

# 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

# Outline

## 1. Introduction

Motivations

Contributions

## 2. Background

## 3. Implementation

## 4. Results and Conclusions

# Introduction

## 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
  - Approximations need to be as fast and accurate as possible
  - Many steps and stuff to keep track of
2. Limited reference material
  - Not much public code
  - Demo or engine code

# Motivations

1. Real-time global illumination is difficult
  - Approximations need to be as fast and accurate as possible
  - Many steps and stuff to keep track of
2. Limited reference material
  - Not much public code
  - Demo or engine code
3. It's cool

# Contributions

- Open-source, cross-platform implementation of real-time global illumination (using voxel cone tracing)
- Comparison of two different methods of scene voxelization (rasterization vs. tessellation)
- Investigation into warped (nonuniform size) voxels

# Outline

1. Introduction

2. Background

Computer Graphics

Lighting

3. Implementation

4. Results and Conclusions

# Computer Graphics Primer

## Goal

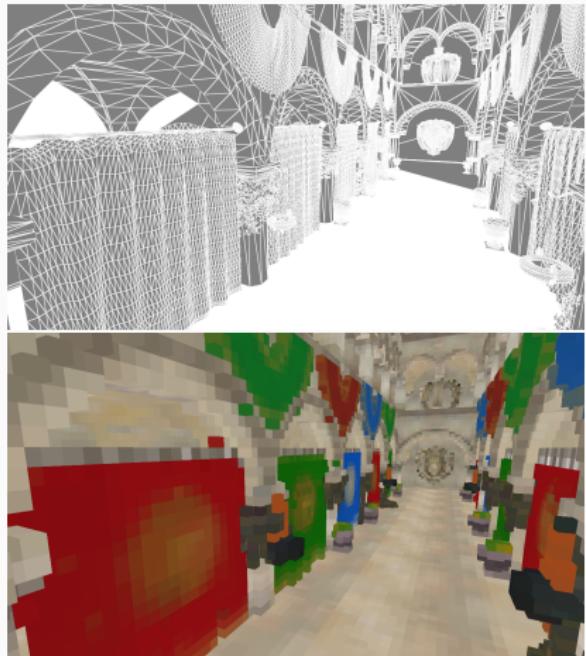
Given a virtual description of a scene, render an image.

## 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?

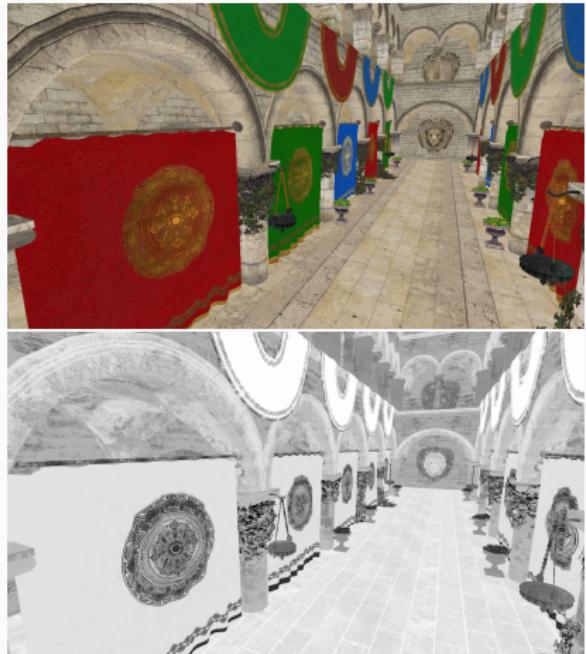
# How do we represent a scene?

- Geometry: triangles, voxels
- Materials: colors and other properties
- Lights: positions, colors, etc.



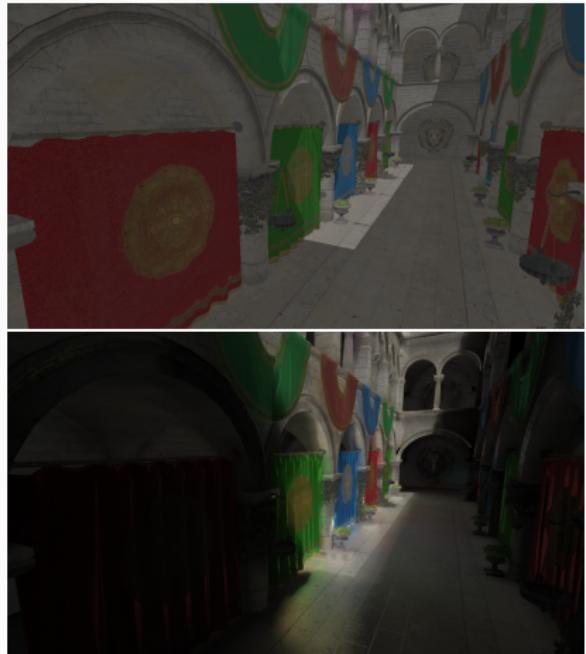
# How do we represent a scene?

- Geometry: triangles, voxels
- Materials: colors and other properties
- Lights: positions, colors, etc.



# How do we represent a scene?

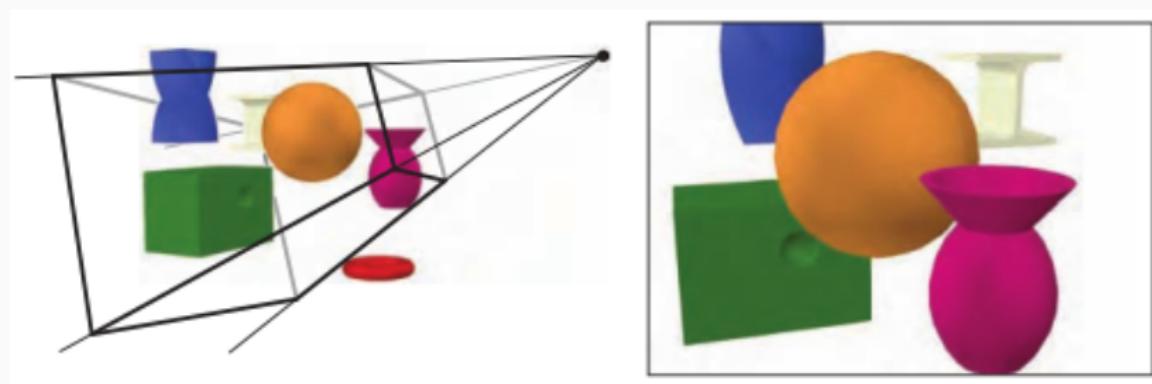
- Geometry: triangles, voxels
- Materials: colors and other properties
- Lights: positions, colors, etc.



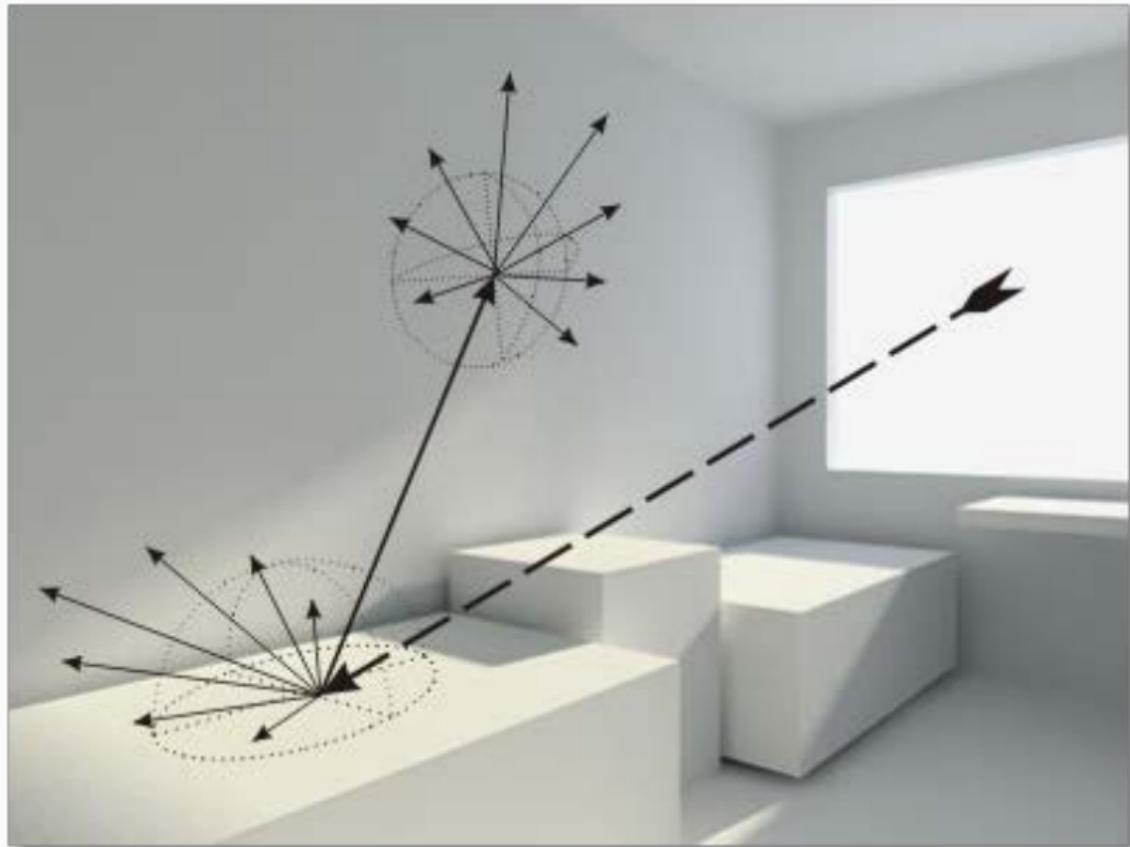
# How is a 3D scene represented as a 2D image?

