignore occlusion by objects in the scene (Vx = 1), the above equation can be simplified to:

$$L(\mathbf{x},\vec{n}) \approx f_r(\mathbf{x}) \int_{\vec{\omega}\in\Omega} L(\vec{\omega})(\vec{\omega}\cdot\vec{n}) d\omega$$

Definiton

- Ambient occlusion refers to the attenuation of ambient light due to the occlusion of nearby geometry.
- The ambient occlusion approximation is as follows:

$$L(\mathbf{x},\vec{n}) = \left(\frac{1}{2\pi}\int_{\vec{\omega}\in\Omega} V_{\mathbf{X}}(\vec{\omega}) d\omega\right)\left(f_r(\mathbf{x})\int_{\vec{\omega}\in\Omega} L(\vec{\omega})(\vec{\omega}\cdot\vec{n})\right)$$

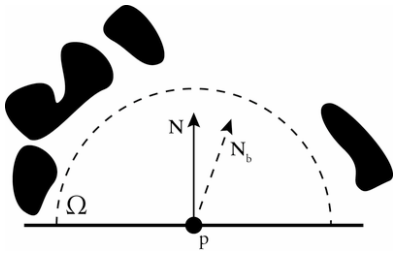
- The first integral is ambient visibility which can be pre-computed and stored in another texture map.

Definiton

- Ambient occlusion refers to the attenuation of ambient light due to the occlusion of nearby geometry.

$$A(\mathbf{x},\mathbf{n}) := \frac{1}{\pi}\int_{\Omega} V(\mathbf{x},\omega)\lfloor\omega\cdot\mathbf{n}\rfloor\, d\omega$$

- Here x is the location and n is the normal vector on the receiving surface. V(x,ω) is the visibility function that has value zero when no geometry is visible in direction w and one otherwise.



Definiton

- Multiplying the classical ambient term with 1−A gives a much better looking result than the dull constant, because the ambient occlusion is able to approximate effects that are otherwise attainable only by computing full global illumination.
- For instance, sharp corners appear darker than open areas and objects cast plausible contact shadows on the surfaces they are resting on.
- Compared to a full global illumination solution, ambient occlusion is signifcantly faster to compute

Definiton

- The self-occlusion of a rigid object can be computed as a preprocess and then re-used in different environments. Since the data can be stored in a texture map or as vertex attributes, the technique has a negligible run-time overhead.
- For this reason, ambient occlusion is gaining interest also in the real-time graphics community.

Computing the Ambient Occlusion

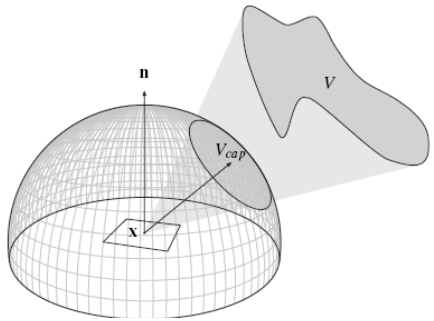
- For each vertex, we compute A using ray tracing or the GPU (rasterization).
- Store it in a the vertex data structure.
- Let x be a point (as seen through a pixel).
- Use the vertex data structure to get the AO values As of the triangle's vertices containing x.
- Interpolate the As to get the AO value Ax of x
- Multiply Ax by the ambient radiance (intensity)

Computing the Ambient Occlusion

- Computing the AO values for each vertex is a demanding process in terms of computing resource.
- Find out fast solutions:
  - Ambien Occlusion Fields
  - Dynamic Ambient Occlusion

Ambient Occlusion Fields Spherical Cap Approximation

- To quickly determine ambient occlusion at an arbitrary point in space around an occluder, we approximate the visibility of the occluder with a spherical cap



Ambient Occlusion Fields Spherical Cap Approximation

- We define $V_{cap}(x,\omega)$ as the visibility function of a cap from position x towards direction ω.
- Thus for each location x there is a corresponding spherical cap representing the occluder.
- $V_{cap}(x,\omega)$ has value one when ω falls within the cap and zero otherwise.
- We get an approximation of ambient occlusion.

