VGP353 – Week 7

- Agenda:
  - Quiz #3
  - Ambient occlusion introduction
  - Real-time calculation of AO
  - Screen-space Ambient Occlusion, part 1

Ambient Lighting

- Hack to approximate global illumination
  - Objects occluded from the light source receive light reflected from other objects
  - Not all locations receive the same amount of indirect light

Ambient Occlusion

- The occlusion at a point is calculated as:

$$A_p = \frac{1}{\pi} \int_{\Omega} V_{p,\omega} (\mathbf{n} \cdot \omega) \, d\omega$$

  - $V_{p,\omega}$ is the visibility function at p in the direction ω

$$V_{p,\omega} = \begin{cases} 0 & \text{if } p \text{ is occluded in the } \omega \text{ direction} \\ 1 & \text{otherwise} \end{cases}$$

Ambient Occlusion

- [Zhukov, et. al 2003] suggest a slightly different formulation

$$A_p = \frac{1}{\pi} \int_\Omega \rho(L(p,\omega))(\mathbf{n} \cdot \omega) \, d\omega$$

  – $L(p, \omega)$ is the distance to the nearest occluder in the ω direction
  – ρ is an arbitrary function with the following properties:

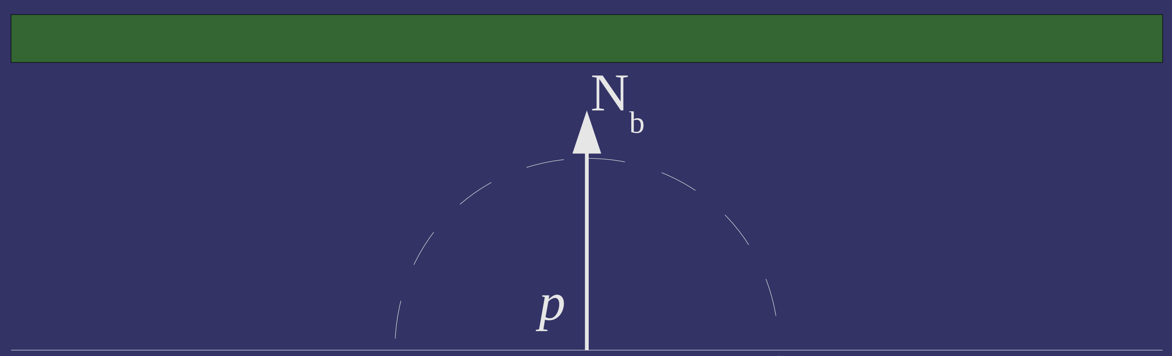
$$\rho(L) = \begin{cases} 0 & \text{for } L = 0 \\ 1 & \text{for } L = +\infty \end{cases} \quad \rho'(L) = \begin{cases} >0 & \text{for } L < +\infty \\ 0 & \text{for } L = +\infty \end{cases} \quad \rho''(L) < 0$$

    – They suggest $(1 - e^{-\tau L})$ where τ is parameter > 0

Average Light Direction Vector

- Calculate the average direction of light arriving at the point
  - Average together unoccluded rays
  - Store delta between this vector and the geometric normal along with the ambient occlusion value
  - Use this “bent normal” to access environment maps or for lighting
    - Attenuate the lighting value using the occlusion factor

Average Light Direction Vector



Average Light Direction Vector

Calculation of Ambient Occlusion

- How can we calculate A_p?

Calculation of Ambient Occlusion

- How can we calculate A_p?
  - Classic answer uses ray tracing:
    - Cast a *large number* of rays from each point on a surface.
    - Each ray that intersects some other surface within a preset distance is occluded

Calculation of Ambient Occlusion

- How can we calculate A_p?
  - Classic answer uses ray tracing:
    - Cast a *large number* of rays from each point on a surface
    - Each ray that intersects some other surface within a preset distance is occluded
  - Can also use a rasterizer:
    - Draw a low resolution hemispherical view from each point on a surface
    - Set far clip plane to limit distance
    - Pixels are either white (not drawn) or black (drawn), and the average is the occlusion value

Calculation of Ambient Occlusion

- Problems:
  - Both methods are expense
  - The expense prevents real-time update
  - Lack of real-time update prevents use on animated models

References

Ambient Occlusion. Internet, http://en.wikipedia.org/wiki/Ambient_occlusion. Accessed on August 29th, 2009.