Math!

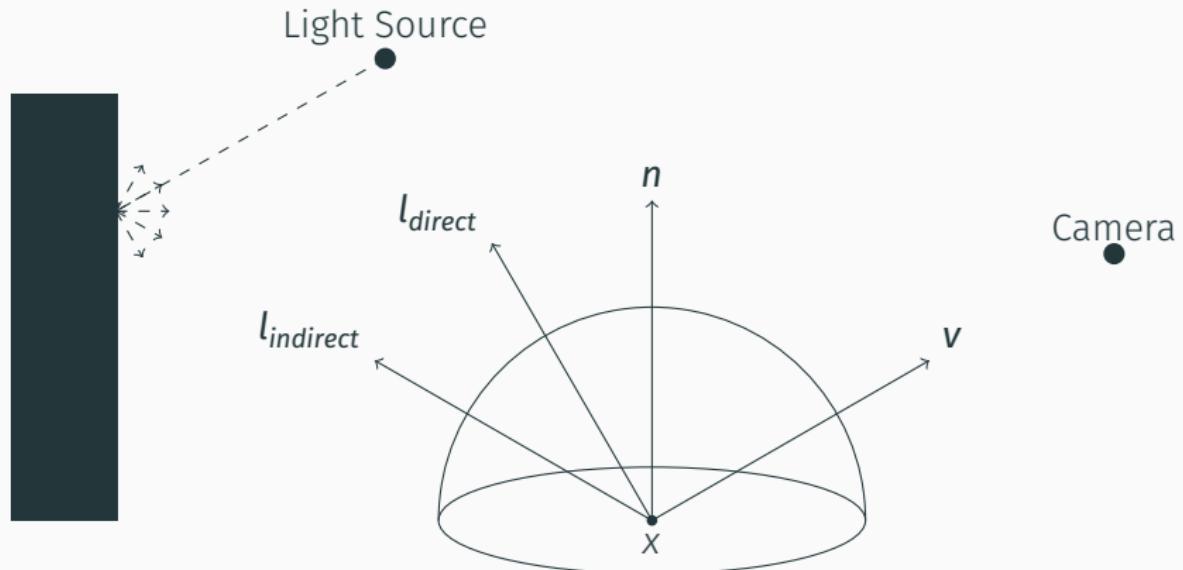
All coordinates are transformed multiple times before ending up at their appropriate place on the screen.



# How do we render?



# Light Theory



# Real-Time Global Illumination

Various approaches to approximating indirect light.

**Constant** Fixed fraction of ambient light

**Partial** Ambient occlusion, soft shadows, screen space reflections

**Static** Baked lighting, light probes

**Dynamic** Reflective Shadowmaps, Light Propagation Volumes, Voxel Cone Tracing

# Real-Time Global Illumination

Most dynamic global illumination algorithms follow a few main steps:

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

# What am I doing?

1. Comparing two approaches for scene voxelization
2. Investigating continuous voxel sizes

# Outline

1. Introduction

2. Background

3. Implementation

Voxelization

Radiance Injection and Filtering

Final Shading

Voxel Warping

4. Results and Conclusions

# Overview of Renderer

## Main Steps

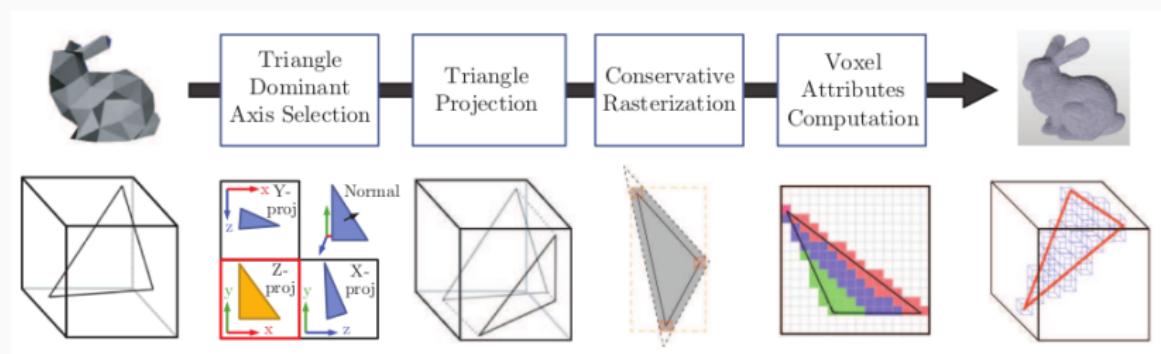
1. Setup (load scene, create textures, compile shaders)
2. Create Shadowmap
3. Voxelize Scene
4. Inject Radiance
5. Filter Radiance
6. Shading

## Important Data

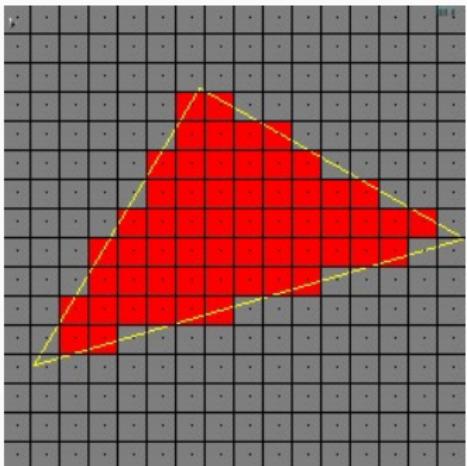
1. Scene (meshes, materials, lights)
2. Camera (position, direction)
3. Shadowmap
4. Voxel Textures (color + opacity, normals, radiance)

# Voxelization with Rasterizer

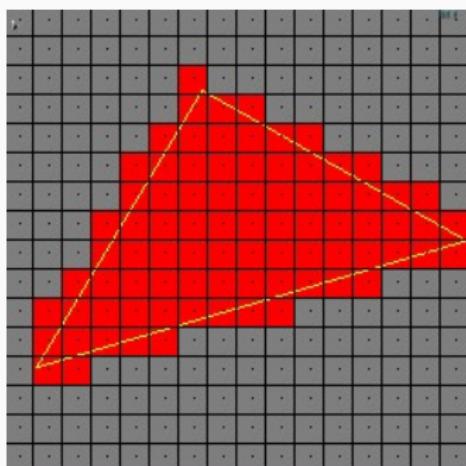
Each fragment corresponds to a voxel



# Conservative Rasterization



Off



On

# Conservative Rasterization



Without conservative rasterization

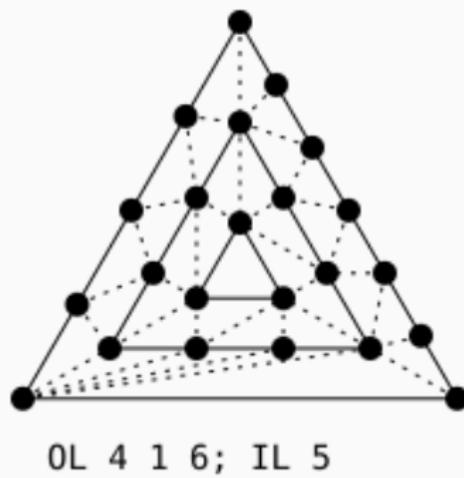
# Conservative Rasterization



With conservative rasterization

# Voxelization with Tessellator

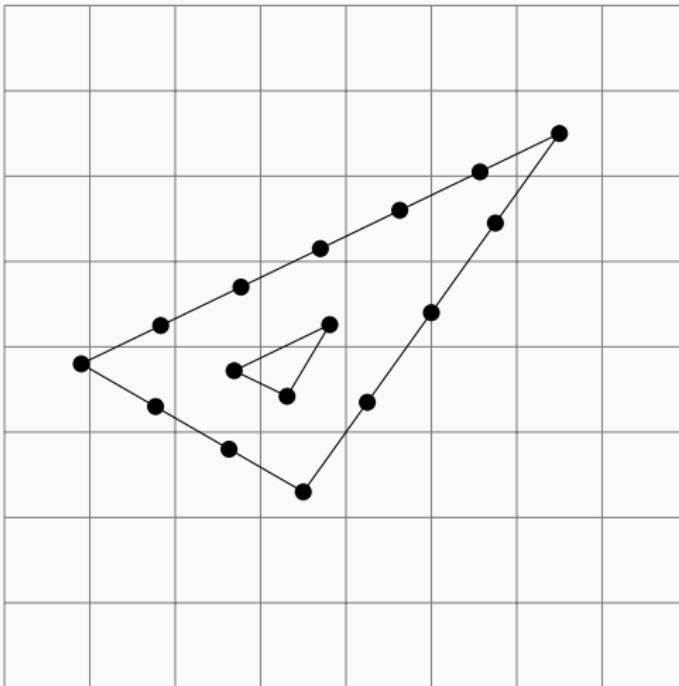
Idea: Generate a *vertex* for each voxel instead of a fragment



