Real-Time Depth Buffer Based Ambient Occlusion

Miguel Sainz – NVIDIA
msainz@nvidia.com
Motivation

- Ambient occlusion approximates soft outdoor lighting (sky light)
- It gives perceptual clues of depth, curvature, and spatial proximity
- Traditional methods require pre-processing (bad for dynamic scenes)
- It can be done as a post-processing pass
Ambient occlusion

- Weighted by cosine term for diffuse shading
- \( V(\vec{\omega}) \) typically attenuated with distance to \( P \)

\[ A_P(\vec{n}) = 1 - \frac{1}{\pi} \int_{\Omega} V_P(\vec{\omega})(\vec{n} \cdot \vec{\omega}) d\omega \]

- Weighted by cosine term for diffuse shading
- \( V(\vec{\omega}) \) typically attenuated with distance to \( P \)
Why screen space?

- Depth buffer is an approximation of the scene
- Depth buffer is available as part of normal rendering
- No dependency on primitive count
- No pre-calculation of any kind
Related work

[Shanmugam and Okan 07]

- Approximate eye-space pixels by micro spheres
- Accumulate occlusion of spheres in a kernel
- Over-occlusion artifacts (multiple neighboring spheres contribute to the same pixel)
Horizon Split AO

- Treat depth buffer as a height field and process each pixel
- Sample a set of directions over the hemisphere
  
  Evaluate a fan of slices around the normal on each surface pixel
  
  Use randomization
- Split AO integral in two parts for each slice
The hemisphere can be partitioned in two parts based on the "horizon" around P:

- Red rays are always occluded
- Blue rays are undetermined and need to be traced

The horizon angle can be found by stepping along the tangent $T(\theta)$.
Horizon determination

On a given slice around the normal, iterate N steps while deflecting the tangent ray in the normal direction
Horizon integration

- Piece-wise linear approximation of the horizon

\[ O_T = \frac{1}{N_d} \sum_{i=1}^{N_d} H^2(\theta_i)dz \]

- Efficient integral of occlusion contribution
For some surface orientations the tangent tracing will not hit any surface.
Normal occluders

- Evaluated by ray marching
- Angle range constrained by the horizon and the normal
- Lambertian distribution of rays
- Simple AO contribution

\[ O_{N,i}(z_k) = \frac{1}{N_d} \left( z_{k+1}^2 - z_k^2 \right) \cdot V(\vec{\omega}) \]
Additional Tweaks

- Linear attenuation
  Remove banding discontinuities at the boundaries

- Smart Blur
  Remove the noise artifacts
  Use depth information to avoid edge leaking

- Final compositing
  Add as an ambient term
Dragon

Resolution: 800x600
AO Render Time: 7.9ms
Trace Radius: 1/4 Model's Width
Horizon Rays: 8
Number of steps: 8
Normal Rays per Direction: 1
Dragon

Resolution: 800x600
AO Render Time: 26.9 ms
Trace Radius: 1/4 Model's Width
Horizon Rays: 16
Number of steps: 16
Normal Rays per Direction: 1
Optimizations

- New formulation - Done
  Improved integration space and math
  No need for marching additional normal rays. 1.5x faster

- Use previous frame(s) information
- Downscale ND buffers

...
Demo time!
For more information

Contact me at msainz@nvidia.com

Slides, code and whitepaper will be soon available at developer.nvidia.com

(Co-authors: Rouslan Dimitrov and Louis Bavoil)
Questions?