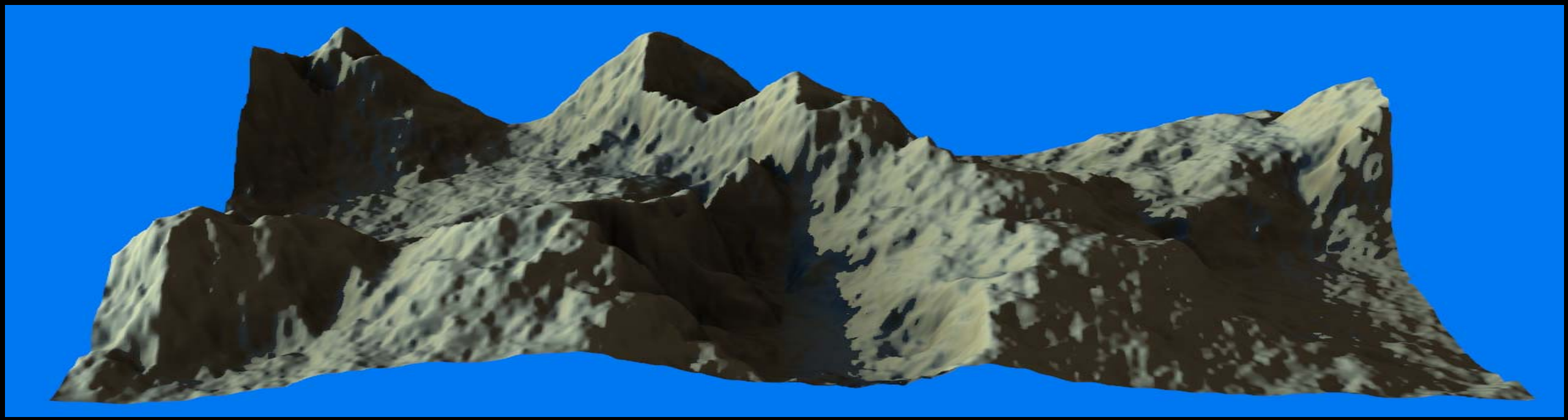
Ambient Aperture Lighting

Chris Oat
3D Application Research Group
ATI Research, Inc.

Outline

- Ambient Aperture Lighting – *45 minutes*
  - Visibility aperture
  - Area light sources
  - Hard & Soft shadows

What is Ambient Aperture lighting?

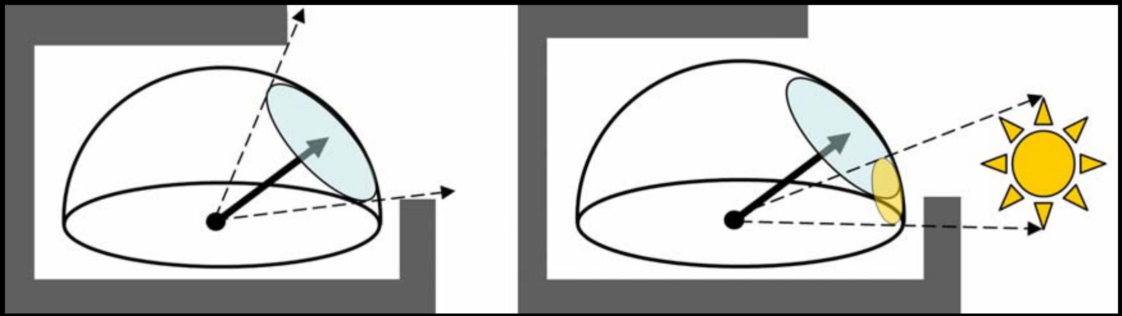


- Shading model that uses apertures to approximate a visibility function
  - ***Precomputed*** visibility
  - **Dynamic** spherical area light sources
  - Dynamic point light sources
  - Hard & Soft shadows
- Similar to horizon mapping, but allows for area light sources
- The “ambient” comes from the fact that we use a modified ambient occlusion calculation to find an aperture of average visibility
- Developed with ***Terrain rendering*** in mind but can be used for other things as well…

What are the applications?

- Non-deformable models
  - Terrains
  - Static scene elements
    - Buildings
    - Statues
- Dynamic spherical area light sources
  - Hard & Soft shadows
- Applications where performance is critical and rendering must still look realistic (but not necessarily physically correct)

How does it work?