$$\tilde{A}(\mathbf{x},\mathbf{n}) = \frac{1}{\pi}\int_{\Omega} V_{cap}(\mathbf{x},\omega)\lfloor\omega\cdot\mathbf{n}\rfloor\, d\omega$$

Ambient Occlusion Fields Spherical Cap Approximation

- To evaluate the above integral we need to be able to construct the function $V_{cap}(x,\omega)$.
- We determine the size of the spherical cap from the solid angle subtended by the occluder. This is defined at position x as.

$$\Omega(\mathbf{x}) := \int_{\Theta} V(\mathbf{x},\omega)\, d\omega$$

- Here Θ refers to integration over a sphere. Ω(x) equals 4π when the object is blocking every direction as seen from point x and zero when the object is not visible at all.

Ambient Occlusion Fields
Spherical Cap Approximation

- As the direction of the cap, we use the average direction of occlusion:

$$\Upsilon(\mathbf{x}) := \text{normalize}\left(\int_{\Theta} V(\mathbf{x},\omega)\omega\, d\omega\right)$$

- Thus Y(x) is the componentwise average of all the directions ω for which the visibility function V(x,ω) evaluates to one.

Ambient Occlusion Fields
Storing Ω and Y

- Now that we have means for evaluating ambient occlusion on an arbitrary surface point based on vector $\Upsilon(x)$ and scalar $\Omega(x)$, we consider how to store these fields compactly in 3D.
- Since our goal is to express accurate contact shadows, we need high resolution near the object and lower resolution suffices at larger distances.
- Optimally, the resolution would depend on the distance from the surface of the object. However, since this is not easily achieved in practice, we use a radial parameterization.
- We parameterize the space by direction w and distance r from the center of the occluder and express the fields with respect to these parameters:

$$\Omega(\mathbf{x}) = \Omega(\omega,r),\ \Upsilon(\mathbf{x}) = \Upsilon(\omega,r).$$

Ambient Occlusion Fields
Storing Ω and Y

- To compactly store Ω and Y, we assume that given a direction ω, the fields behave predictably as functions of radial distance.
- Thus in the following we construct models for both quantities as functions of r.
- For an efficient representation of Ω (ω, r) we use the knowledge that the solid angle subtended by an object is approximately proportional to the inverse square root of r.
- To capture this, we use the following model to express the solid angle:

$$\Omega(\omega,r) \approx \tilde{\Omega}(\omega,r) = \frac{1}{a(\omega)r^2 + b(\omega)r + c(\omega)}$$

- In order to fit the above model to data, the coefficients a(ω), b(ω) and c(ω) have to be determined for each direction ω.

Ambient Occlusion Fields
Storing Ω and Y

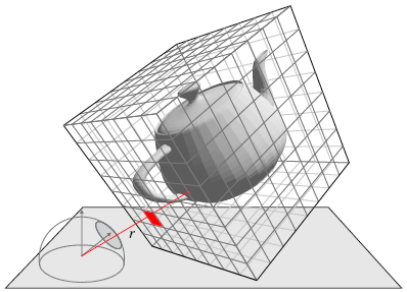
- To find a model for the average direction Y, we note that when going farther away from the object, the average direction approaches the direction towards the center point of the object, while in the close proximity the direction might deviate considerably.
- We model this by:

$$\Upsilon(\omega,r) \approx \tilde{\Upsilon}(\omega,r) = \text{normalize}\,(\mathbf{C}_o(\omega) - r\omega)$$

- In above, given a direction ω, Co(ω) can be understood intuitively as a characteristic point of the occluder, i.e a point in space in which the direction of occlusion mostly points.
- The approximation sets $\tilde{Y}(\omega, r)$ to point from (ω, r) to $C_o(\omega)$. The challenge of fitting this model to data is to find C_o that minimizes a suitable error metric.

