Tessellated Voxelization for Global Illumination using Voxel Cone Tracing

Sam Freed

Advisor: Dr. Christian Eckhardt
Committee Members: Dr. Maria Pantoja, Dr. Aaron Keen

June 2018

California Polytechnic State University, San Luis Obispo

Outline

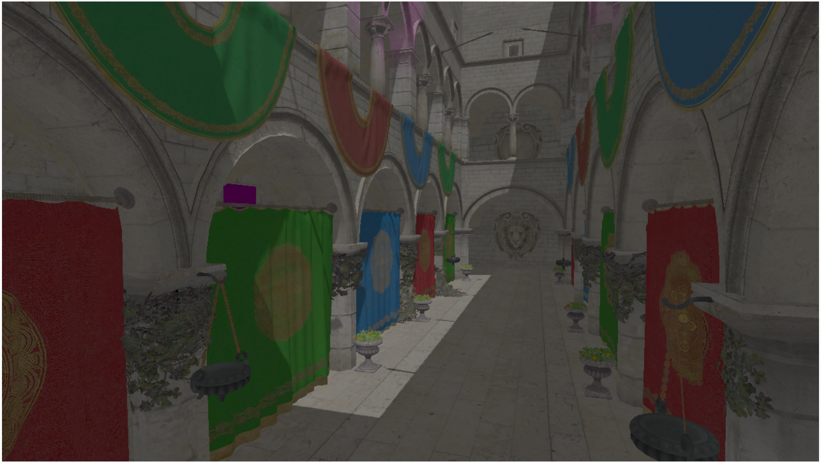
1. Introduction
2. Background
3. Implementation
4. Results and Conclusions

Computer Graphics

How can we simulate light in a virtual world?

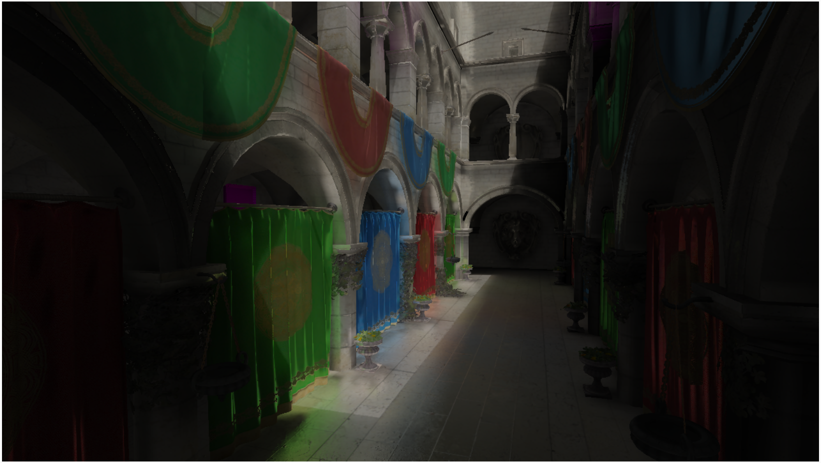
Physically accurate lighting is too complex for real-time—approximate!

Motivations



:(

Motivations



:)

Motivations

1. Real-time global illumination is difficult
2. Limited reference material

Contributions

- Open-source, cross-platform implementation of real-time global illumination
- Comparison of two different methods of scene voxelization
- Investigation into warped (nonuniform size) voxels

Goal

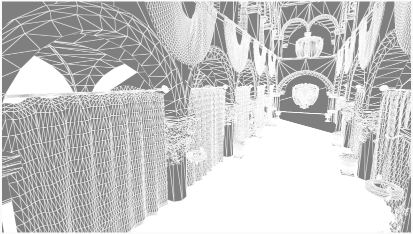
Given a virtual description of a scene, render an image.

Big Issues

1. How do we represent a scene? What information is required?
2. How is a 3D scene represented as a 2D image?
3. How do we render—how is the final pixel color computed?

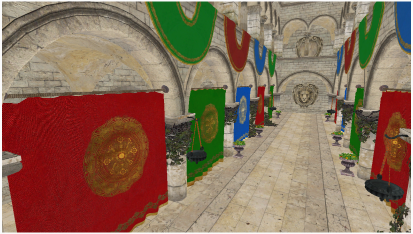
How do we represent a scene?

Geometry triangles, voxels

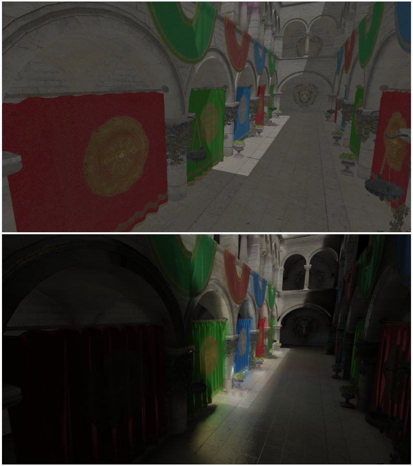


How do we represent a scene?

Materials colors and other properties



How do we represent a scene?



Lights positions, colors, etc.

How is a 3D scene represented as a 2D image?

Math!

All coordinates are transformed multiple times before ending up at their appropriate place on the screen.