Landis, Hayden. 2002. "Production-Ready Global Illumination." Course 16 notes, SIGGRAPH 2002. Available online at http://www.renderman.org/RMR/Books/sig02.course16.pdf.

- Chapter 5 covers ambient occlusion.
- Chapter 2 covers techniques for “texture baking.”

Iones, A., Krupkin, A., Sbert, M., and Zhukov, S. 2003. Fast, Realistic Lighting for Video Games. *IEEE Computer Graphics and Applications*. 23, 3 (May. 2003), 54–64. http://ima.udg.edu/iiia/GGG/UsersDocs/mateu/obscurances.pdf

Calculation of Ambient Occlusion

- How can we make the AO calculation faster?
  - We *really* want to use AO with animated models
  - We *really* want to use AO across the whole scene

Calculation of Ambient Occlusion

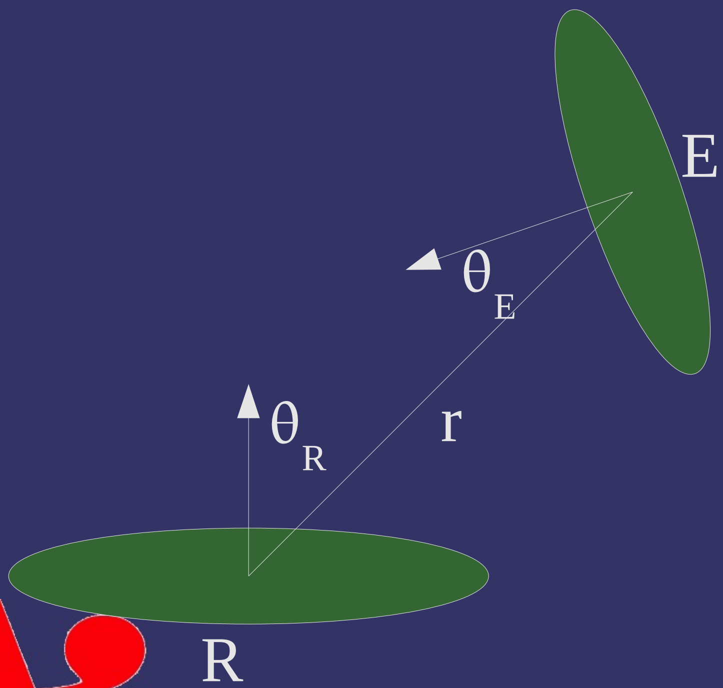
- How can we make the AO calculation faster?
  - We *really* want to use AO with animated models
  - We *really* want to use AO across the whole scene
- Three common strategies:
  - Calculate occlusion factor on GPU using GPGPU techniques (using CUDA, OpenCL, etc.)
    - See [Pharr 04]
  - Calculate approximate occlusion factor
    - See [Bunnel 05]
  - Use screen space ambient occlusion (SSAO)
    - See [Mittring 07] [Shanmugam 07]

Dynamic AO

- Approximate mesh as a set of *surface elements*
  - Each element is represented by an oriented disc
    - Each disc has a position, normal, and area
    - One disc per vertex of the original mesh
  - Disc has two sides
    - Front side emits and reflects light
    - Back side transmits light and shadows
  - Store element information in a texture