Ambient Occlusion Fields
Storing Ω and Y

- The solid angle Ω subtended by an object and the average direction of occlusion Y are stored for each direction as functions of distance r.
- At run-time these functions are fetched from a cube-map and evaluated at the receiving surface in order to compute ambient occlusion



Ambient Occlusion Fields
Implementation: Preprocess

As a preprocess, we use the method of least squares to fit the models $\tilde{\Omega}$ and $\tilde{\Upsilon}$ to the computed occlusion and direction samples. Then these radially parameterized functions are stored into two cube-maps surrounding the object. We denote the resulting discretized direction with $\hat{\omega}$. The stored components are:

$a(\hat{\omega}), b(\hat{\omega}), c(\hat{\omega})$ for $\tilde{\Omega}$	3
$\mathbf{C}_o(\hat{\omega})$ for $\tilde{\Upsilon}$	3
$r_0(\hat{\omega})$, distance from the center to the convex hull	1
Total	7 scalars

Ambient Occlusion Fields
Implementation: Preprocess

- Ray tracing or rasterization can be used for computing the samples.
- We chose rasterization to be able to utilize graphics hardware.
- To compute Ω and Y for a single location, six images have to be rasterized to account for all directions.
- The images contain binary information indicating whether the occluder is visible in a certain direction or not.
- The images are read from the graphics card to the main memory, and Ω and Y are computed

Ambient Occlusion Fields
Implementation: Preprocess

To fit $\tilde{\Omega}$ for each $\hat{\omega}$ we minimize the following error:

$$\varepsilon_\Omega(\hat{\omega}) = \sum_{i=1}^{N} (\Omega(\hat{\omega}, r_i) - \tilde{\Omega}(\hat{\omega}, r_i))^2 \qquad (9)$$

where $\Omega(\hat{\omega}, r_i)$ refers to the sampled occlusion value (Equation 4) and $\tilde{\Omega}(\hat{\omega}, r_i)$ to the approximated value (Equation 6) at the sampling location $(\hat{\omega}, r_i)$. N is the number of sampling locations. The fitting process yields the coefficients a, b and c for each direction $\hat{\omega}$.

Ambient Occlusion Fields Implementation: Preprocess

To fit the model for the average direction $\tilde{\Upsilon}$, we minimize the deviation from the sampled directions:

$$\varepsilon_{\Upsilon}(\hat{\omega}) = \sum_{i=1}^{N} (1 - \tilde{\Upsilon}(\hat{\omega}, r_i) \cdot \Upsilon(\hat{\omega}, r_i))^2 \qquad (10)$$

where $\Upsilon(\hat{\omega}, r_i)$ refers to the sampled average direction (Equation 5) and $\tilde{\Upsilon}(\hat{\omega}, r_i)$ to the approximated direction (Equation 7). The optimization yields $\mathbf{C}_o(\hat{\omega})$, the characteristic point of the occluder.

Ambient Occlusion Fields Implementation: Algorithm

- When rendering a receiving surface, the polynomials are fetched from the cube-map associated with the occluder.
- To compute the ambient occlusion from the direction Y and subtended solid angle Ω , we need to integrate a cosine weighted spherical cap.

We computed a small look-up-table parameterized by solid angle and the elevation angle relative to surface.

- Note that the azimuth angle can be ignored, since it does not affect to the result.

Ambient Occlusion Fields Implementation: Algorithm

Algorithm 1 Pseudocode for the fragment program used to render the receiving surfaces.

Input:
- $\mathbf{x}$ receiver position in field space
- $\mathbf{n}$ receiver surface normal in fields space

Output:
- $\tilde{A}$ ambient occlusion value

Constants:
- r_{near} near falloff distance
- r_{far} far falloff distance
- A_0 inside-hull ambient occlusion
- p inside-hull falloff

Textures:
- $\mathbf{T}_{Occ}$ occlusion data cube-map
- $\mathbf{T}_{\mathbf{C}_\Phi}$ direction data cube-map
- $\mathbf{T}_{LUT}$ spherical cap integration look-up-table