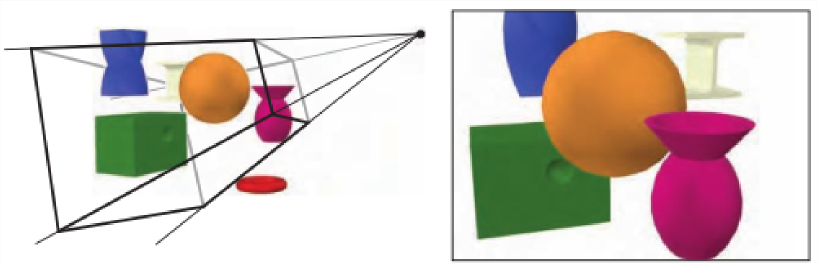
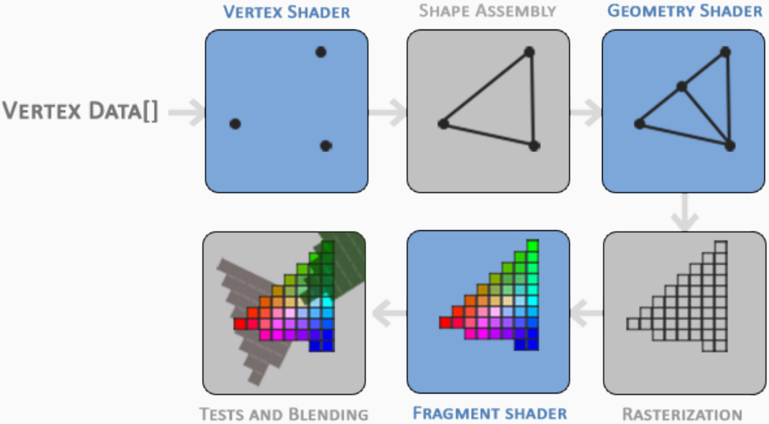


image from Real-Time Rendering

The Graphics Pipeline



How do we render?

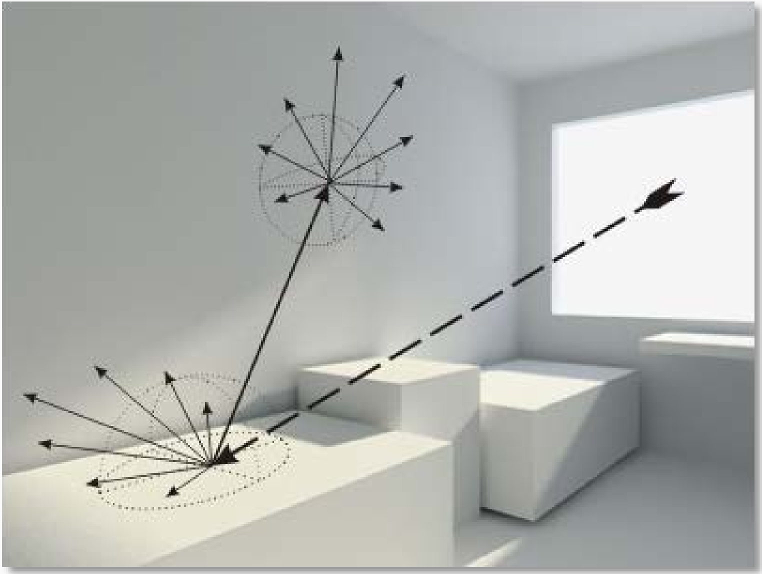
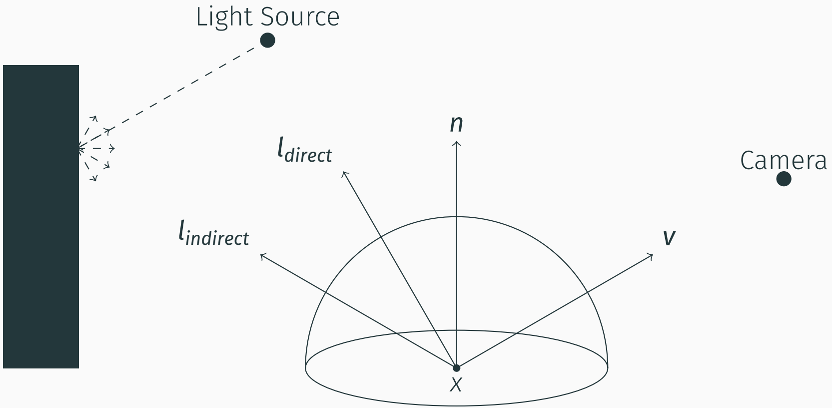


image from VCT and SVO for Real-time Global Illumination by Cyril Crassin

Light Theory



Real-Time Global Illumination

Various approaches to approximating indirect light.

Constant	Fixed fraction of ambient light
Partial	Ambient occlusion, screen space reflections
Static	Baked lighting, light probes
Dynamic	Reflective Shadowmaps, Light Propagation Volumes, Voxel Cone Tracing

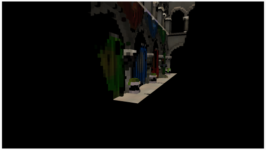
Most dynamic global illumination algorithms follow a few main steps:

1. Construct representation of the scene
2. Calculate indirect lighting information
3. Collect indirect lighting when rendering