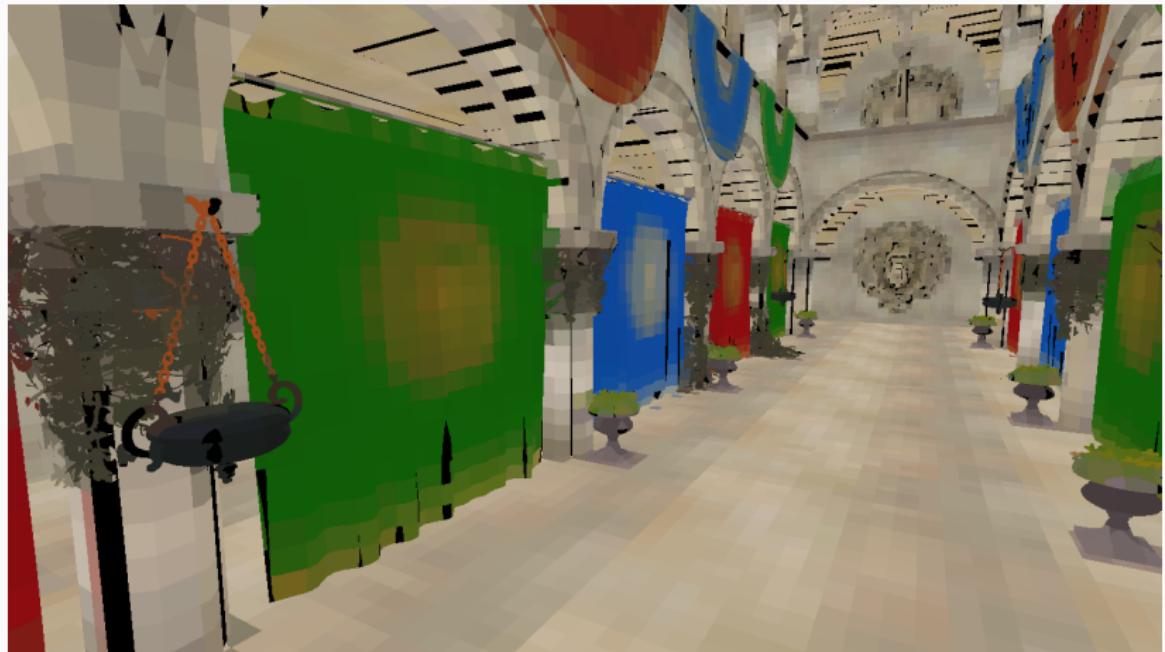
# Determining Tessellation Levels

Outer levels determined from respective edge lengths

Inner level determined from maximum triangle altitude length



# Voxelized Scene



Scene colored with rasterized voxels

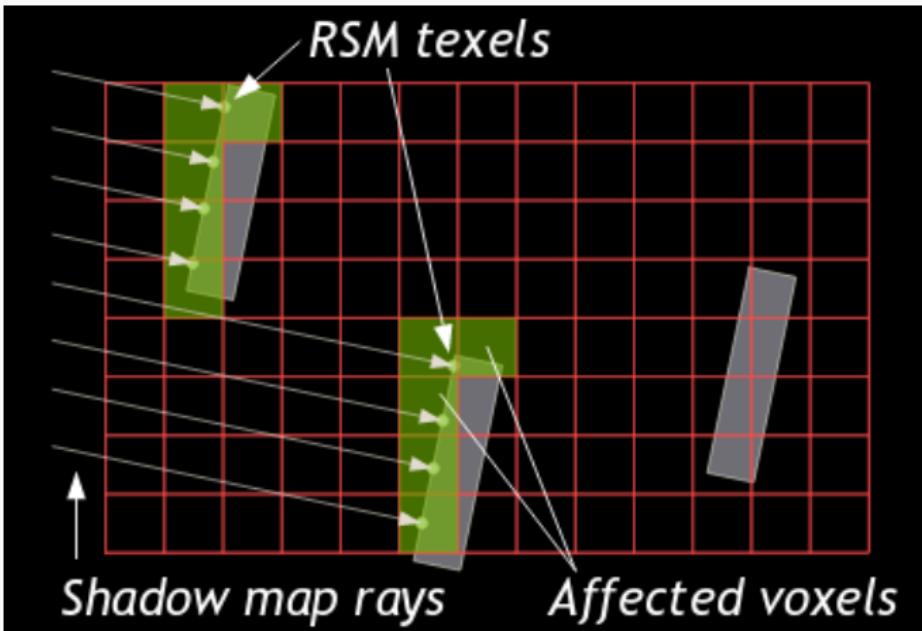
# Voxelized Scene



Scene colored with tessellated voxels

# Radiance Injection

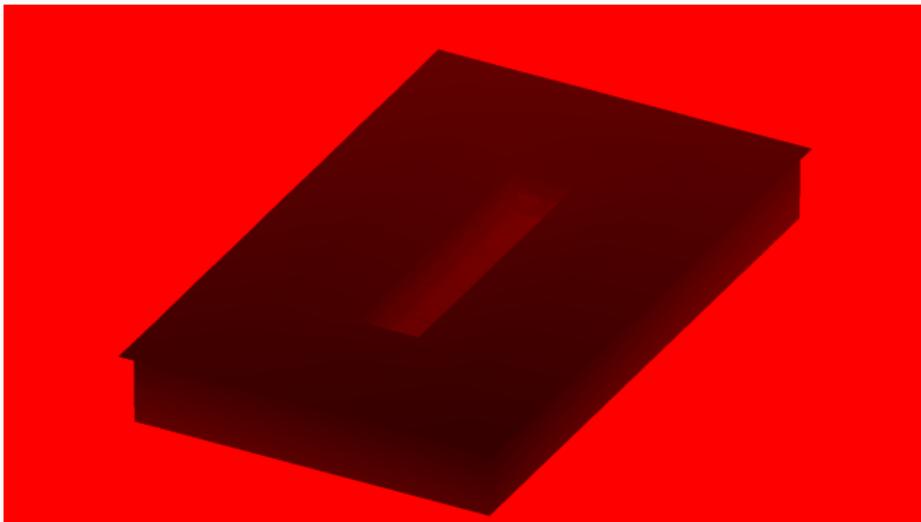
- Create virtual point lights for all geometry hit by the light.
- These lights approximate a single-bounce of indirect lighting.



# Radiance Injection—Shadow Mapping

To determine where the virtual point lights should be, we use a **shadowmap**.

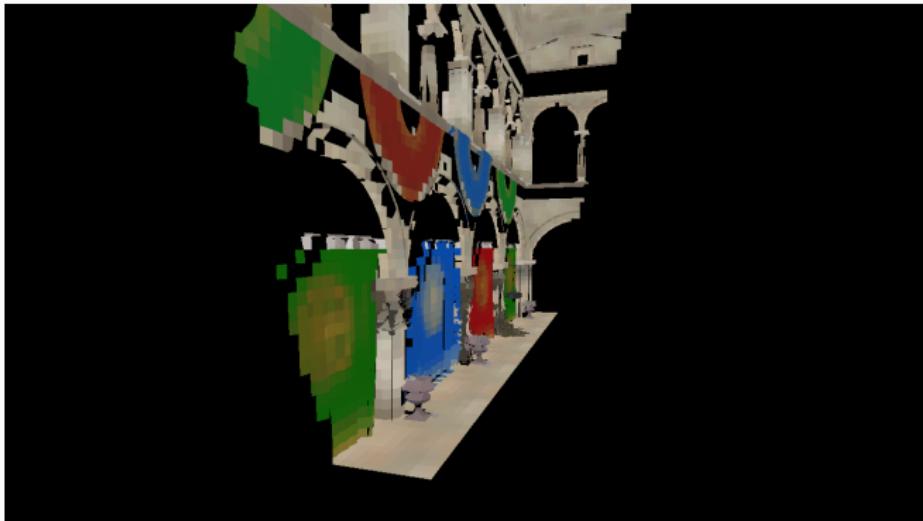
Render the scene from the *light's* point of view.



## Radiance Injection—Injecting VPLs

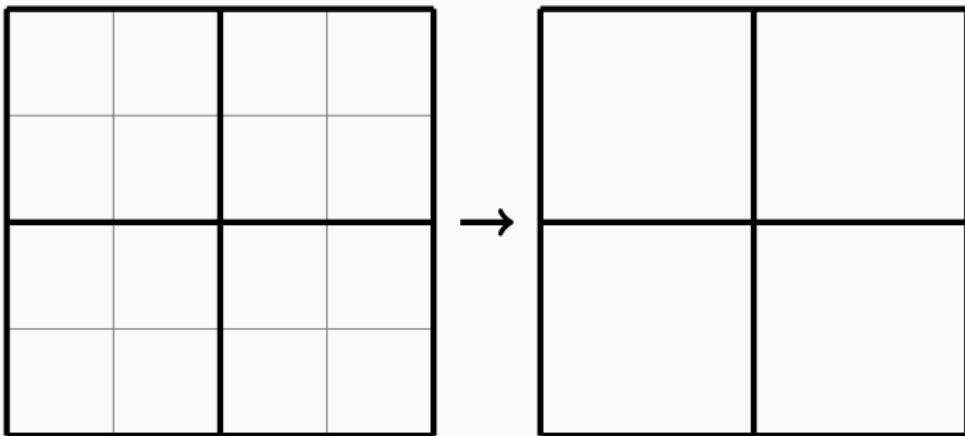
For each pixel in the shadowmap, find it's voxel index and insert the corresponding color into the radiance texture.