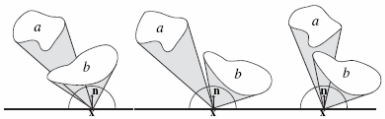
$r = \|\mathbf{x}\|$
$\omega = \texttt{normalize}(\mathbf{x})$
if $r > r_{far}$ **then** `discard`
$(a, b, c, r_0) = \mathbf{T}_{Occ}[\omega]$
$r_{clamp} = \max(r, r_0)$
$\tilde{\Omega} = (ar_{clamp}^2 + br_{clamp} + c)^{-1}$
$\tilde{\Upsilon} = \texttt{normalize}(\mathbf{T}_{\mathbf{C}_\Phi}[\omega] - r_{clamp}\omega)$
$\tilde{A} = \mathbf{T}_{LUT}[\tilde{\Omega}, \mathbf{n} \cdot \tilde{\Upsilon}]$
end if
return $\tilde{A}$

Ambient Occlusion Fields Implementation: Combining Occluders

- We want to get an approximate value for A_{ab} by utilizing the known ambient occlusion values A_a and A_b.
- We are not using knowledge about V_a or V_b, but we may think of three different cases that are
- When an occluder completely overlaps the other one, the combined ambient occlusion is given by picking up the larger of the values A_a and A_b
- When the occluders do not overlap at all the value is given by the sum of the ambient occlusions of each object. It is easy to see that these two cases represent the extremes and the combined ambient occlusion Aab always satisfies:

  $\max(A_a, A_b) <= A_{ab} <= A_a + A_b$

- This suggests the multiplicative blending of the shadow casted by each occluder into the framebu



Ambient Occlusion Fields Implementation: Rendering

Algorithm 2 Simple algorithm for rendering ambient occlusion from multiple casters

```
Render the scene once with ambient light only
for all occluders, O do
  for all receivers, R do
    if R in the region of influence of O then
      Render R with ambient occlusion field of O with multi-
      plicative blending
    end if
  end for
end for
```


Dynamic Ambient Occlusion

GameDevelopers Conference

NVIDIA

Dynamic Ambient Occlusion and Indirect Lighting

Michael Bunnell
NVIDIA Corporation

Dynamic Ambient Occlusion

GameDevelopers Conference

NVIDIA

Environment Lighting

Environment Map + Ambient Occlusion + Indirect Lighting

Dynamic Ambient Occlusion

GameDevelopers Conference

NVIDIA

New Radiance Transfer Algorithm

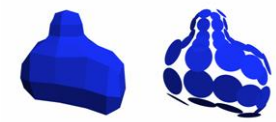
- Useful for calculating Ambient Occlusion and Indirect Lighting
- Efficient and parallelizable
- Implementation is real-time on GPU
- Ideal for non-rigid bodies and dynamic environments

Dynamic Ambient Occlusion

GameDevelopers Conference

NVIDIA

Dynamic Ambient Occlusion



- Define polygon meshes as disk-shaped elements
  - one element created for each vertex
  - elements defined by position, normal, and area
  - simplifies form factor calculation

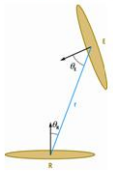
IRISA

Dynamic Ambient Occlusion

GameDevelopers Conference

NVIDIA

Form Factor



Emitter element E occludes receiver element R based on distance r and angles

$$1 - \frac{r \cos\theta_E \max\left(1,\ 4\cos\theta_R\right)}{\sqrt{\frac{A}{\pi} + r^2}}$$

- Percentage of the hemisphere above a point occluded by an element (Solid Angle)
- Like radiosity form factor with 100% visibility

IRISA

Dynamic Ambient Occlusion

GameDevelopers Conference

NVIDIA

Calculating Occlusion

- Calculate occlusion at a receiver element by summing form factors:

```
occlusion = 0;
for each element E
    occlusion += form factor of E;
```