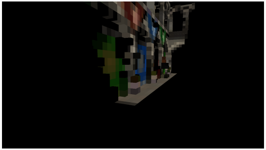
Overview of Renderer

(1) Voxelize

(2) Direct lighting



(3) Inject direct lighting into voxels



(4) Filter voxels

(5) Sample using voxel cone tracing

Each fragment corresponds to a voxel

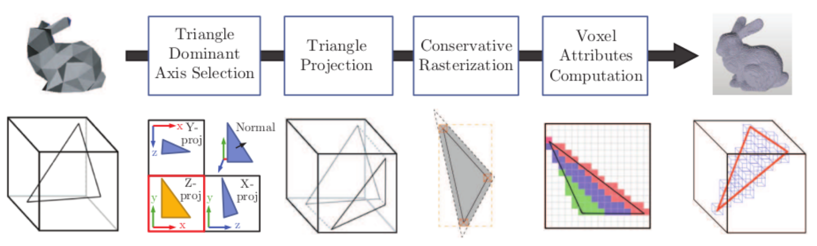
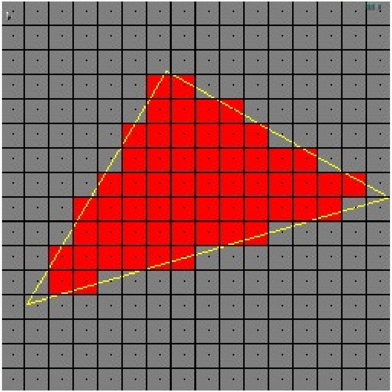
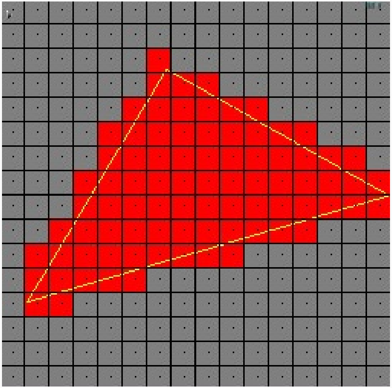


image from OpenGL Insights: Octree-Based Sparse Voxelization Using the GPU Hardware Rasterizer by Cyril Crassin and Simon Green

Conservative Rasterization



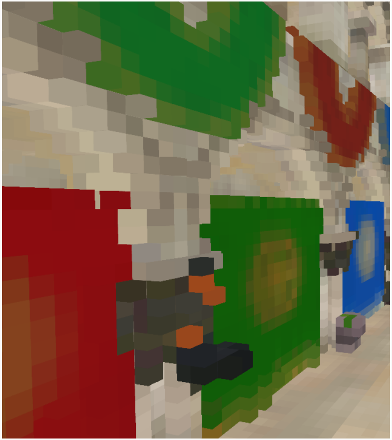
Off



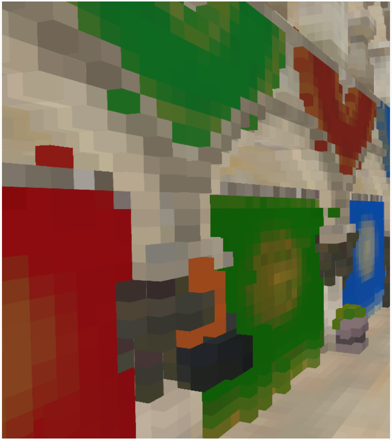
On

images by Jon Story from Don’t be conservative with Conservative Rasterization

Conservative Rasterization



Off



On

Voxelization with Tessellator

Idea: Generate a *vertex* for each voxel instead of a fragment

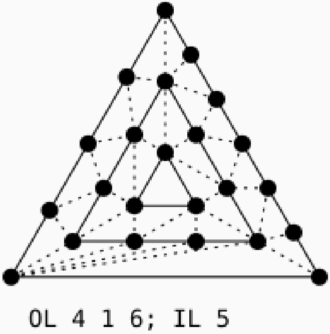
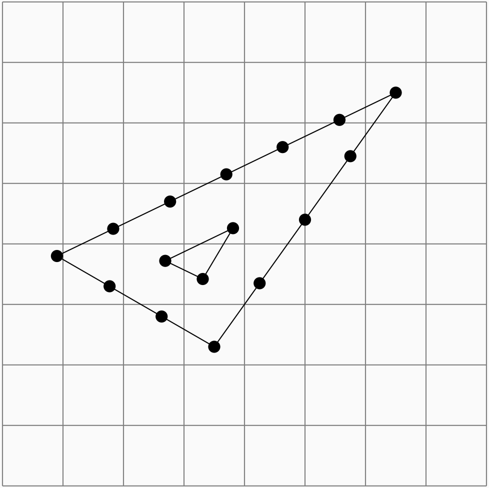


image from OpenGL Wiki

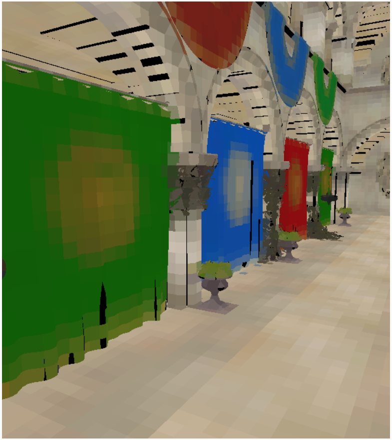
Determining Tessellation Levels

Outer levels determined from respective edge lengths

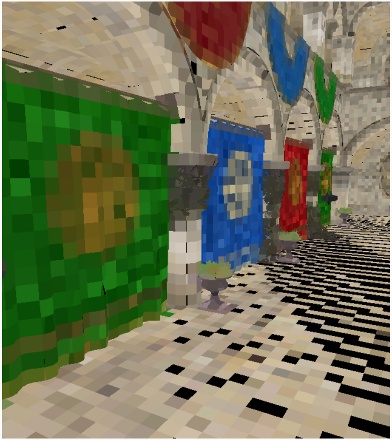
Inner level determined from maximum triangle altitude length