- Ambient aperture lighting works in 2 stages
- ***Precomputation Stage***
  - Visibility function is computed at every point on mesh
    - Per-vertex or per-pixel
  - Visibility function is stored using a spherical cap
  - ***Spherical cap*** stores an average, contiguous region of visibility
    - A spherical cap is a portion of a sphere cut off by a plane (a hemisphere itself is a spherical cap)
- ***Rendering Stage***
  - Spherical cap ***acts as an aperture***
  - Aperture is used to restrict incoming light so that it only enters the from visible (un-occluded) directions
  - Area light sources are projected onto the hemisphere and are clipped against the aperture
  - This determines how much of their light passes through the aperture

Precomputation stage

- The precomputation stage can be thought of as a two step process:

- **Step 1:**
  - Find visible area
    - Area of hemisphere that is unoccluded by the surrounding scene
  - This serves as the area of our aperture/spherical cap

- **Step 2:**
  - Find average direction of visibility
    - Just like finding a bent normal
    - Average of all un-occluded rays fired from a given point
  - This serves as the orientation of our aperture/spherical cap

Visible area (aperture size)

SIGGRAPH2006

$$VisibleArea(x) = 2\pi \int_{\Omega} V(x,\omega)\,d\omega$$

- For every point on the mesh (vertex/pixel):
  - Cast a bunch of rays
  - Determine what percentage of rays reach infinity (un-occluded)
  - Multiply by 2PI (area of unit hemisphere)
- The average area of visibility used as aperture size
  - We assume visible area is contiguous and circular region (i.e. a spherical cap)
- Store arc length of the cap's radius
  - arc length of radius = **`acos( -area/2PI + 1 )`**
- Single float value, stored per vertex/pixel

Visible direction (aperture orientation)

SIGGRAPH2006

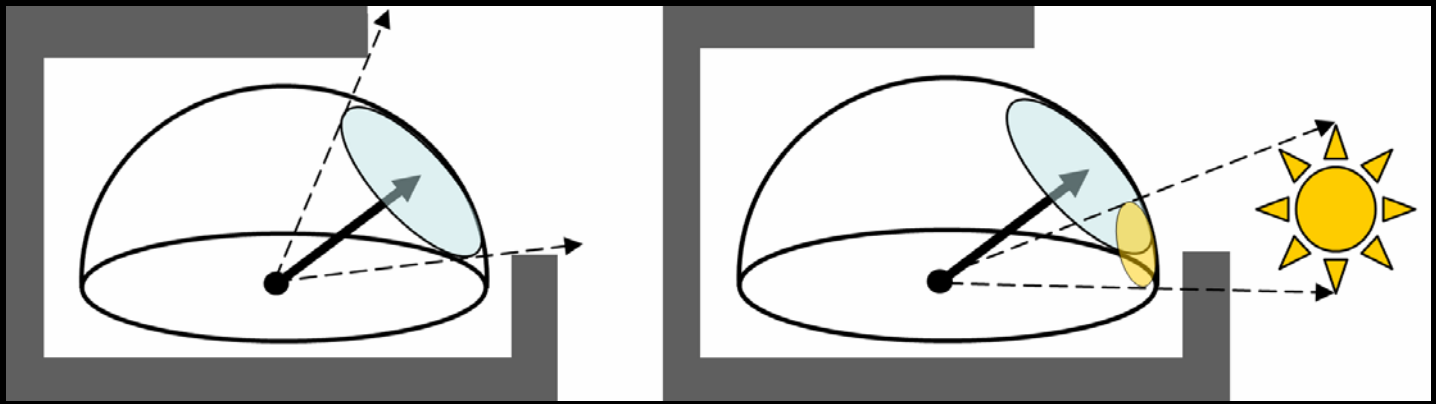
$$VisibleDir(x) = \int_{\Omega} V(x,\omega)\omega d\omega$$

- For every point on the mesh (vertex/pixel):
  - Cast a bunch of rays
  - Determine average direction for which rays reach infinity (un-occluded)
    - This is frequently referred to as a ***bent normal***

- This gives you the average direction of visibility
- Use this for your aperture’s orientation
- A float3 per vertex/pixel

How to render using apertures?