IRISA

Dynamic Ambient Occlusion

GameDevelopers Conference

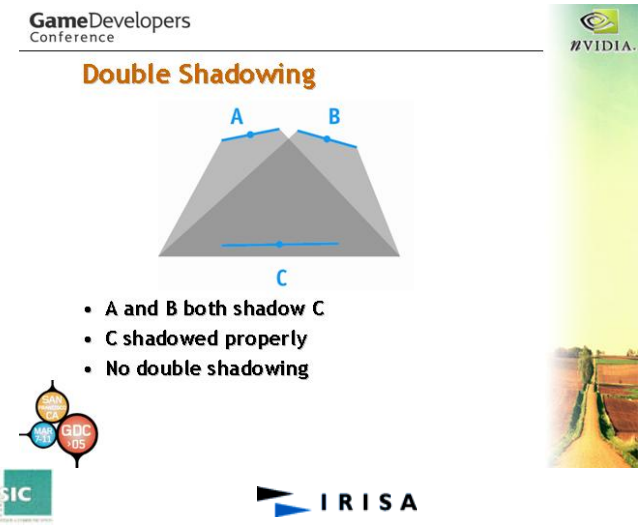
NVIDIA

Element Hierarchy

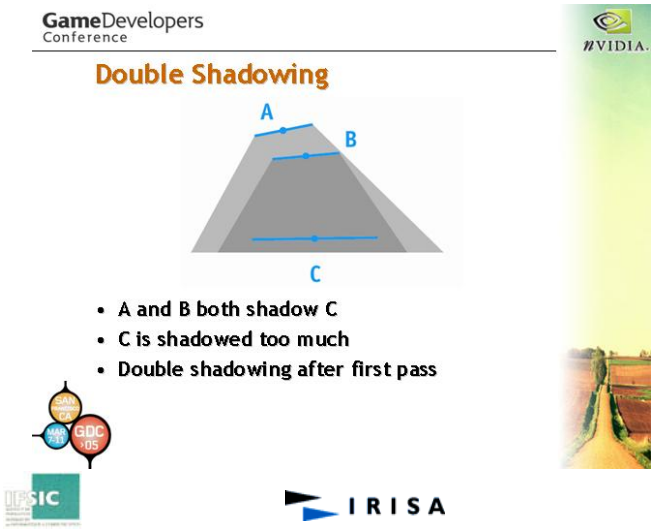
- We do not need to consider so many elements to get an accurate answer
  - A detailed head or simple ball will shadow distant objects the same
- Group elements together, forming larger elements
- Only traverse children when close to parent
- Easy to generate automatically since we don't need actual geometry

IRISA

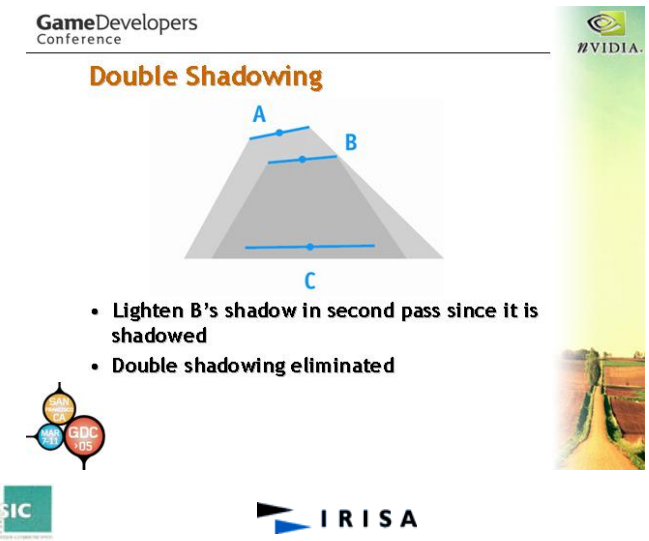
Dynamic Ambient Occlusion



Dynamic Ambient Occlusion



Dynamic Ambient Occlusion



Dynamic Ambient Occlusion

Eliminating Double Shadowing

• We calculate the accessibility values (1 – occlusion) in two passes.
• In the first pass, we approximate the accessibility for each element by summing the fraction of the hemisphere subtended by every other element and subtracting the result from 1.
• After the first pass, some elements will generally be too dark because other elements that are in shadow are themselves casting shadows.
• So we use a second pass to do the same calculation, but this time we multiply each form factor by the emitter element's accessibility from the last pass.
• The effect is that elements that are in shadow will cast fewer shadows on other elements.
• After the second pass, we have removed any double shadowing.
• However, surfaces that are triple shadowed or more will end up being too light. We can use more passes to get a better approximation

