Baking Ambient Occlusion in the glTF Pipeline

Analytical Graphics, Inc.
Kangning (Gary) Li
https://github.com/AnalyticalGraphicsInc/gltf-pipeline

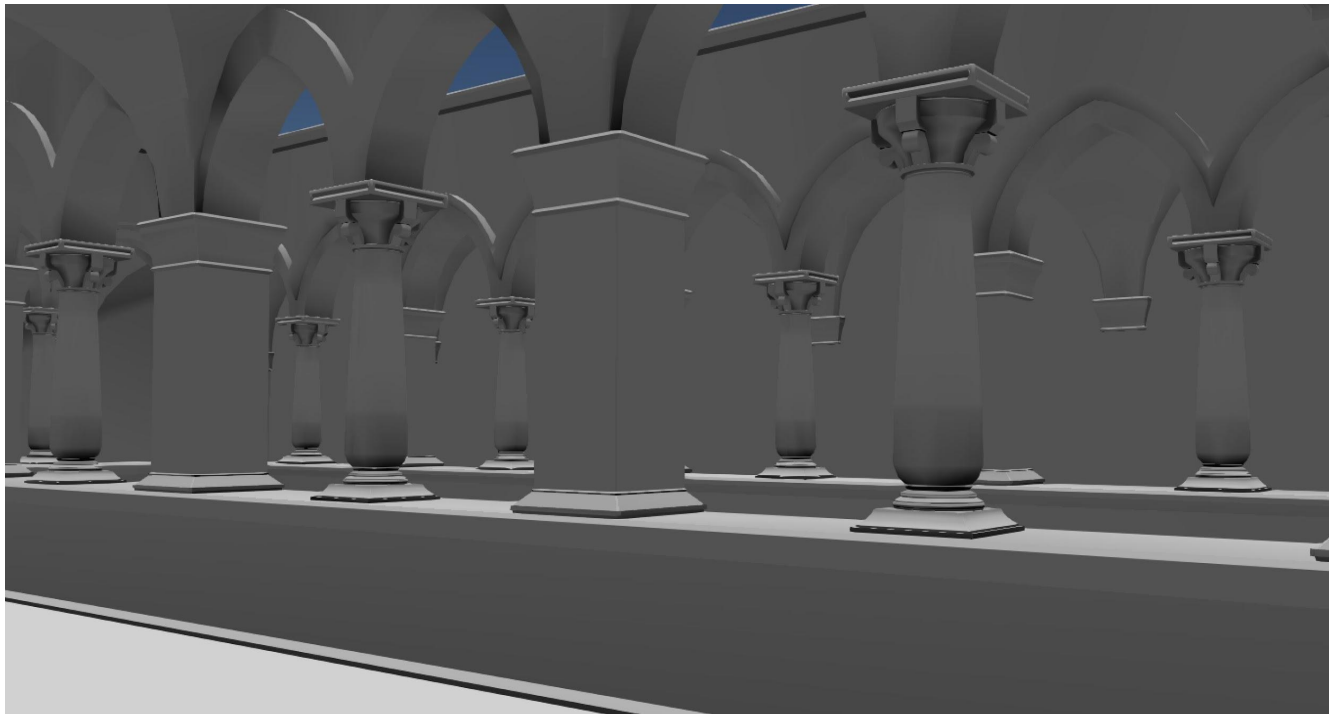
Ambient Occlusion?

- **Ambient** Light - few real surfaces are truly “unlit”
- **Occlusion** - light has a harder time getting into enclosed spaces
  - Crevices
  - Inside corners
  - Small details
- **Ambient Occlusion**, or **AO** - shadows from occluding ambient light
  - Independent of scene lighting
  - User can specify how “small” a space has to be to occlude ambient light
- Baking: AO precomputed and “painted” onto the model
  - “Paint” data directly on the model’s vertices, add vertex attribute, modify shader
  - Or “paint” data into model’s existing texture - could use some improvements...

glTF Pipeline?

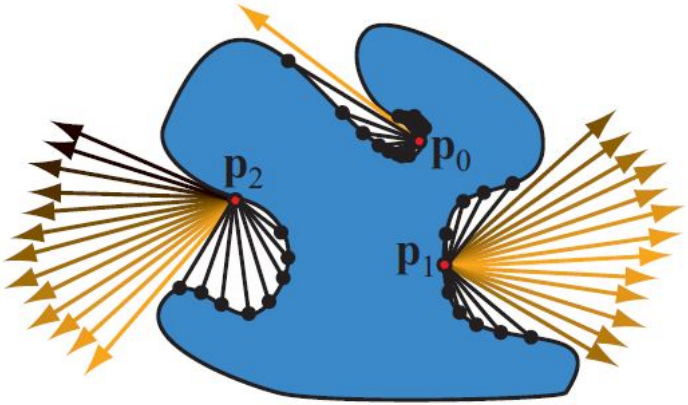
- Open-source pipeline for cleaning glTF, adding features
  - Shaders/techniques/materials
  - Geometry compression
  - Ambient Occlusion!
- Availability:
  - Integration with online glTF converter coming soon
  - Also available via npm and github
- Significance: Ambient Occlusion is not just for Cesium but for ALL glTF!
  - Pipeline handles AO computation, shader/primitive modification as needed
  - AO in existing glTF engines becomes “free” - no modification required!