Voxelized Scene

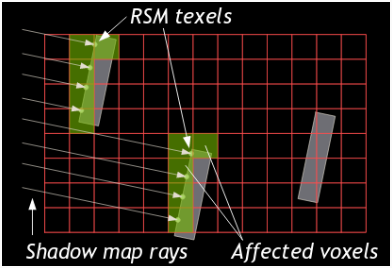


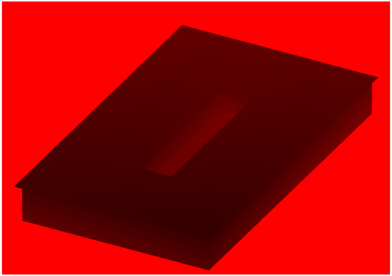
Rasterized voxels



Tessellated voxels

Use **shadowmap** to determine where direct light hits voxels

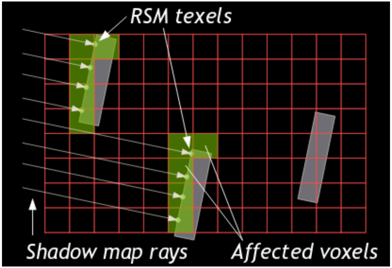


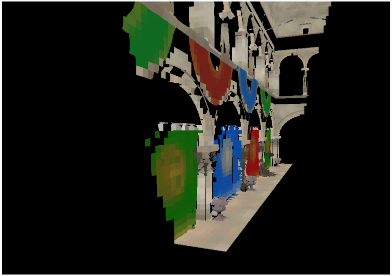


left image from Practical Real-time Voxel-based Global Illumination for Current GPUs by Alexey Panteleev

Radiance Injection

Use **shadowmap** to determine where direct light hits voxels

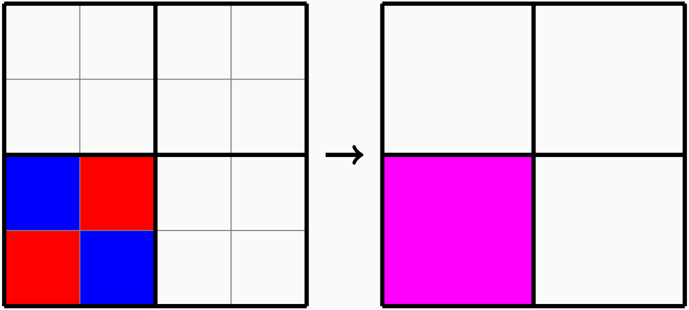




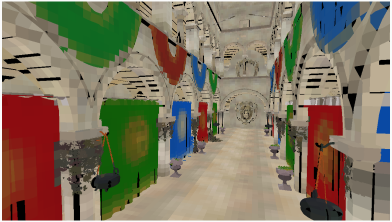
left image from Practical Real-time Voxel-based Global Illumination for Current GPUs by Alexey Panteleev

Radiance Filtering

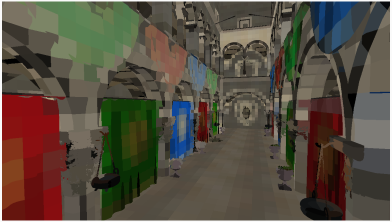
- Multiple **levels** (mipmaps)
- Each level is half the size of the previous
- Computing the next level is a 2x2x2 average



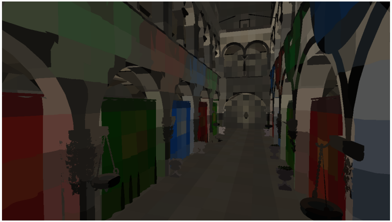
Radiance Filtering



Level 0



Level 1



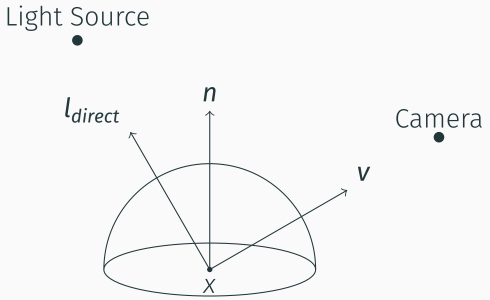
Level 2

Level 3

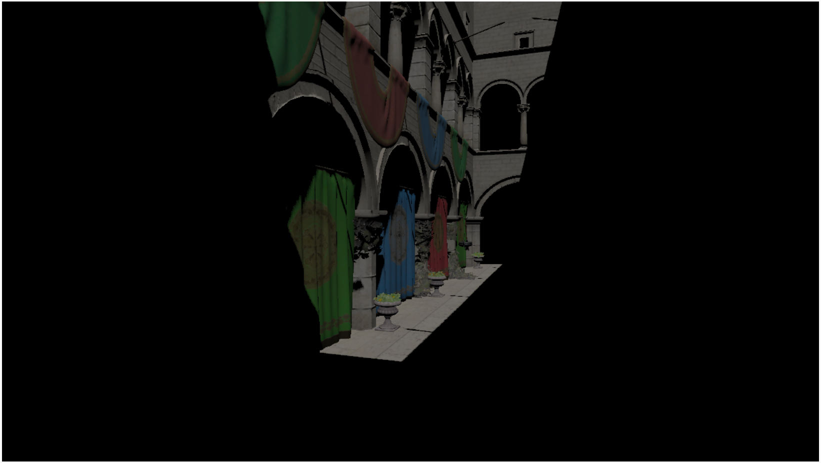
Mipmaps

Direct Lighting

```
color = 0
for each light in the scene do
    if not in shadow then
        color += computeLighting()
    end if
end for
```



Direct Lighting



Direct Lighting

Voxel Cone Tracing

1. Sample light and occlusion from the radiance texture along a particular direction
2. Adjust level of detail as we get farther from sample point