Using the light matrix and stored depth value, we compute the point's world space position.

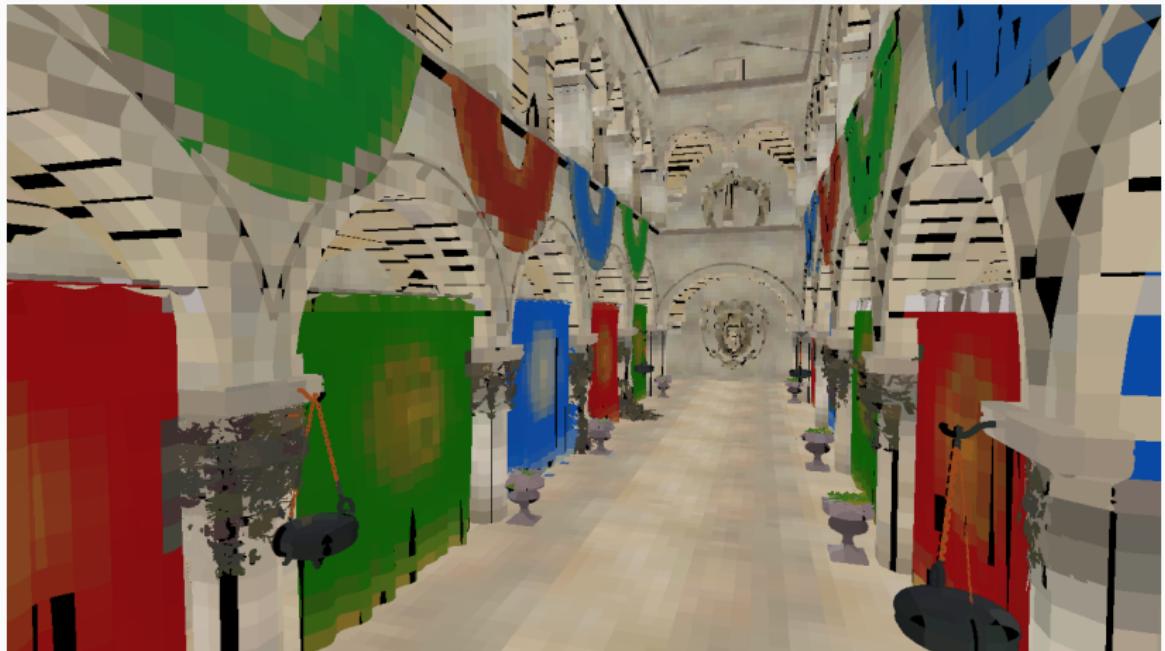


# Radiance Filtering

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



# Radiance Filtering



Mipmap level 0

# Radiance Filtering



Mipmap level 1

# Radiance Filtering



Mipmap level 2

# Radiance Filtering



Mipmap level 3

# Final Shading

**Direct Lighting** sum light contributions using Cook-Torrance shading model

**Indirect Lighting** voxel cone tracing

**Post Processing** tone mapping and gamma correction

# Direct Lighting

---

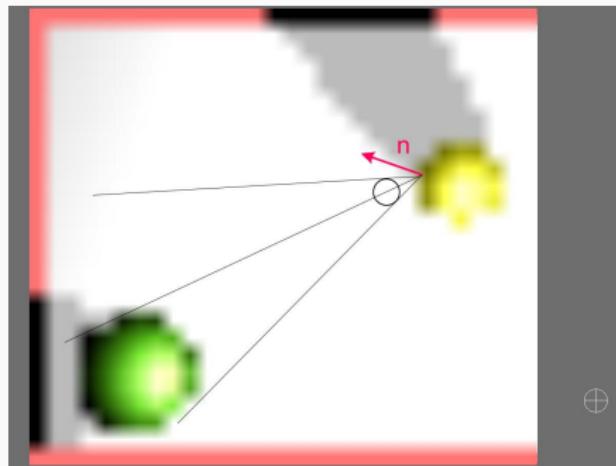
```
color = 0
for each light in the scene do
    if not in shadow then
        color += computeLighting()
    end if
end for
```

---

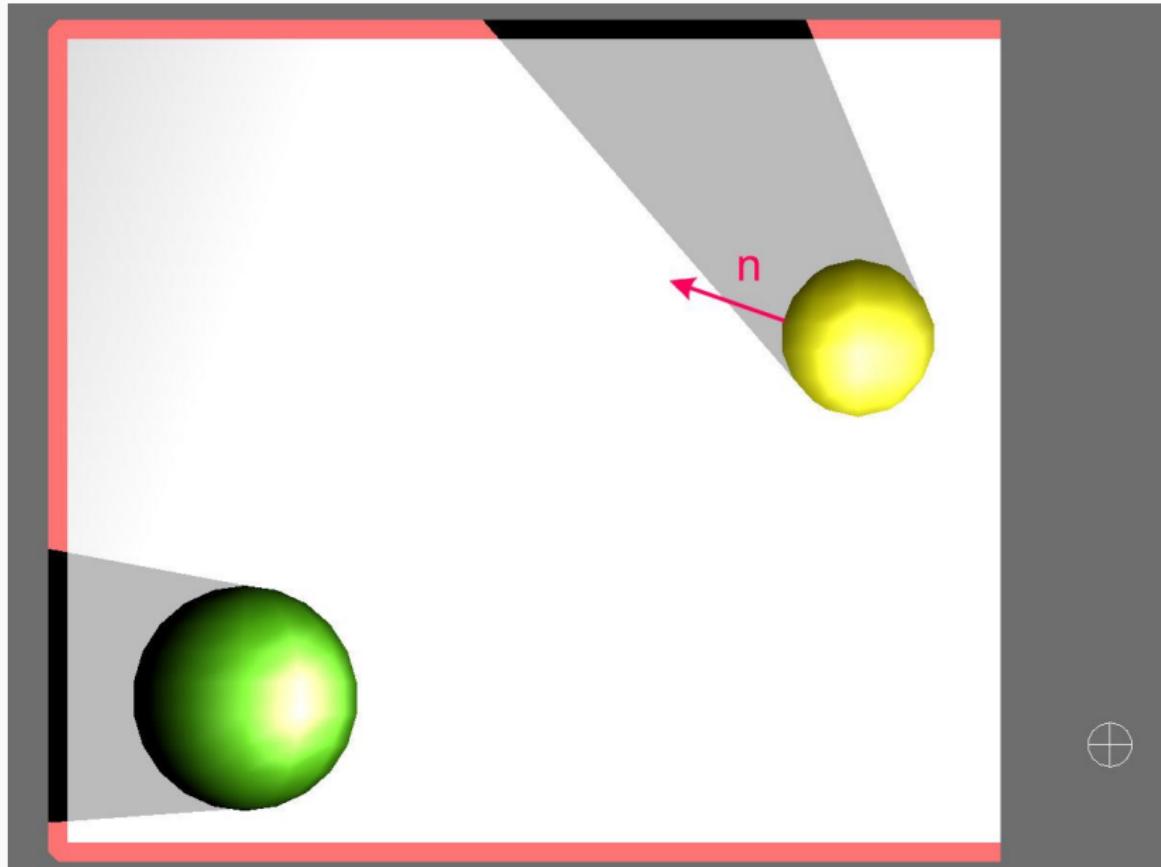
# Indirect Lighting

## Voxel Cone Tracing

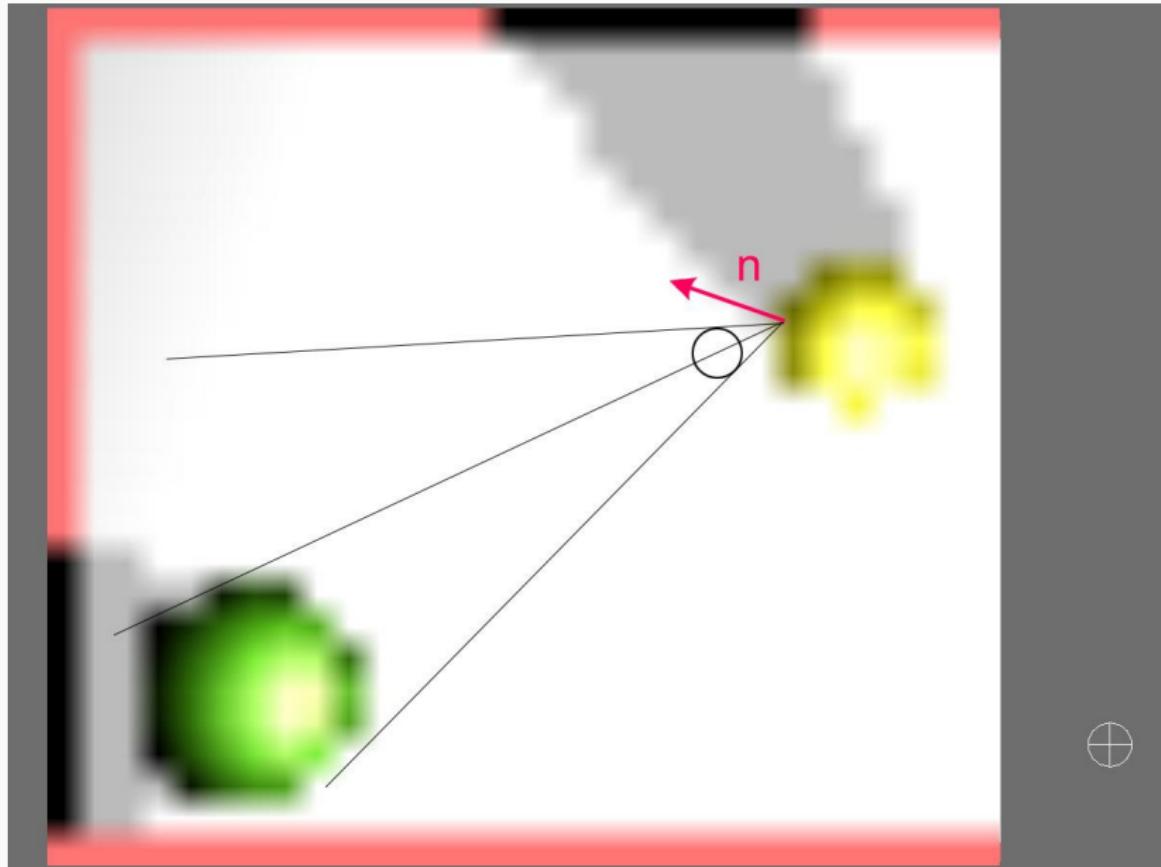
1. Sample light from the radiance texture along a particular direction
2. Adjust level of detail as we get farther from sample point