Results - Pretty pictures!



How do you compute AO?

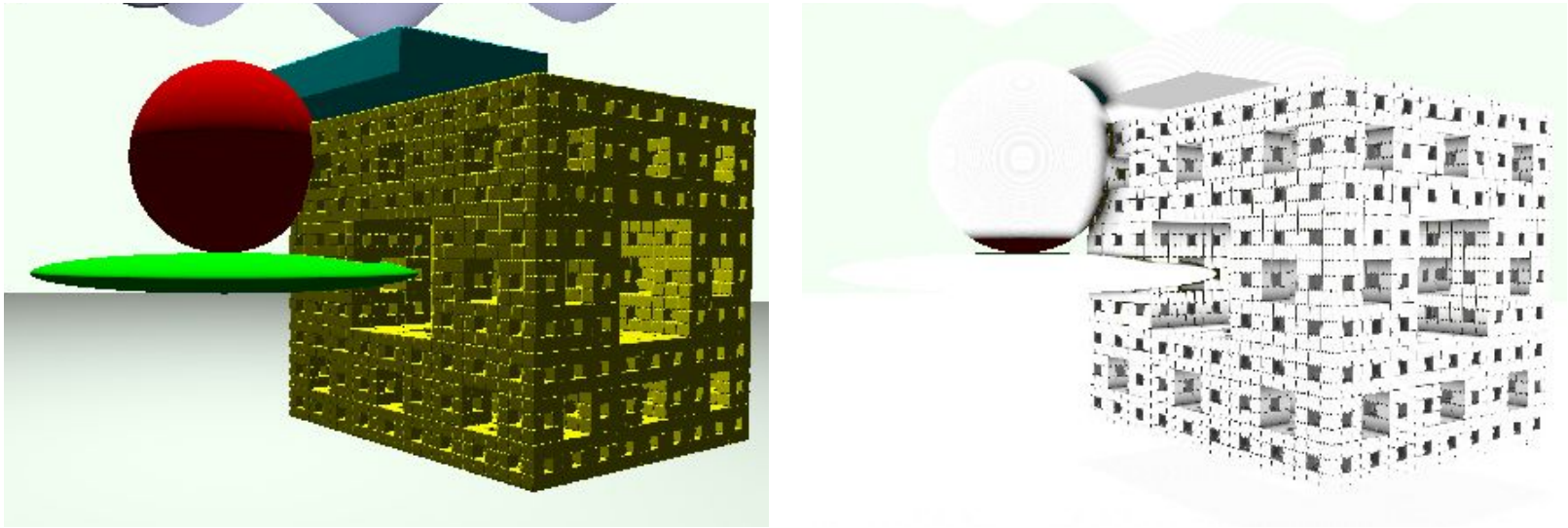
- Fundamental question of AO: How “occluded” is this point on the model?
- Can approximately answer this question with **raytracing**



- Shoot a bunch of rays in a hemisphere pattern, see if they hit anything
- Problem: checking every ray against every triangle is slow!

Optimization: Problem Statement

- Raytracing: “Where is the first intersection for this ray, and what does it hit?”
- AO: “Does this ray hit anything within a specified distance?”

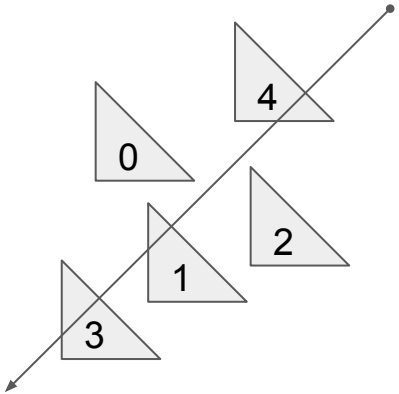


Diffuse shading vs. Ambient Occlusion in Shadertoy

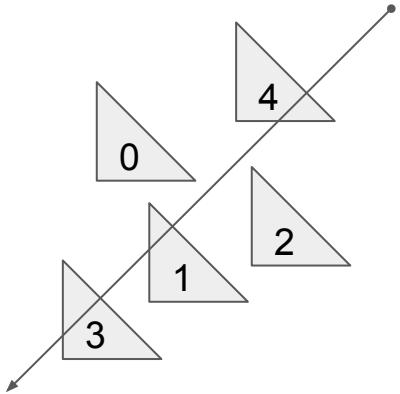
Optimization 1: Early Return

- Raytracing: “**Where** is the first intersection for this ray, and **what** does it hit?”
- AO: “**Does** this ray hit anything within a specified distance?”