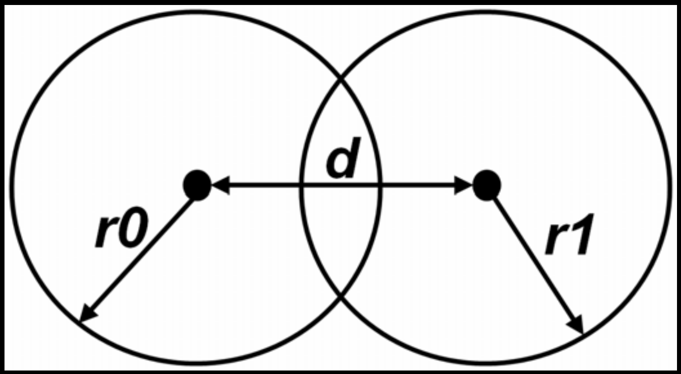
SIGGRAPH2006



- Project spherical area light source onto hemisphere
- Projected area light source covers some area of the hemisphere
  - Projected sphere forms a spherical cap, just like our aperture
- Find the intersection of the projected light's spherical cap and the aperture's spherical cap
- Once the area of intersection is found, we know the portion of the light source that passes through the aperture

Finding area of intersection

SIGGRAPH2006



- Intersection area of two spherical caps is a function of the arc lengths of their radii (r0, r1) **and** the distance between their centroids (d)

- If **d >= r0 +r1**
  - ***No intersection***
  - Thus area is 0

- If **min(r0,r1) <= max(r0,r1)-d**
  - ***Fully intersected***
  - Use the area of the smallest cap
  - Area of cap: $\left(2\pi - 2\pi\cos\left(\min\left(r1, r0\right)\right)\right)$

- Otherwise…

Spherical cap intersection

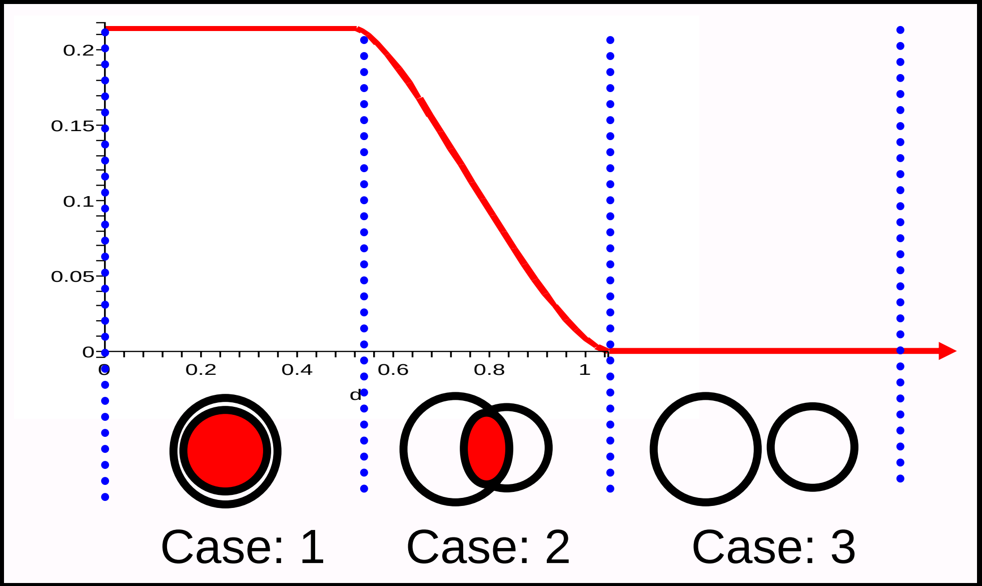
SIGGRAPH2006

$$
\begin{aligned}
&2\cos(r1)\arccos\left(\frac{-\cos(r0)+\cos(d)\cos(r1)}{\sin(d)\sin(r1)}\right) \\
&-2\cos(r0)\arccos\left(\frac{\cos(r1)-\cos(d)\cos(r0)}{\sin(d)\sin(r0)}\right) \\
&-2\arccos\left(\frac{-\cos(d)+\cos(r0)\cos(r1)}{\sin(r0)\sin(r1)}\right) \\
&-2\pi\cos(r1)
\end{aligned}
$$

- **Oh no!**
- After all our simplifications, we’re left with this monster to solve!
- Let’s take a closer look at the intersection area function…