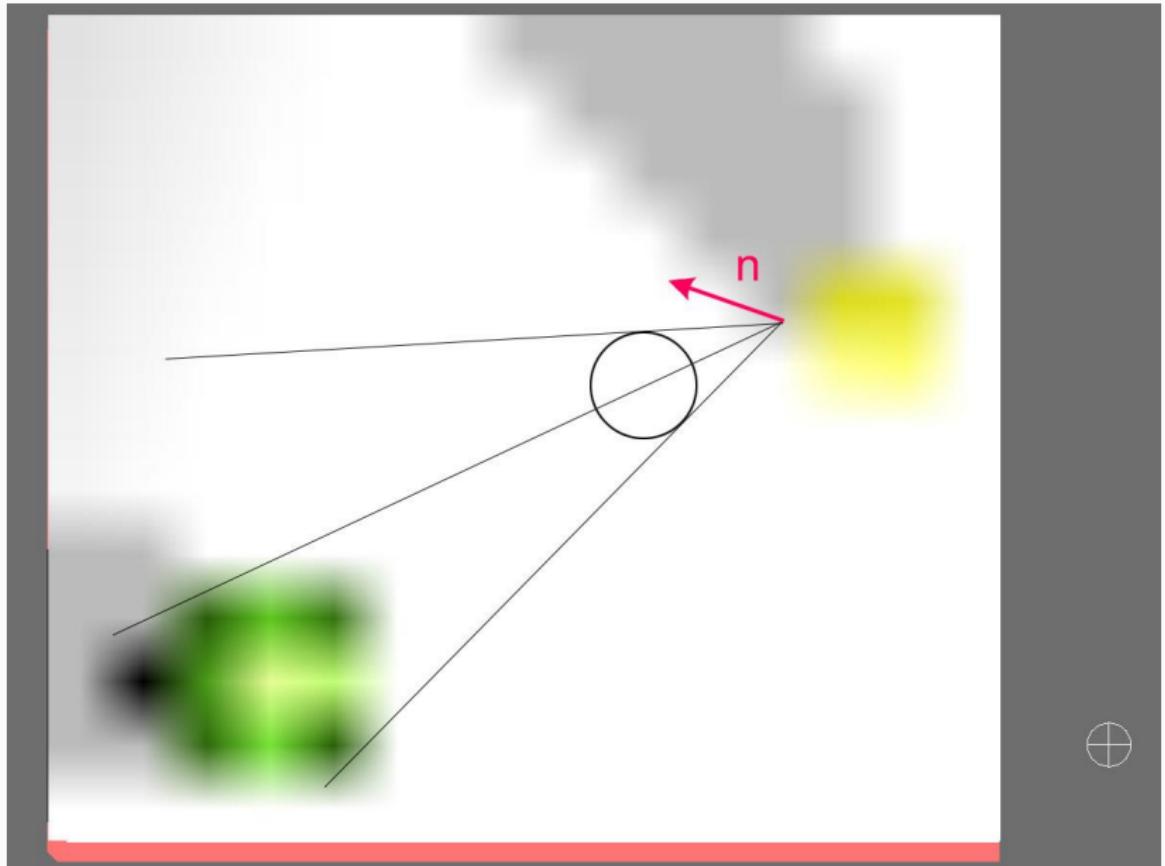
# Voxel Cone Tracing



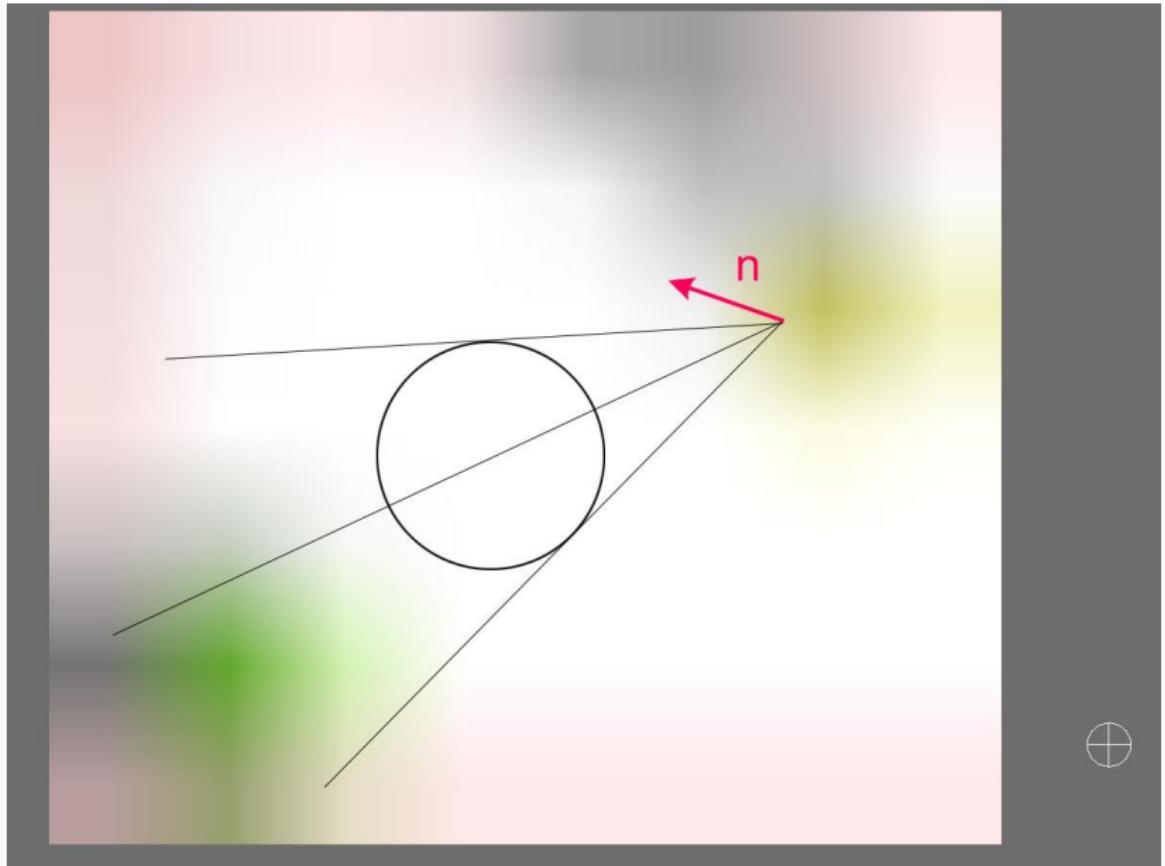
# Voxel Cone Tracing



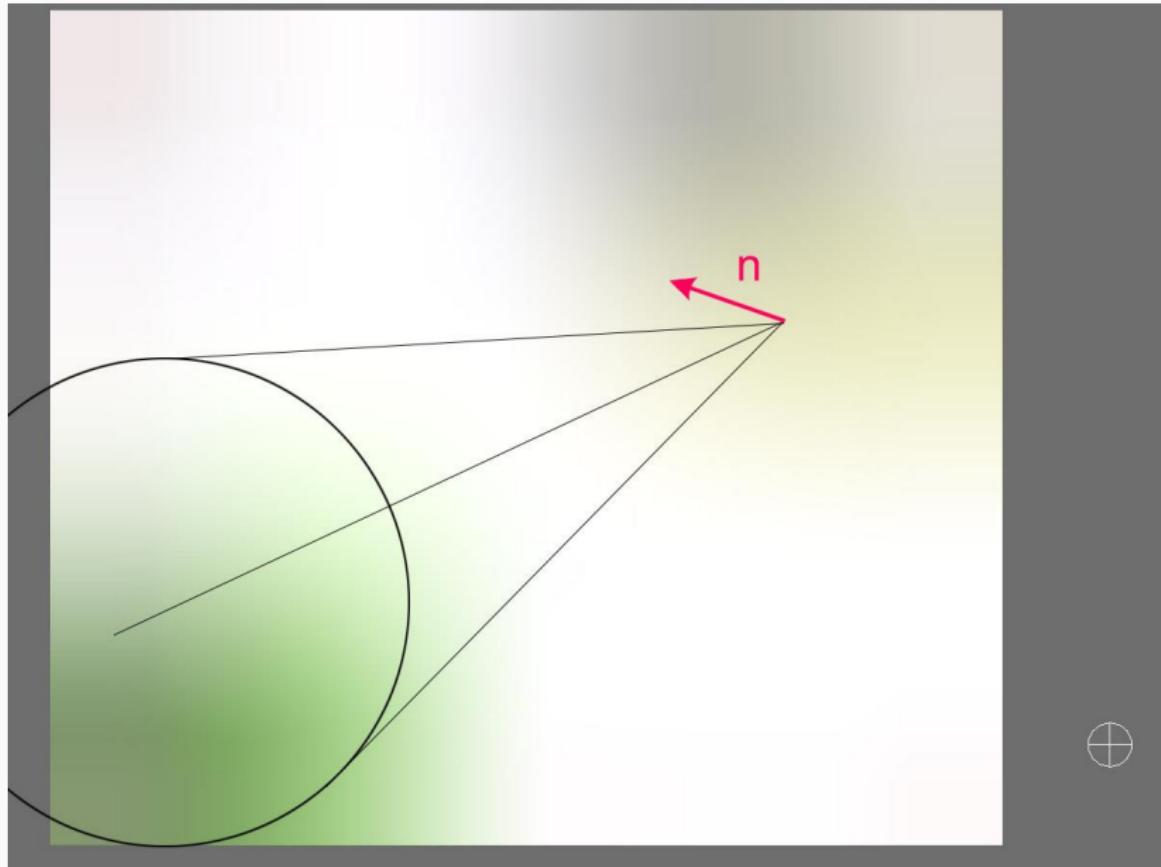
# Voxel Cone Tracing



# Voxel Cone Tracing



# Voxel Cone Tracing

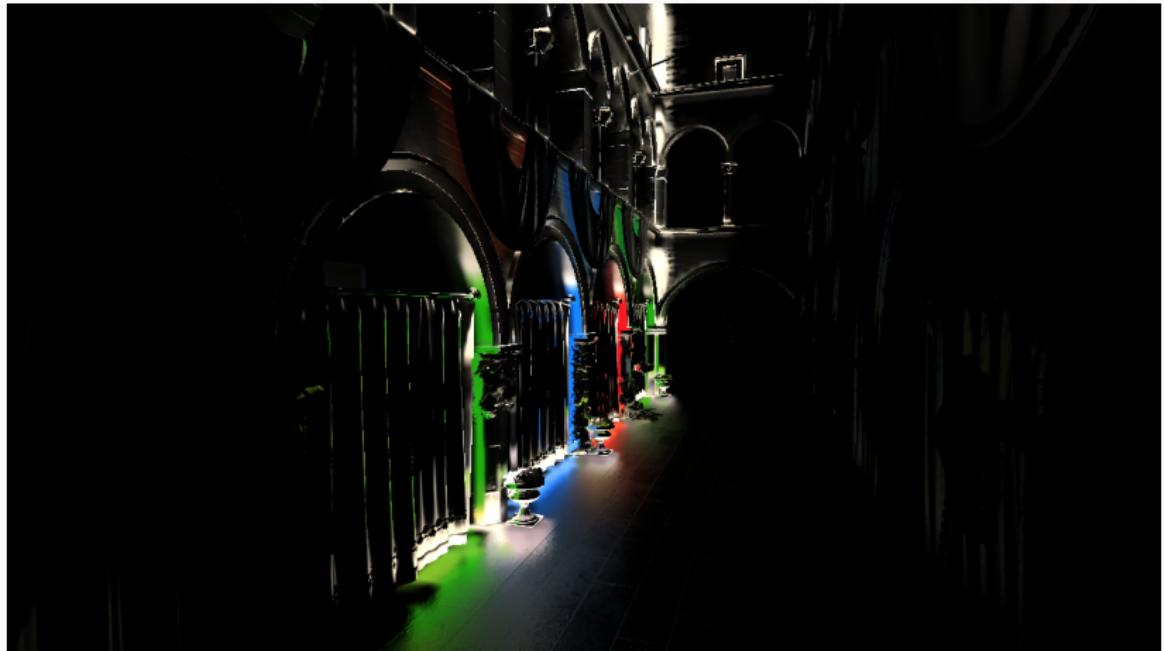


# Indirect Lighting



Diffuse Indirect (no occlusion)

# Indirect Lighting



Specular Indirect (no occlusion)