Dynamic Ambient Occlusion

GameDevelopers Conference

NVIDIA

Eliminating Double Shadowing

1 pass 2 passes 3 passes Ray traced

- Multiply form factor by 1 - occlusion calculated in the previous pass
- Converges to correct shadowing quickly (2 passes are often enough)
- Results compare favorably with ray tracing

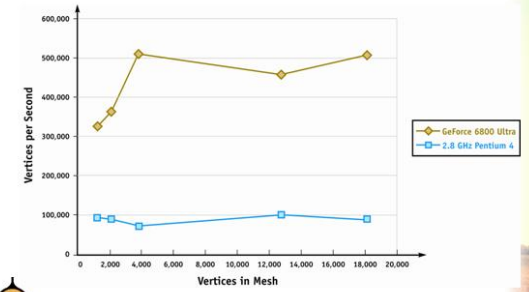
IRISA

Dynamic Ambient Occlusion

GameDevelopers Conference

NVIDIA

Performance



IRISA

Dynamic Ambient Occlusion

GameDevelopers Conference

NVIDIA

Indirect Lighting

- Light reflecting off diffuse surfaces
- Used effectively in Shrek 2
- Adds an extra level of realism
- Can be used with traditional and environment lighting

IRISA

Dynamic Ambient Occlusion Indirect Lighting

- We can add an extra level of realism to rendered images by adding indirect lighting caused by light reflecting off diffuse surfaces.
- We can add a single bounce of indirect light using a slight variation of the ambient occlusion shader.
- We replace the solid angle function with a disk-to-disk radiance transfer function. We use one pass of the shader to transfer the reflected or emitted light and two passes to shadow the light.

IRISA

Dynamic Ambient Occlusion Indirect Lighting

- For indirect lighting, first we need to calculate the amount of light to reflect off the front face of each surface element.
- If the reflected light comes from environment lighting, then we compute the ambient occlusion data first and use it to compute the environment light that reaches each vertex.
- If we are using direct lighting from point or directional lights, we compute the light at each element just as if we are shading the surface, including shadow mapping.
- We can also do both environment lighting and direct lighting and sum the two results. We then multiply the light values by the color of the surface element, so that red surfaces reflect red, yellow surfaces reflect yellow, and so on.
- Area lights are handled just like light-reflective diffuse surfaces except that they are initialized with a light value to emit.

Dynamic Ambient Occlusion Indirect Lighting

- We calculate the amount of light transferred from one surface element to another using the geometric term of the disk-to-disk form factor.

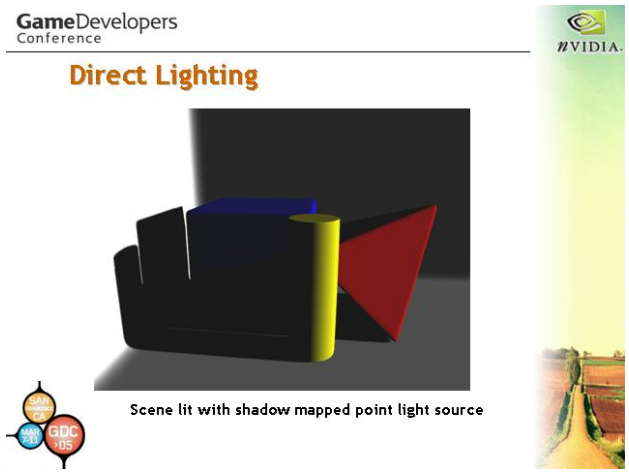
$$\frac{A \cos\theta_E \cos\theta_R}{\pi r^2 + A}$$

- We leave off the visibility factor, which takes into account blocking (occluding) geometry.
- Instead we use a shadowing technique like the one we used for calculating ambient occlusion, only this time we use the same form factor that we used to transfer the light. Multiply form factor by (1 – occlusion)
- Also, we multiply the shadowing element's form factor by the three-component light value instead of a single-component accessibility value.