Disc-to-disc Occlusion

- Approximate the disc-to-disc occlusion
  - A is the area of the emitter



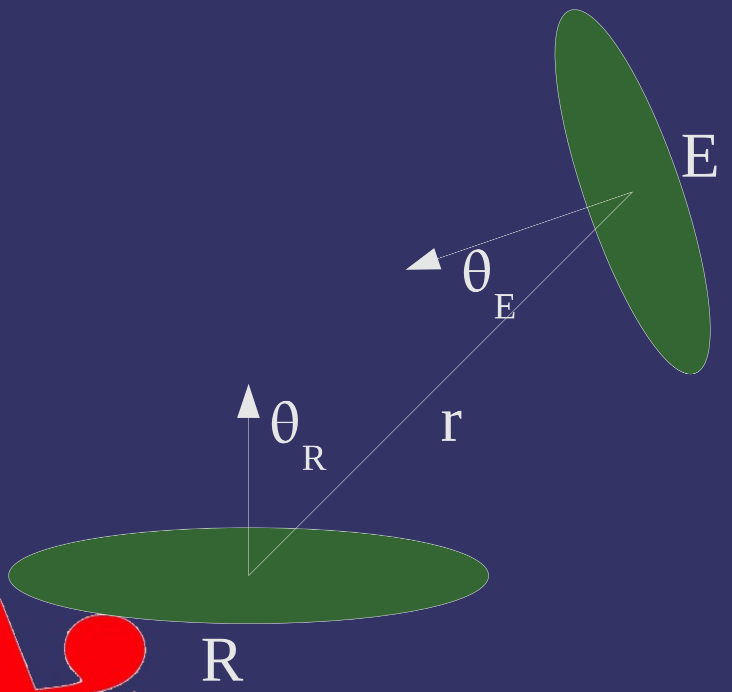
$$1-\frac{r\cos\theta_E\, max\left(1, 4\cos\theta_R\right)}{\sqrt{\frac{A}{\pi}+r^2}}$$

Disc-to-disc Occlusion

- Approximate the disc-to-disc occlusion
  - A is the area of the emitter

Disc-to-disc accessibility

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



Multipass Shadow Algorithm

- First pass:
  - Approximate accessibility for each element as one minus the sum of the accessibility to all other discs
  - After first pass, many surfaces have *too much* shadow
    - Elements that are themselves shadowed still cast shadows
- Second pass:
  - Perform same calculation as first pass
  - Multiply each form factor by the element's accessibility from the first pass
  - Some surfaces *still* have too much light
    - Elements that are triple shadowed

Multipass Shadow Algorithm

- Third pass:
  - Lather, rinse, repeat...

Multipass Shadow Algorithm

- Third pass:
  - Lather, rinse, repeat...
- Too expensive!
  - Just use a weighted average of the first two passes

Performance

- What is the time complexity of the algorithm?

Performance

- What is the time complexity of the algorithm?
  - Accessibility is computed for each of the n elements with each of the other n-1 elements
  - Sounds like $O(n^2)$

Performance

- Performs well because hardware is fast
  - Even an old Geforce 6800 can perform ~150 million calculations per second
  - Can the algorithm be improved to $O(n \log n)$?

Element Hierarchy

- Create a hierarchy of elements
  - Repeatedly merge groups of elements near each other on the mesh
- During processing, traverse the hierarchy
  - Start with the coarsest level of the hierarchy
  - If the element is far enough away, use that. Otherwise move down the hierarchy.
    - The paper suggests 4x the radius of the emitter

Indirect Lighting

- Same data structure can be used to implement a single level of indirect lighting
  - Replace the occluder function with a disc-to-disc radiance transfer function
  - Use one pass to transfer light
  - Use two passes to shadow light

Indirect Lighting

- Calculate the light reflected at each element
  - Computation proceeds as normal using either AO for environment maps or shadow maps for point lights
  - Use the disc-to-disc form factor approximation

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

Indirect Lighting

- Run one pass of the radiance-transfer algorithm
  - Calculate the maximum amount of reflected (or emitted) light that can reach the element
- Run one pass of the shadow algorithm
  - Subtract from each element's total light based on how much light reaches the shadowing elements
  - Can run a third pass to remove double shadowing
    - Just like the dynamic AO algorithm

References

Pharr, Matt and Green, Simon. "Ambient Occlusion" in Fernando, Randima (editor) GPU Gems, Addison-Wesley, 2004. http://http.developer.nvidia.com/GPUGems/gpugems_ch17.html