*Simplified form of intersection area function given by [Tovchigrechko]

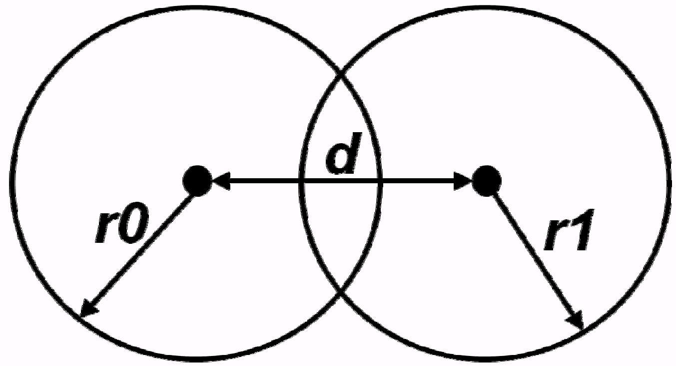
Intersection function



- Case 1 and 3 handled by our early outs
  - ***Case 1*** : *Full intersection*
  - **Case 3** : *No intersection*

- Intersection area decreases as caps move away from each other
- Smooth falloff with respect to distance

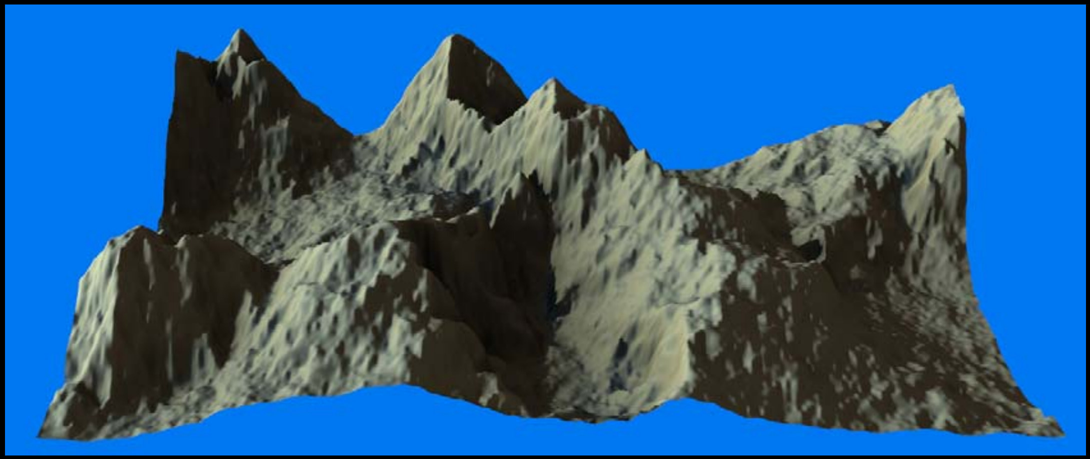
Smoothstep saves the day

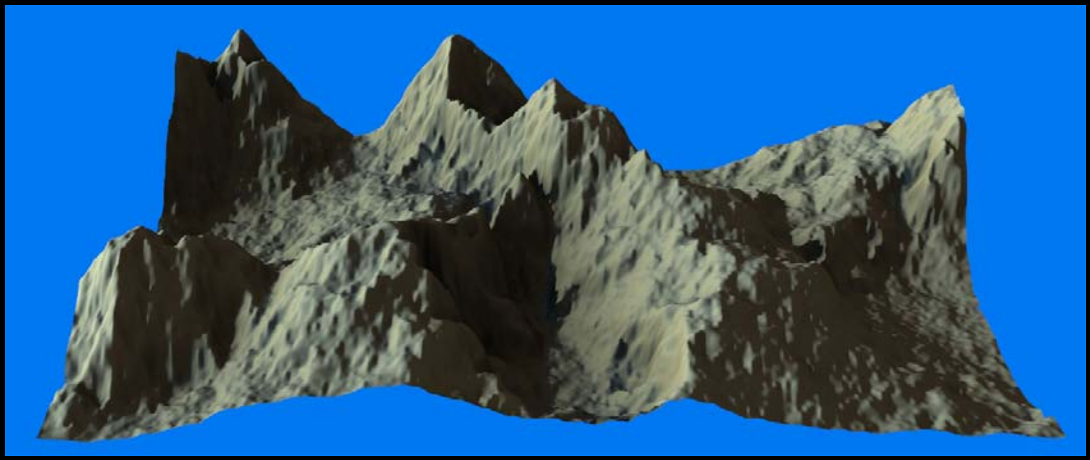


$$\underbrace{(2\pi - 2\pi \cos(\min(r1, r0)))}_{\text{Area of smallest spherical cap}} \ \text{smoothstep}\left(0,1,1 - \frac{d - |r0 - r1|}{r0 + r1 - |r0 - r1|}\right)$$