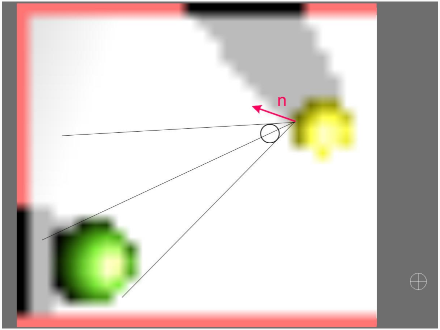


image from Christian Eckhardt

Indirect Lighting—Voxel Cone Tracing

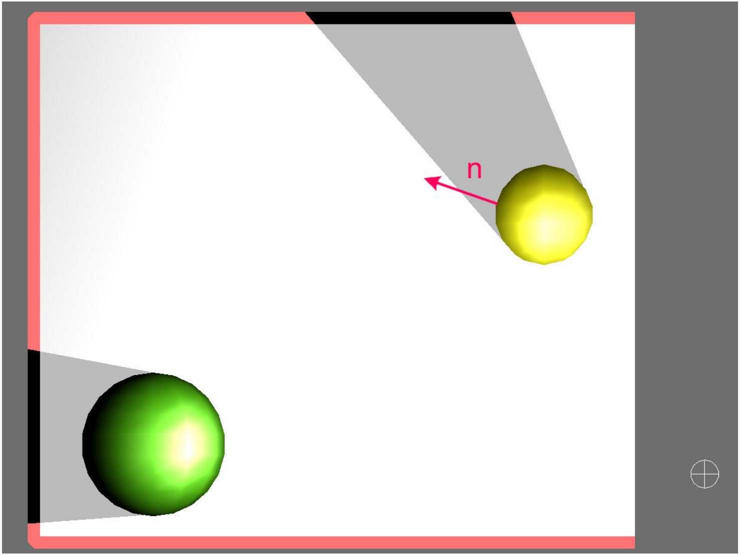


image from Christian Eckhardt

Indirect Lighting—Voxel Cone Tracing

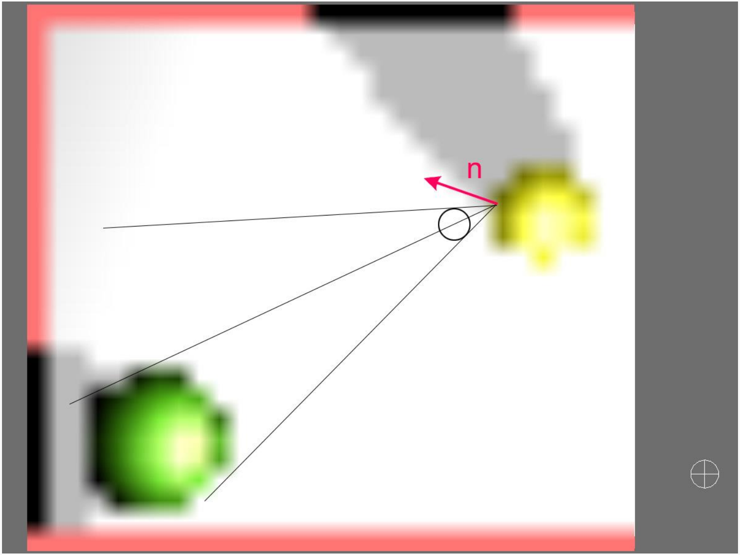


image from Christian Eckhardt

Indirect Lighting—Voxel Cone Tracing

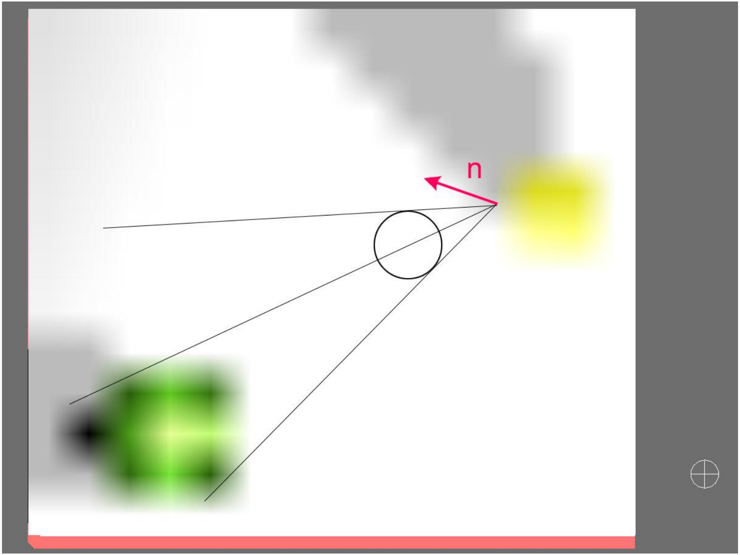


image from Christian Eckhardt

Indirect Lighting—Voxel Cone Tracing

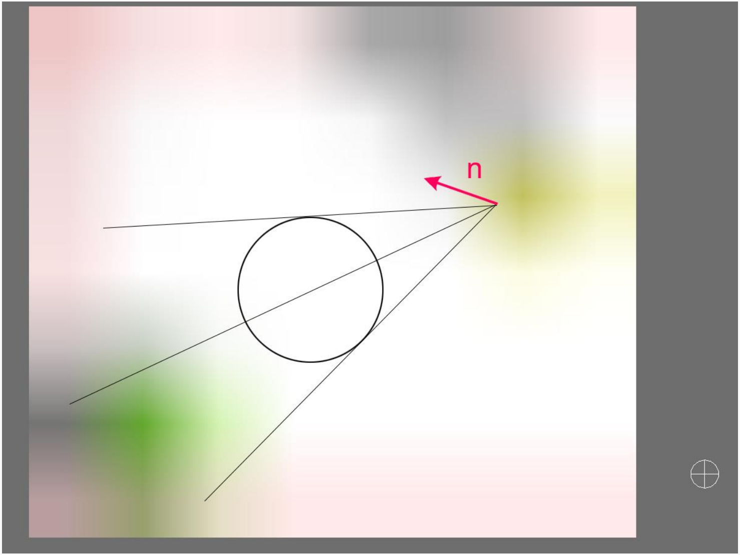


image from Christian Eckhardt

Indirect Lighting—Voxel Cone Tracing

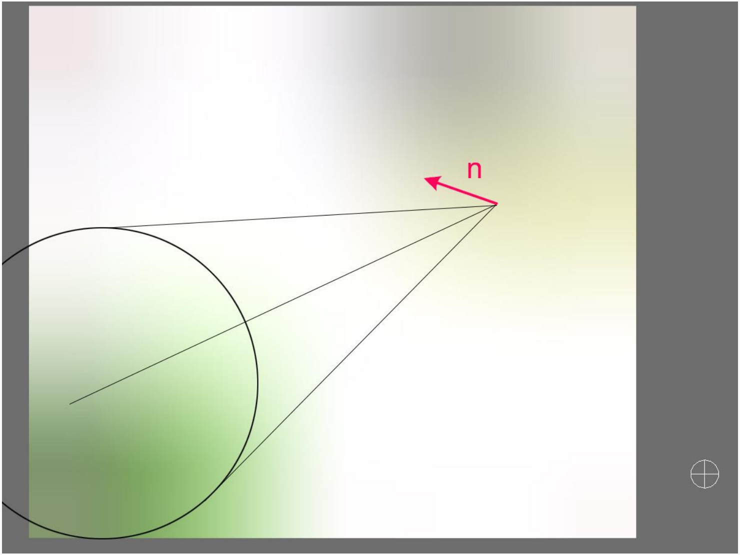