Conventional Raytracing:
check every triangle in order to figure out that we hit #4 first



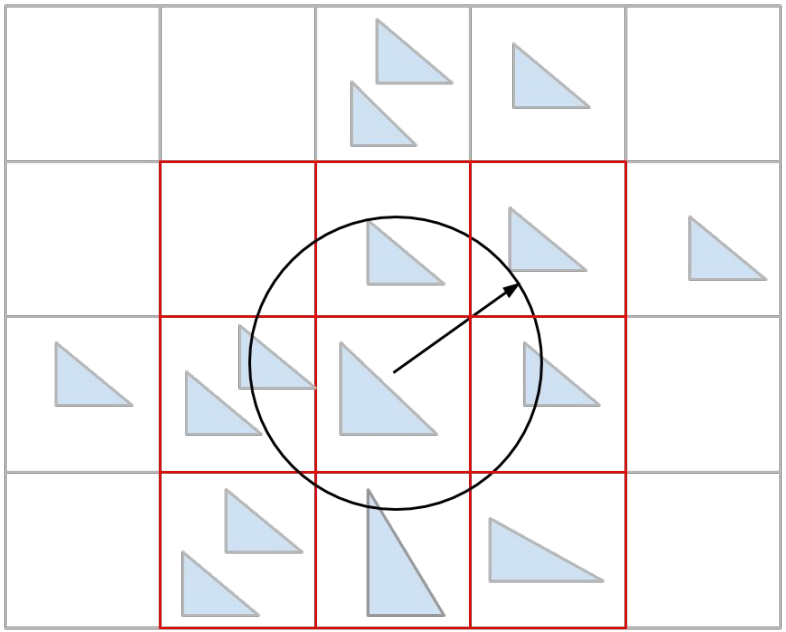
AO Raytracing:
check triangles in order, but stop when we compute a hit with #1



Optimization 2: Uniform Grid

- Raytracing: “**Where** is the first intersection for this ray, and **what** does it hit?”
- AO: “Does this ray hit anything **within a specified distance**?”

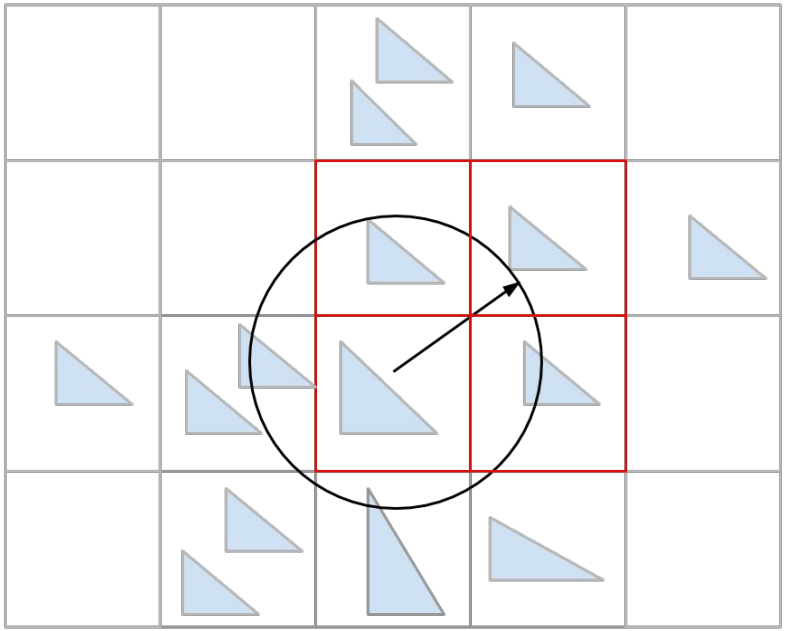
- Put all triangles in a **Uniform Grid**
- Grid cells are as wide as **specified distance**
- so we only have to check cells neighboring the ray origin



Optimization 3: Uniform Grid with cell culling

- Raytracing: “**Where** is the first intersection for this ray, and **what** does it hit?”
- AO: “Does this **ray** hit anything **within a specified distance**?”

- Ray has direction, so we know which cells it points away from
- Don’t have to check triangles in those cells



Results - Numbers

- Using the COLLADA duck, 4212 triangles
- Default Ambient Occlusion settings
- Node 4.4.5 on Windows 10 64 bit, i7-4890HQ with 16.0 GB RAM

Technique	**Seconds**
Naive ray-triangle testing	17.5044
Early Return	16.8362
Uniform Grid	4.8826
Uniform Grid + Cell Culling	2.7528
Grid + Cell Culling + Early Return	2.627