- Case 1: **Full intersection**
  - Smoothstep returns 1
- Case 2: **Partial intersection**
  - Smoothstep returns smooth falloff (depending on amount of overlap)
  - Gives a smooth transition from full intersection to no intersection
  - Scaled by area of smallest cap
- Case 3: **No intersection**
  - Smoothstep returns 0

Quality Comparison





Top: Exact results Bottom: Approximation

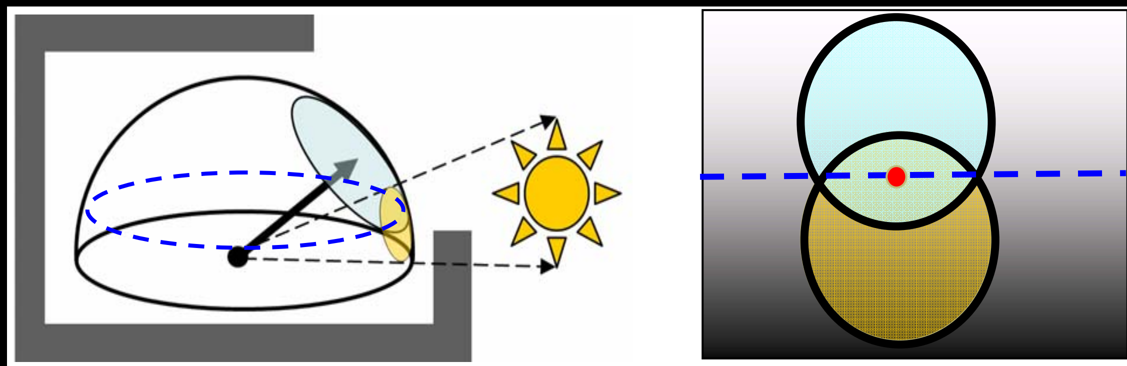
Intersection area approximation

```
// Approximate the are of intersection of two spherical caps
// fRadius0 : First cap's radius (arc length in radians)
// fRadius1 : Second caps' radius (in radians)
// fDist : Distance between caps (radians between centers of caps)
float SphericalCapIntersectionAreaFast ( float fRadius0, float fRadius1, float fDist )
{
   float fArea;

   if ( fDist  <= max(fRadius0, fRadius1) - min(fRadius0, fRadius1) )
   {
      // One cap in completely inside the other
      fArea = 6.283185308 - 6.283185308 * cos( min(fRadius0,fRadius1) );
   }
   else if ( fDist >= fRadius0 + fRadius1 )
   {
      // No intersection exists
      fArea = 0;
   }
   else
   {
      float fDiff = abs(fRadius0 - fRadius1);
      fArea = smoothstep(0.0,
                         1.0,
                         1.0-saturate((fDist-fDiff)/(fRadius0+fRadius1-fDiff)));
      fArea *= 6.283185308 - 6.283185308 * cos( min(fRadius0,fRadius1) );
   }
   return fArea;
}
```

Don't forget about our friend Lambert

SIGGRAPH2006



- Reflectance is determined by the area of intersection ***and*** Lambert's Cosine Law
  - Find a vector to the **centroid** for the region of intersection
  - This is estimated by averaging the aperture's vector and the light's vector
  - Scale the intersection area by N.Vcentroid
    - IntersectionArea * saturate(N.Vcentroid)
  - This provides a Lambertian falloff as the light source approaches the horizon

- Just another approximation on top of all the others we're making ☺
- Assumes the area above intersection's centroid is about the same as the area below the intersection's centroid
  - ***Negative error above*** the centroid **cancels** the ***positive error below*** the centroid