image from Christian Eckhardt

Indirect Lighting—Voxel Cone Tracing

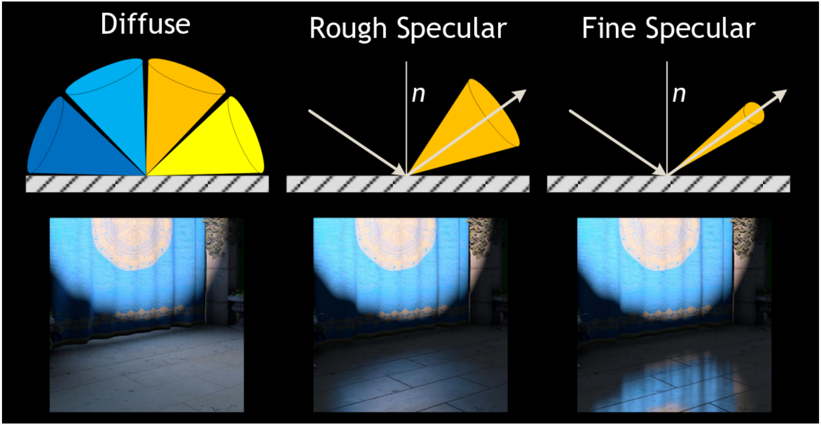
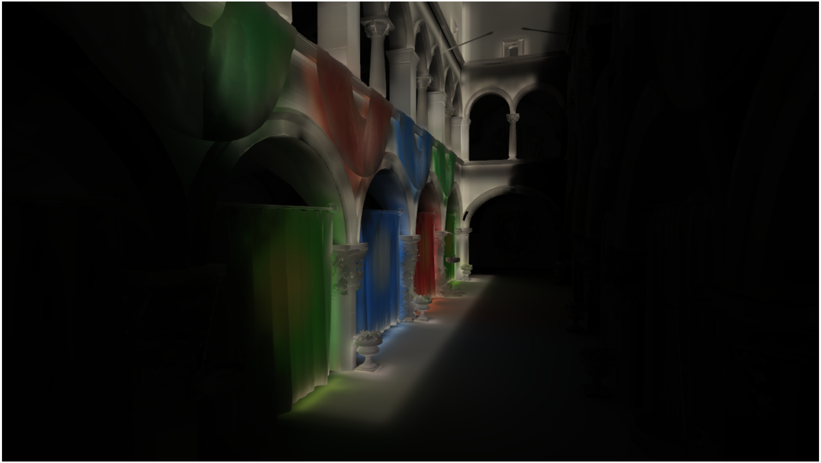


image from Practical Real-time Voxel-based Global Illumination for Current GPUs by Alexey Panteleev

Indirect Lighting



Diffuse Indirect (no occlusion)

Indirect Lighting

Specular Indirect (no occlusion)

Occlusion

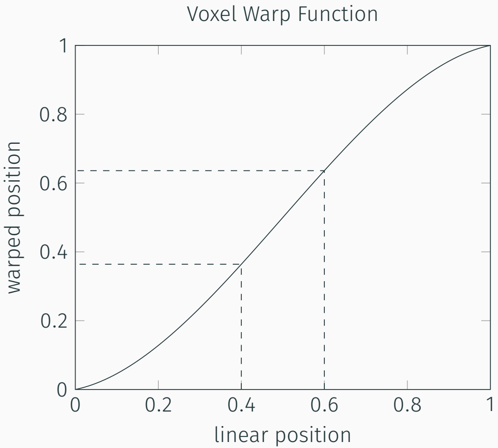
Voxel Warping

- Voxels are usually restricted to discrete sizes
- What if the size is not restricted?
    1. Vary with distance from camera
    2. Vary based on perspective