# Indirect Lighting



Occlusion

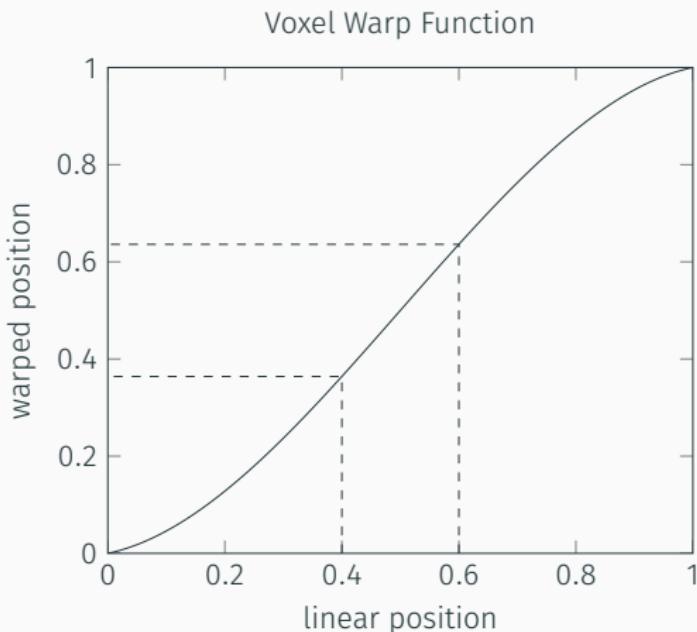
# Voxel Warping

- 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

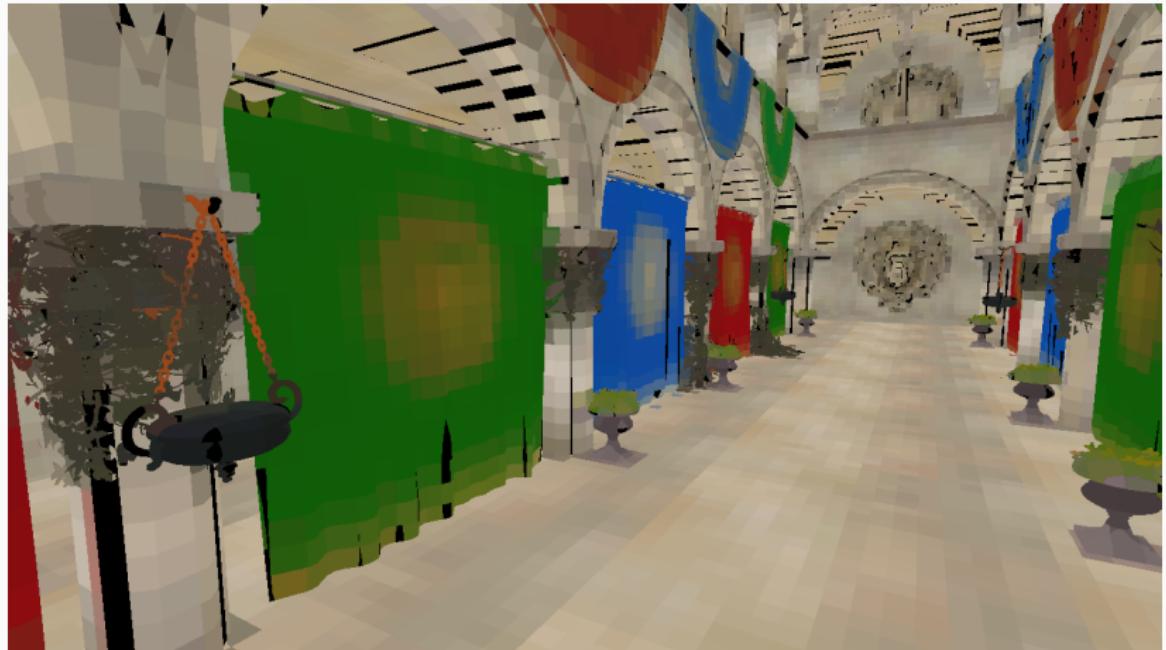
## Vary with distance from camera

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

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

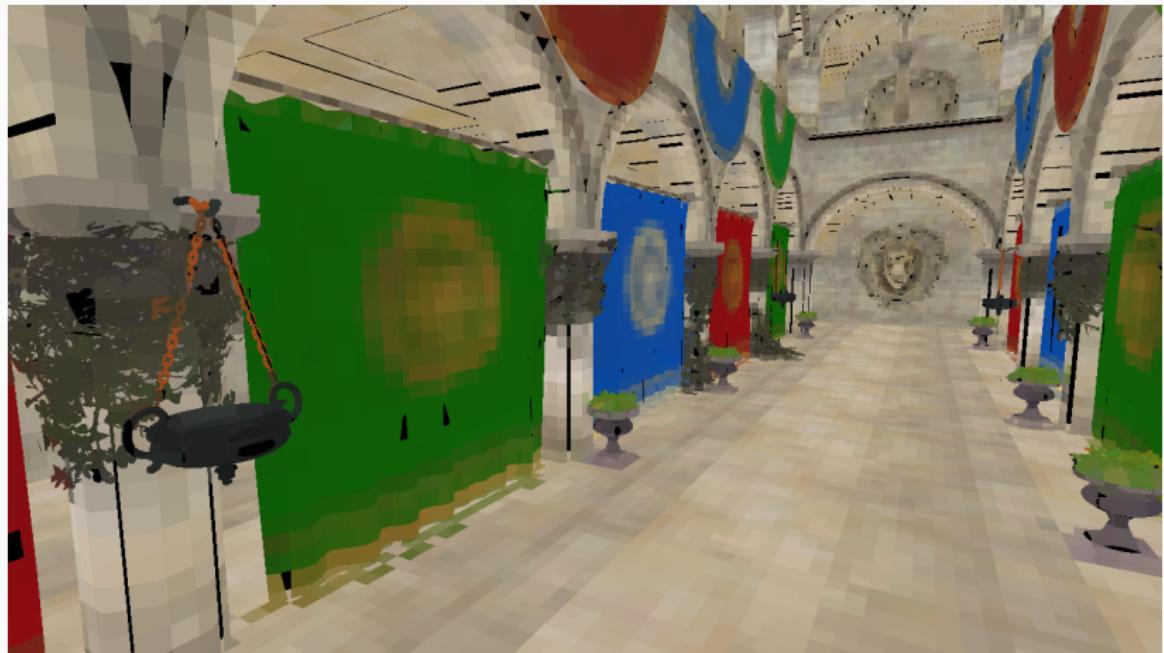


Vary with distance from camera



Without warping

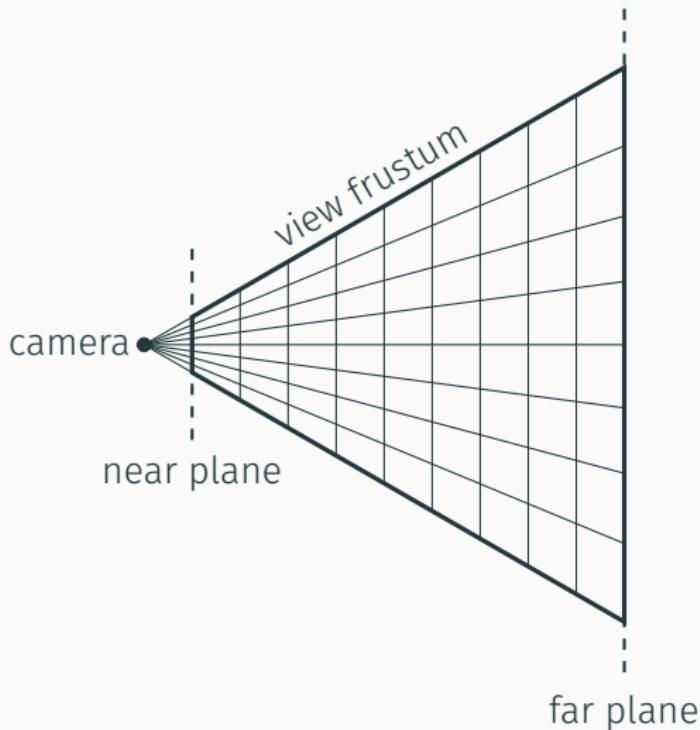
Vary with distance from camera



With warping

## Vary based on perspective

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

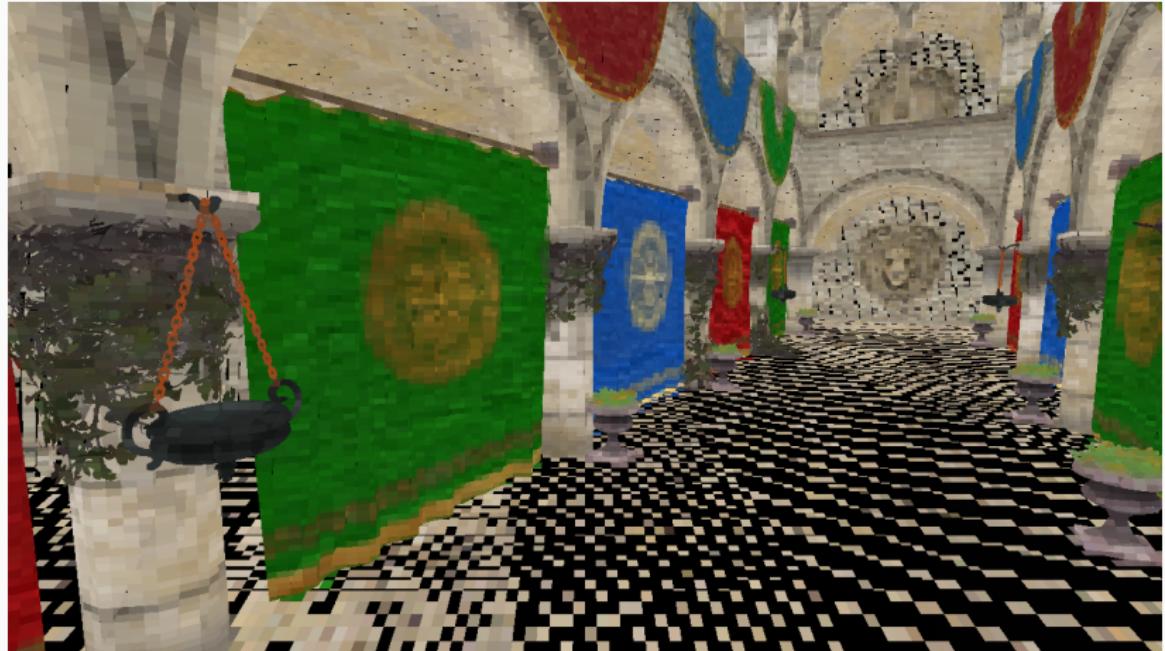


Vary based on perspective