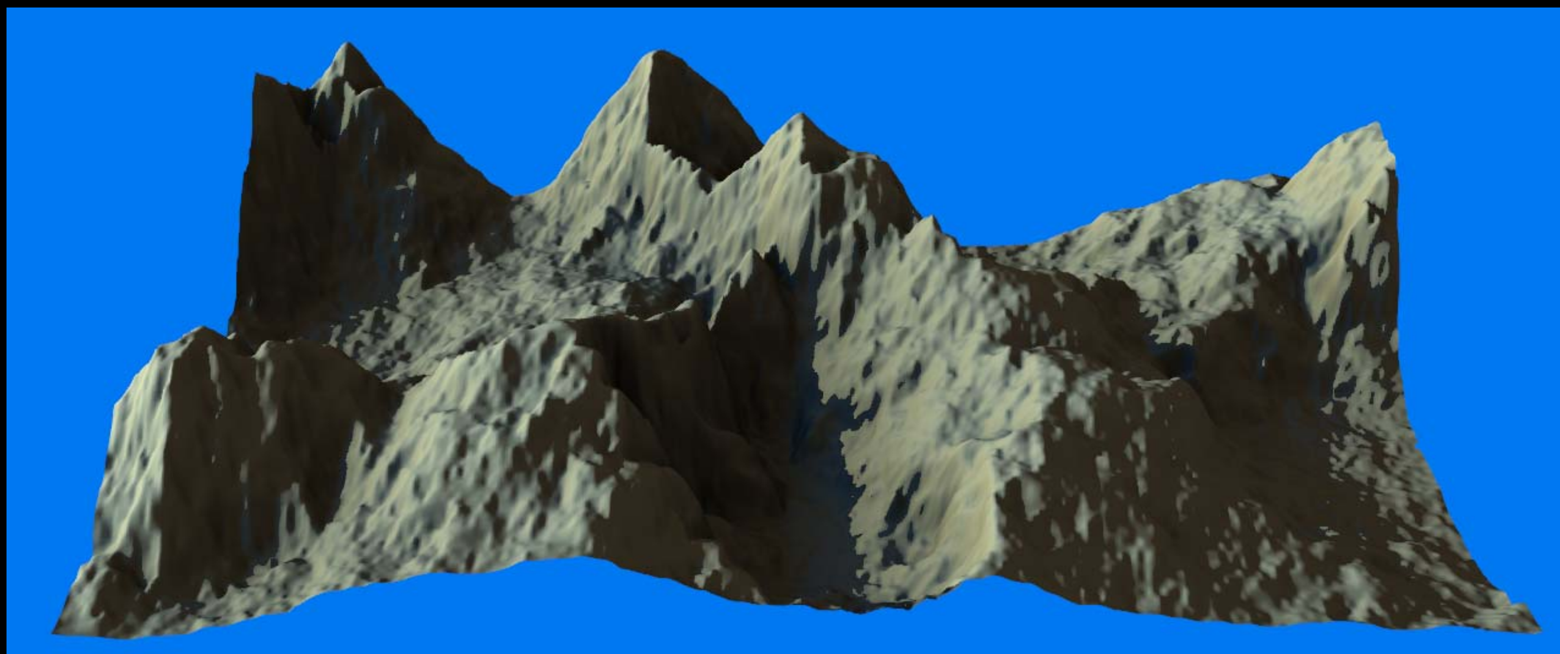
Ambient light

- We now have a function for finding direct lighting from area light sources, but we'd like to incorporate some form of ambient light to account for light scattered in from the sky

- Treat sky as if it were a giant area light behind the sun:
  - Compute area light/aperture intersection
  - If area of intersection is less that area of aperture, fill the missing space with indirect "ambient light"
    - For a terrain, use the average sky color (lowest MIP level of sky dome?)
      - Blue during the day
      - Redish-pink at sun set
      - Black at night

- Works better than the standard constant ambient term
  - Only applies to areas that aren't being lit directly and aren't totally occluded from the outside world