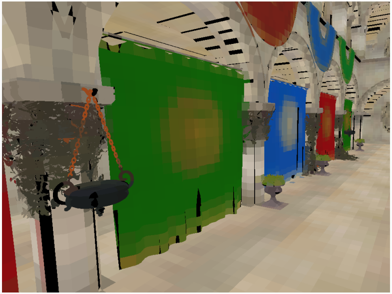
Vary with distance from camera

1. Find voxel position normalized to $[0, 1]$
2. Apply 'warping' function $w : [0, 1] \to [0, 1]$

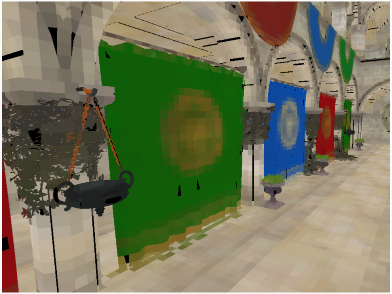
The camera is in the middle ($x = 0.5$)



Vary with distance from camera



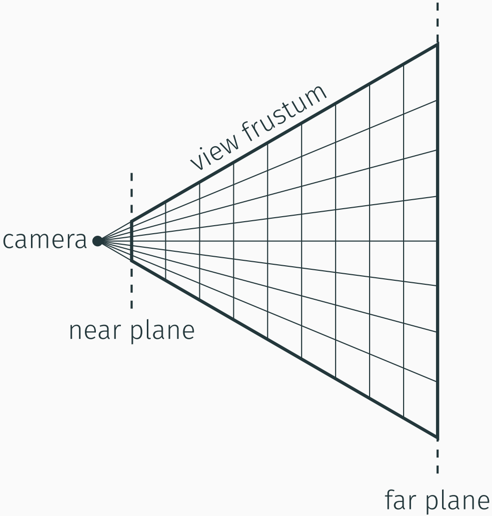
Without warping



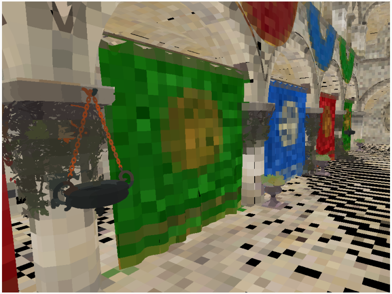
With warping

Vary based on perspective

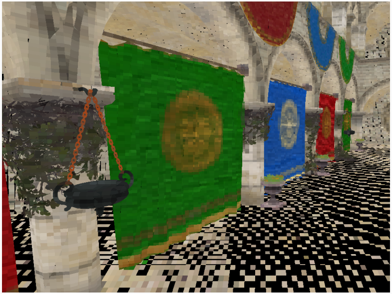
- Use perspective projection to determine voxel size
- Makes voxel size based on relative size in screen space