Without warping

Vary based on perspective



With warping

# Outline

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

# Performance



$64^3$  voxel grid

# Performance



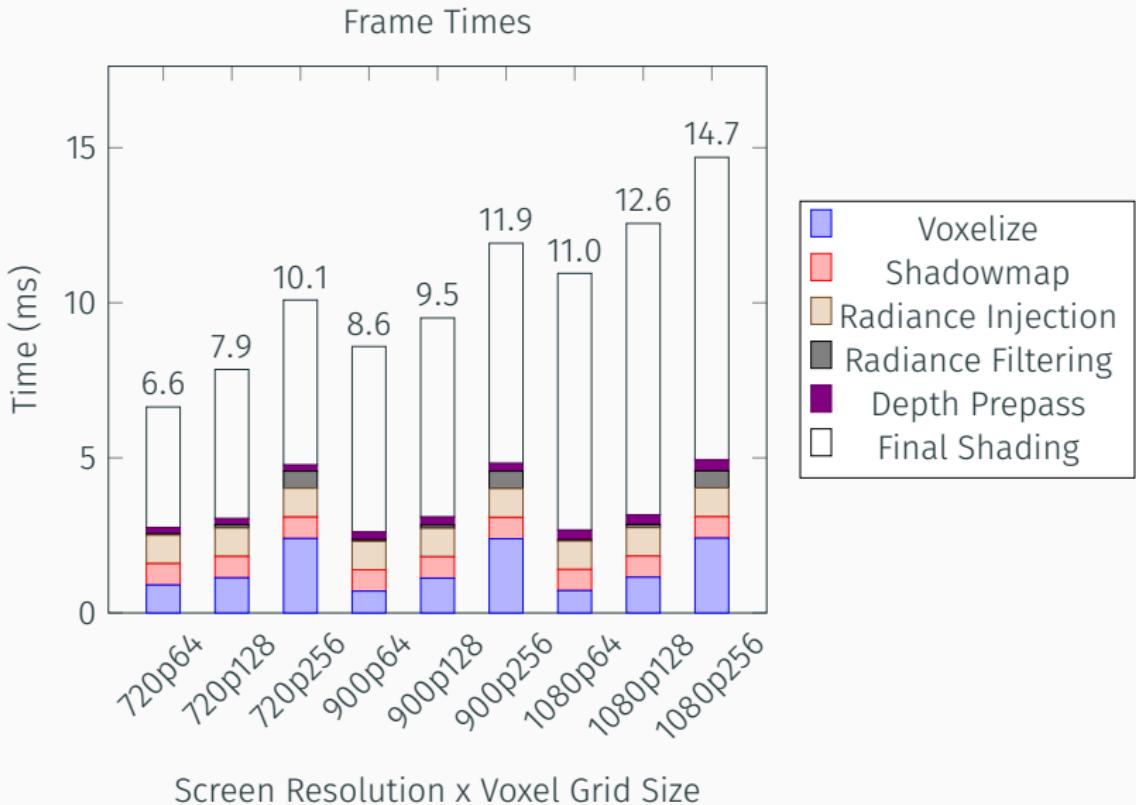
$128^3$  voxel grid

# Performance



$256^3$  voxel grid

# Performance



# Rasterized vs. Tessellated Voxels



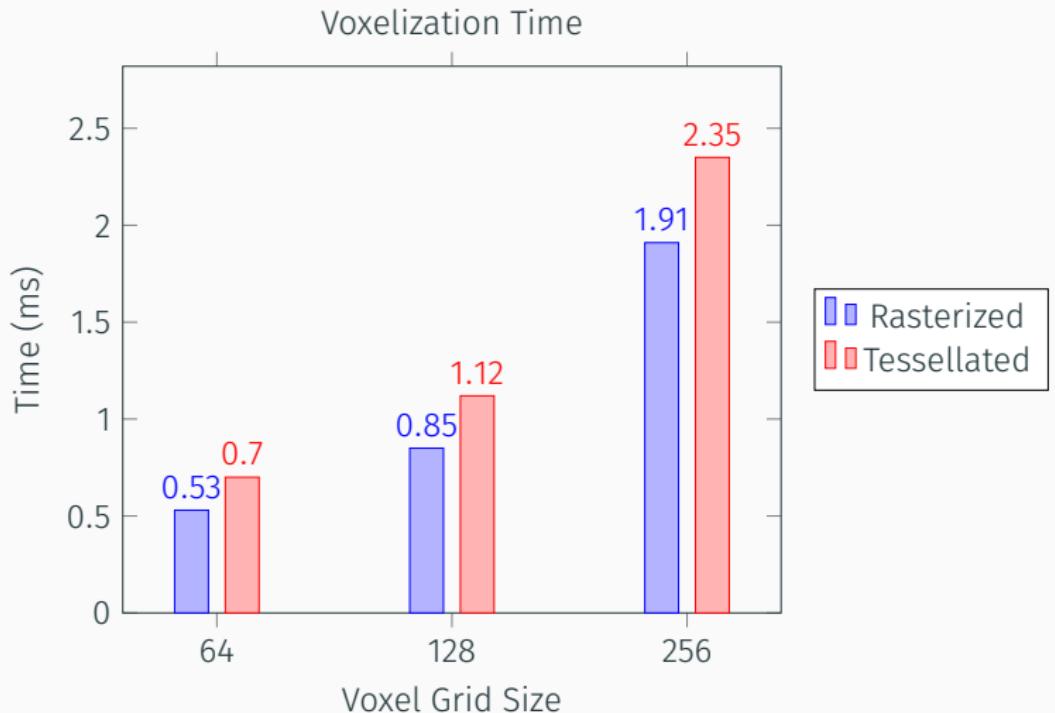
Rasterized voxels

# Rasterized vs. Tessellated Voxels



Tessellated voxels

# Rasterized vs. Tessellated Voxels



# Voxel Warping



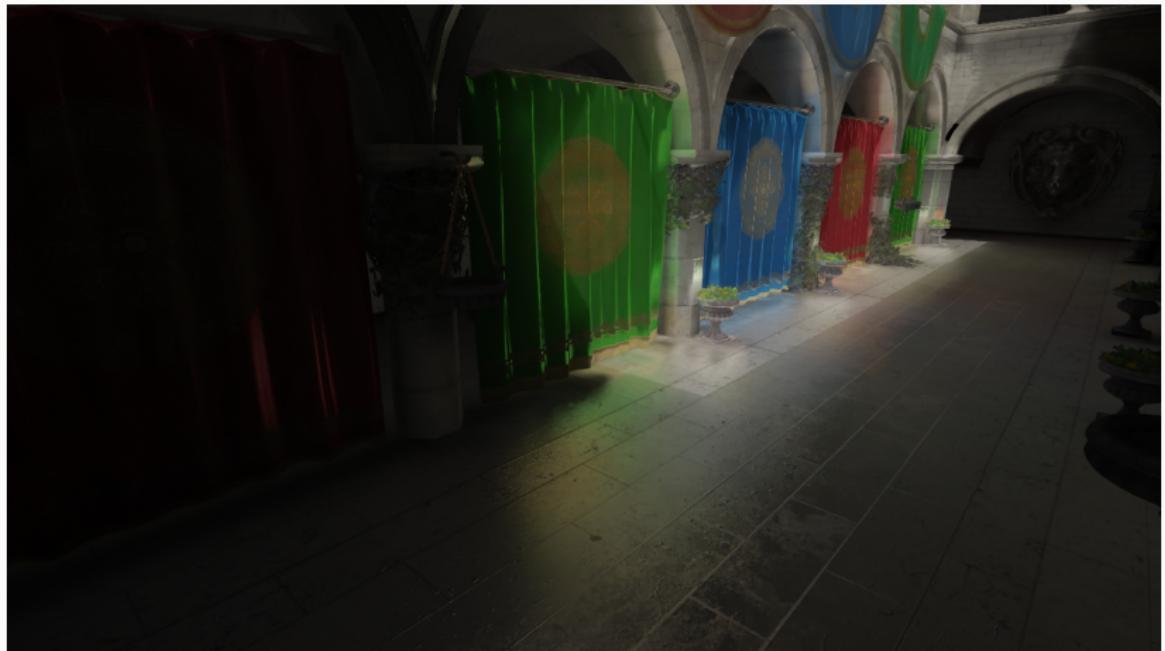
Without voxel warping

# Voxel Warping



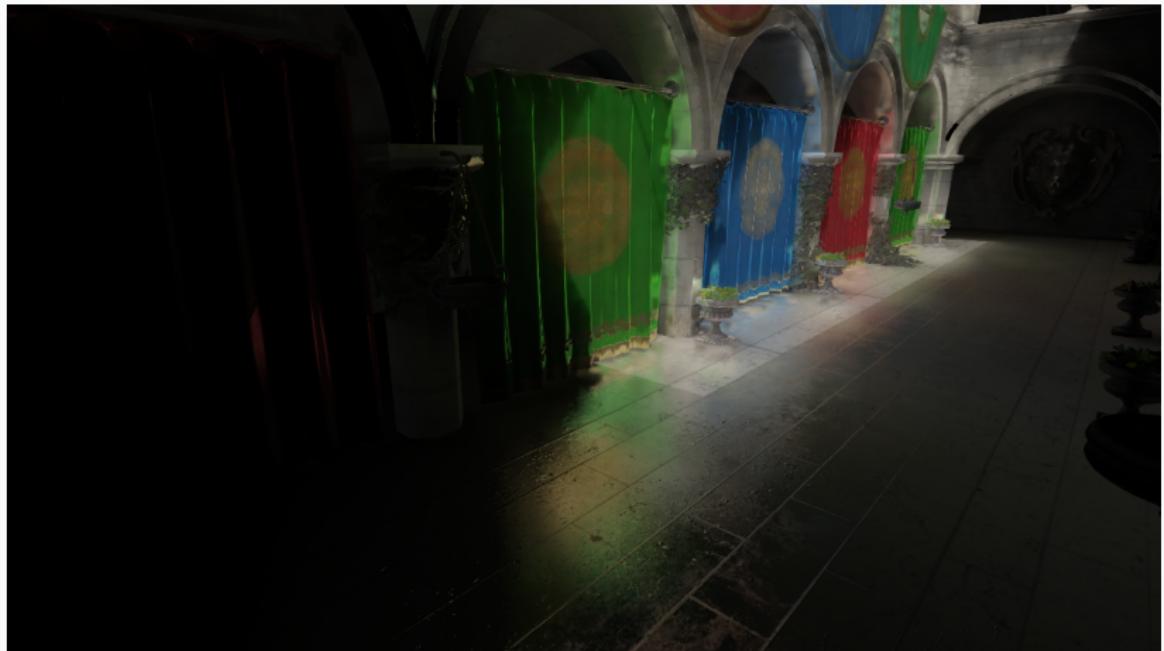
With voxel warping

# Perspective Voxel Warping



Without voxel warping