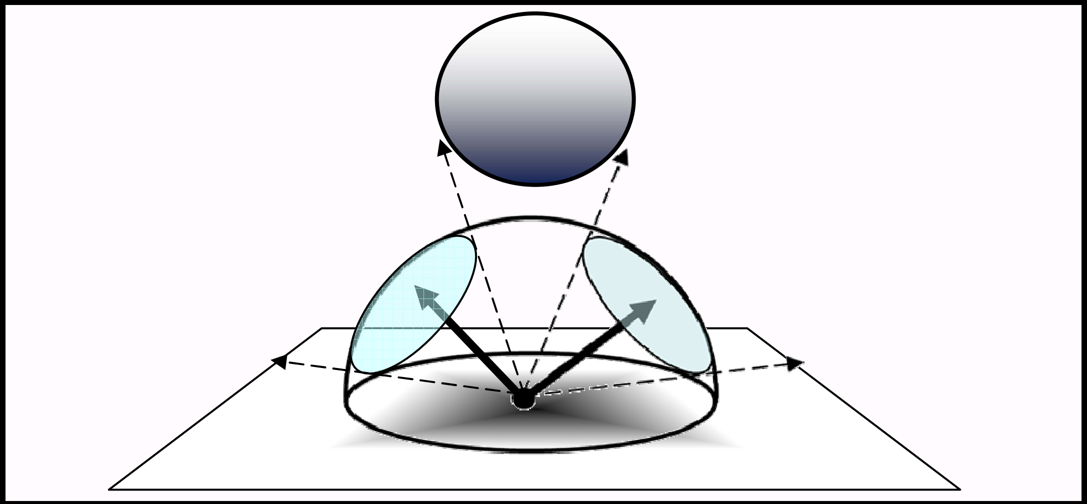
Demo: Terrain



What are the benefits of this technique?

- Area light sources
  - Better than N.L with point light sources
  - Hard shadows for small area light sources
  - Soft shadows for large area lights sources
- Small storage requirements
  - Just 4 floats per-vertex or per-pixel
  - Or 3 floats if you store aperture orientation in tangent space and derive z component in your shader
- Doesn't require additional transforms
  - Shadow maps require transforming model one or more extra times
- Very cheap to compute
  - Just a handful of vertex shader or pixel shader instructions
  - Gives pleasing results

What are the potential downfalls?



- Assumes visible region is contiguous and circular
  - Sphere over plane (see example)
  - **Which way should visibility aperture point?**
  - Visible region is a band around the horizon, this is poorly approximated by a spherical cap
- Multiple light sources don't occlude each other
  - You'd have to compute area of overlap to make sure you don't over light
  - In practice this isn't necessarily a huge issue (people expect 2 light sources to make things twice as bright)
- Assumes non-local light sources
  - Light source can't be between point being shaded and it's blocker
  - Results in incorrect shadowing
- Works well with terrains
  - Terrains typically have nicely behaving visibility functions
  - Occlusion is a band along the horizon
  - Visibility region is generally a contiguous, circular region somewhere in the sky

Taking it to the next level

- Multiple visibility apertures
  - Fixes case where you’re in a room with multiple windows
  - Multiple contiguous regions of visibility
- Occlusion “anti-apertures”
  - Contiguous regions of occlusion
  - Fixes sphere over plane case
  - Spherical cap intersection gives amount of occlusion rather than amount of light

Preprocessor optimizations

SIGGRAPH2006

- Speed up or even eliminate the preprocessing step
  - Exploit the fact that Aperture can be computed using modified ambient occlusion and bent normal preprocessors
- Google for:
  - **GPU accelerated ambient occlusion**
    - Improve preprocessing speed
    - D3DX provides a GPU accelerated SH direct lighting function
      - First coefficient can be used to approximate visible area
      - Next 3 coefficients approximate average visible direction
  - **Dynamic ambient occlusion**
    - Eliminate the need to preprocess
    - Allows for deformable meshes

Conclusion

- A method for shading using dynamic area light sources
- Well suited for outdoor environments
  - Static environment
  - Spherical area light source: Sun
  - Contiguous, circular regions of visibility
- Low computational complexity
- Very low storage cost

References

- MAX, N. L. Horizon Mapping: Shadows for Bump-mapped Surfaces. The Visual Computer 4, 2 (July 1988), 109-117.
- SLOAN, P.-P., KAUTZ, J., SNYDER, J., Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments, SIGGRAPH 2002.
- TOVCHIGRECHKO, A. AND VAKSER, I.A. 2001. How common is the funnel-like energy landscape in protein-protein interactions? Protein Sci. 10:1572-1583

Questions?

Chris Oat

coat@ati.com

These slides are available for download:

www.ati.com/developer