Vary based on perspective



Without warping



With warping

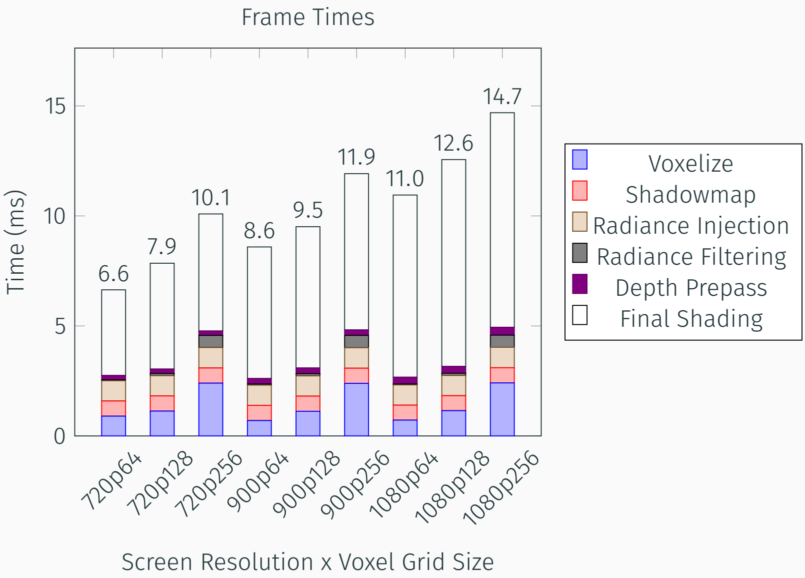
Results

64^3

128^3

256^3

Performance

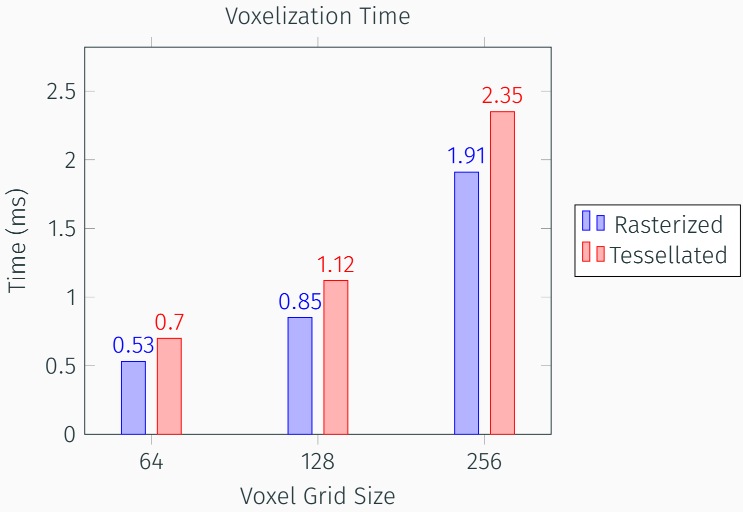


Rasterized vs. Tessellated Voxels

Rasterized voxels

Tessellated voxels

Rasterized vs. Tessellated Voxels

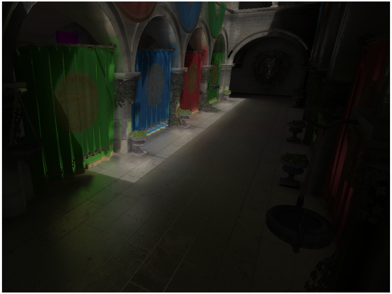


Rasterized vs. Tessellated Voxels

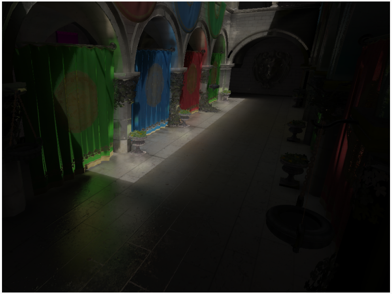
Summary

Rasterized:	limited by rasterizer, slightly faster
Tessellated:	limited by hardware, easier to implement

Voxel Warping

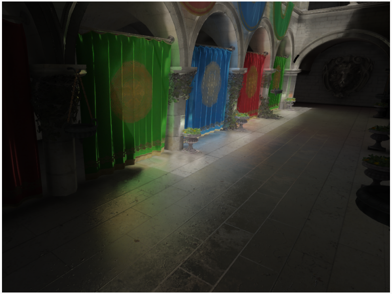


Without voxel warping

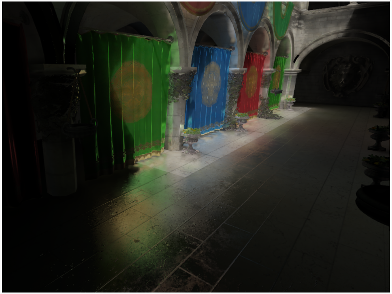


With voxel warping

Perspective Voxel Warping



Without voxel warping



With voxel warping

Related Work

How does it compare with other methods?

Important parts of global illumination algorithms:

1. Scene representation?
2. Light computation?
3. Light sampling?

Related Work—Reflective Shadowmaps

1. Scene representation? Reflective shadowmap (RSM)
2. Light computation? None, use color and normal from RSM
3. Light sampling? Sample nearby points in RSM

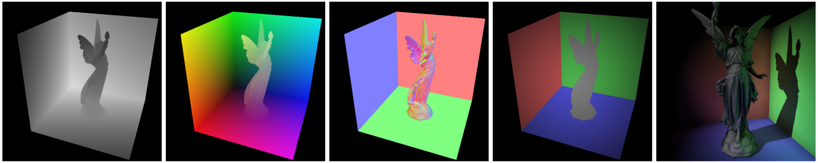


image from Reflective shadow maps by C. Dachsbacher and M. Stamminger

Related Work—Light Propagation Volumes

1. Scene representation? Voxel grid (incomplete)
2. Light computation? Iterative propagation
3. Light sampling? Texture lookup

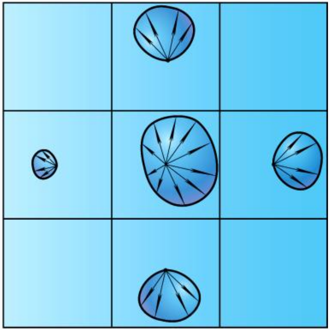
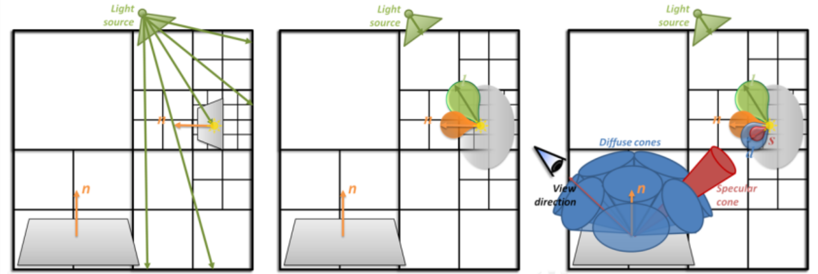


image from Light Propagation Volumes in CryEngine 3 by A. Kaplanyan

Related Work—Voxel Cone Tracing

1. Scene representation? Sparse voxel octree
2. Light computation? Mipmaps
3. Light sampling? Voxel cone tracing



Step 1: Render from light sources. Bake incoming radiance and light direction into the octree

Step 2: Filter irradiance values and light directions inside the octree

Step 3: Render from camera. Sample diffuse + specular BRDF components using voxel based cone tracing

image from Interactive Indirect Illumination Using Voxel Cone Tracing by C. Crassin et al.

Conclusion

- Implementation[1] of real-time global illumination using voxel cone tracing
- Implementation and comparison of two voxelization methods
- Investigation into warped voxels

[1]Find the source here: `github.com/sfreed141/vct`

Future Work

- Try to alleviate flickering with warped voxels
- Cascaded sparse 3D textures
- Spherical harmonics, anisotropic filtering, adaptive cone tracing quality, other miscellaneous optimizations

Thank you!

Questions?