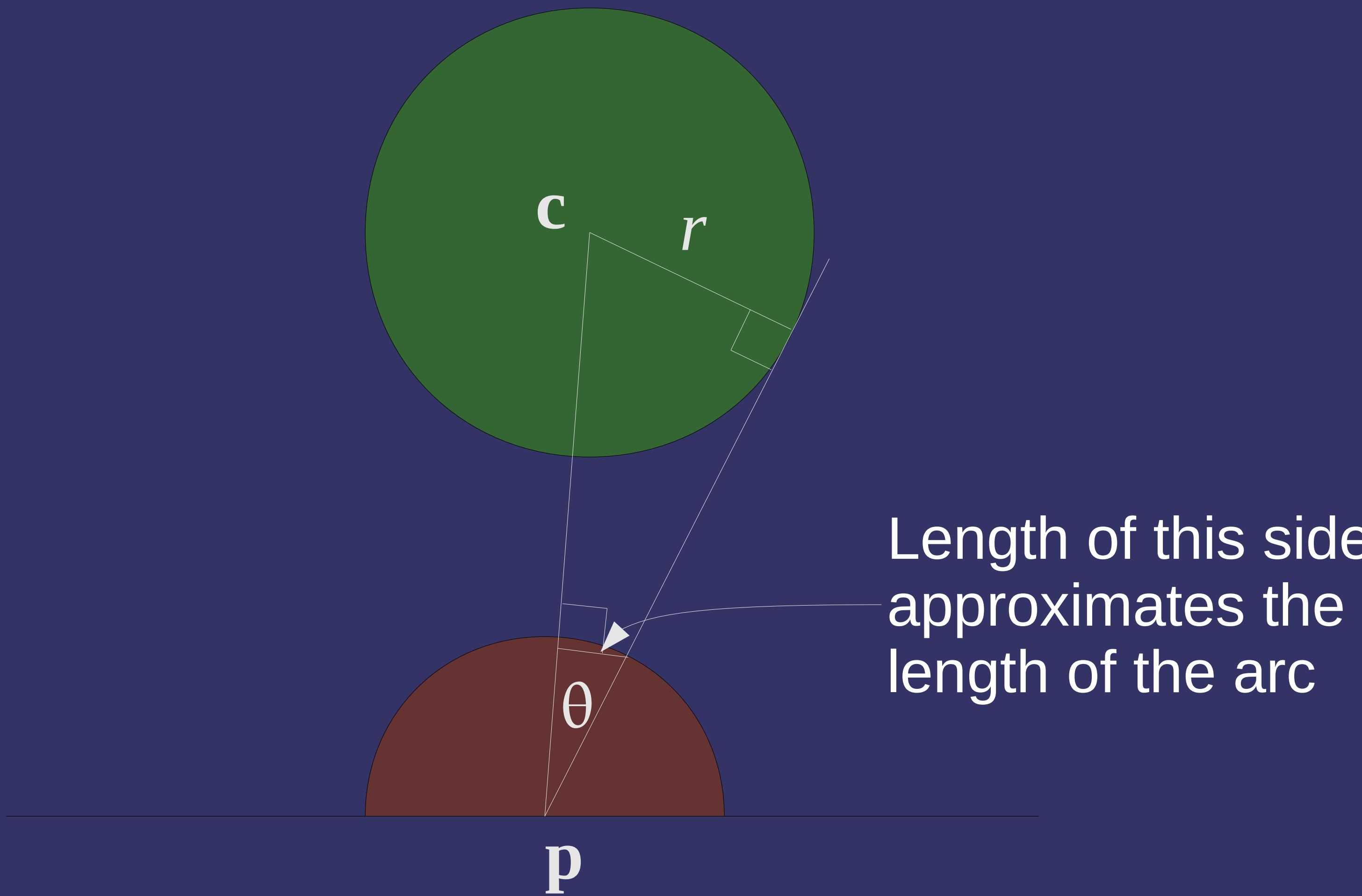
Bunnel, Michael. "Dynamic Ambient Occlusion and Indirect Lighting" in Fernando, Randima (editor) GPU Gems 2, Addison Wesley, 2005. http://download.nvidia.com/developer/GPU_Gems_2/GPU_Gems2_ch14.pdf

SSAO

- Can approximate ambient occlusion using information from the depth buffer
  - First game shipped to use this technique was Crysis by Crytek in 2007
  - The depth buffer is a rough approximation of the scene geometry

SSAO

- Approximate AO (A_Ψ) due to a sphere:



SSAO

- Approximate AO (A_Ψ) due to a sphere:

$$A_\Psi(\mathbf{c}, r, \mathbf{p}, \mathbf{n}) = S_\Omega(\mathbf{p}, \mathbf{c}, r) \max\left(\mathbf{n} \cdot \frac{\overrightarrow{\mathbf{pc}}}{|\overrightarrow{\mathbf{pc}}|}, 0 \right)$$

  - $\mathbf{c}$ and r are the center and radius of the sphere
  - $\mathbf{n}$ is the normal vector at $\mathbf{p}$
  - $\overrightarrow{\mathbf{pc}}$ is the vector from $\mathbf{p}$ to $\mathbf{c}$
  - S_Ω is surface area of the solid angle of the circle

SSAO

- Approximate AO (A_Ψ) due to a sphere:

$$A_\Psi(\mathbf{c}, r, \mathbf{p}, \mathbf{n}) = S_\Omega(\mathbf{p}, \mathbf{c}, r)\,\mathrm{max}\left(\mathbf{n}\cdot\frac{\overrightarrow{\mathbf{pc}}}{|\overrightarrow{\mathbf{pc}}|}, 0\right)$$

$$S_\Omega(\mathbf{p}, \mathbf{c}, r) = 2\pi h$$

$$h = 1 - \cos\theta$$

$$\theta = \sin^{-1}\left(\frac{r}{|\overrightarrow{\mathbf{pc}}|}\right)$$

$$S_\Omega(\mathbf{p}, \mathbf{c}, r) = 2\pi\left(1 - \cos\left(\sin^{-1}\left(\frac{r}{|\overrightarrow{\mathbf{pc}}|}\right)\right)\right)$$

SSAO

- Around each pixel, sample near-by positions:
  - Back project the screen (x, y, z) to camera space
    - Bias the center slightly along -**n** to prevent self-occlusion from flat surfaces
  - Back project the size of the pixel into camera space
    - This sets the size of the sphere
  - Perform approximate sphere AO calculation
- Use resulting sum to modulate color in framebuffer

SSAO

- Straightforward approach requires *piles* of samples to look good
  - The Crysis developers say ~200

SSAO

- Straightforward approach requires *piles* of samples to look good
  - The Crysis developers say ~200
- Use a similar irregular sampling technique as with PCF
  - Unlike PCF, add a geometry-aware filter
  - Rotate the kernel for each pixel
    - Repeat every N pixels
  - Results in only high-frequency noise in the final image

Geometry-Aware Filter

- Perform a normal Gaussian blur or box filter
  - Use an $N \times N$ filter size
  - Do *not* include pixels that span discontinuities
    - Use change in depth
    - Store normals in a secondary buffer and use normals
  - Eliminates most of the high-frequency noise

References

Shanmugam, P. and Arikan, O. 2007. Hardware accelerated ambient occlusion techniques on GPUs. In *Proceedings of the 2007 Symposium on interactive 3D Graphics and Games* (Seattle, Washington, April 30 - May 02, 2007). I3D '07. ACM, New York, NY, 73-80. http://perumaal.googlepages.com/

Screen Space Ambient Occlusion. Internet, http://en.wikipedia.org/wiki/Screen_Space_Ambient_Occlusion. Accessed on August 29th, 2009.

Next week...

- Bilateral filtering
- Depth peeling
- More SSAO
  - Horizon Split AO
  - Multi-Layer Dual-Resolution SSAO
- Read:

Cass Everitt, “Interactive order-independent transparency”, Technical report, NVIDIA Corporation, 2001. http://developer.nvidia.com/object/Interactive_Order_Transparency.html

Tobias Ritschel, Thorsten Grosch, Hans-Peter Seidel. Approximating Dynamic Global Illumination in Screen Space. Proceedings ACM SIGRAPH Symposium on Interactive 3D Graphics and Games, Boston, MA, February 27 —March 1, 2009. http://www.mpi-inf.mpg.de/~ritschel/SSDO/

Legal Statement

This work represents the view of the authors and does not necessarily represent the view of Intel or the Art Institute of Portland.

OpenGL is a trademark of Silicon Graphics, Inc. in the United States, other countries, or both.

Khronos and OpenGL ES are trademarks of the Khronos Group.

Other company, product, and service names may be trademarks or service marks of others.