Dynamic Ambient Occlusion Indirect Lighting

- We now run one pass of our radiance-transfer shader to calculate the maximum amount of reflected or emitted light that can reach any element.
- Then we run a shadow pass that subtracts from the total light at each element based on how much light reaches the shadowing elements.
- Just as with ambient occlusion, we can run another pass to improve the lighting by removing double shadowing.

Dynamic Ambient Occlusion

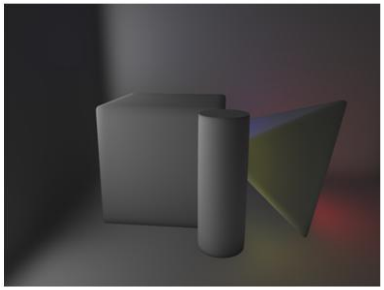


Dynamic Ambient Occlusion

GameDevelopers Conference

NVIDIA

Indirect Light Pass 1



Distribute indirect light in first pass

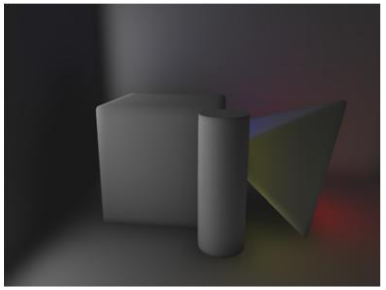
IRISA

Dynamic Ambient Occlusion

GameDevelopers Conference

NVIDIA

Indirect Light Pass 2



Shadow indirect light in second pass

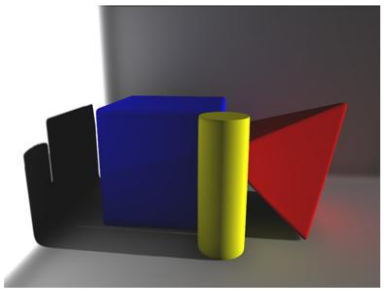
IRISA

Dynamic Ambient Occlusion

GameDevelopers Conference

NVIDIA

Direct Light + 1 Bounce Indirect Light



Indirect light * surface color + direct light

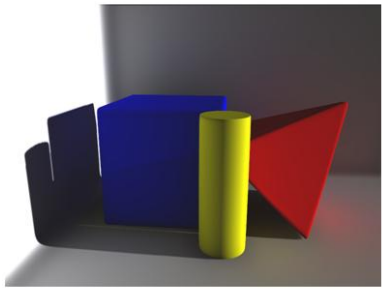
IRISA

Dynamic Ambient Occlusion

GameDevelopers Conference

NVIDIA

Direct Light + 2 Bounces Indirect Light



Second bounce of indirect light takes 2 more passes

IRISA

Dynamic Ambient Occlusion

GameDevelopers Conference

NVIDIA

Indirect Lighting Shader

- Use the same basic shader as ambient occlusion
- Uses standard radiosity disk to disk radiance transfer approximation
- First pass distributes 3-component light values
- One or more subsequent passes shadow that light, subtracting from it
- Area lights can use the same shader

Dynamic Ambient Occlusion

GameDevelopers Conference

NVIDIA

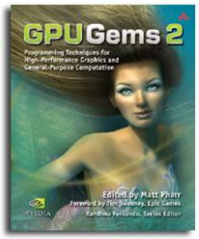
Applications

- Shadow environment lighting of non-rigid objects
- Indirect lighting
- Area lights
- Subsurface scattering*
- Accelerate generation of
  - pre-computed radiance transfer data
  - light maps
  - ambient occlusion data

Dynamic Ambient Occlusion

Dynamic Ambient Occlusion

GPU Gems 2
Programming Techniques for High-Performance Graphics and General-Purpose Computation



- **880 full-color pages, 330 figures, hard cover**
- **$59.99**
- **Experts from universities and industry**

"The topics covered in *GPU Gems 2* are critical to the next generation of game engines."
— Gary McTaggart, Software Engineer at Valve, Creators of Half-Life and Counter-Strike

"*GPU Gems 2* isn't meant to simply adorn your bookshelf—it's required reading for anyone trying to keep pace with the rapid evolution of programmable graphics. If you're serious about graphics, this book will take you to the edge of what the GPU can do."
—Rémi Arnaud, Graphics Architect at Sony Computer Entertainment