# Perspective Voxel Warping



With voxel warping

## Related Work

How does it compare with other methods?

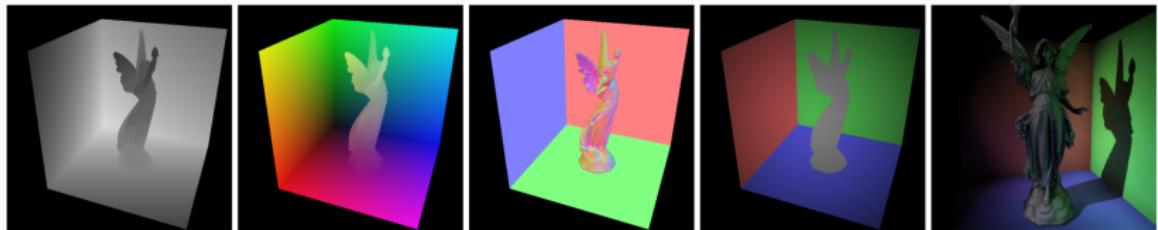
Important parts of global illumination algorithms:

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

## Related Work—Reflective Shadowmaps

Important parts of global illumination algorithms:

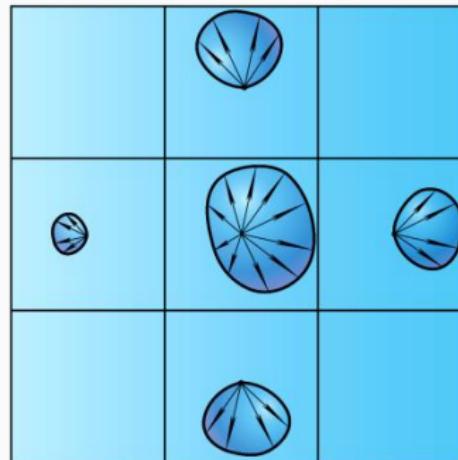
1. Scene representation? **Reflective shadowmap (RSM)**
2. Light computation? **None, use color and normal from RSM**
3. Light sampling? **Sample nearby points in RSM**



## Related Work—Light Propagation Volumes

Important parts of global illumination algorithms:

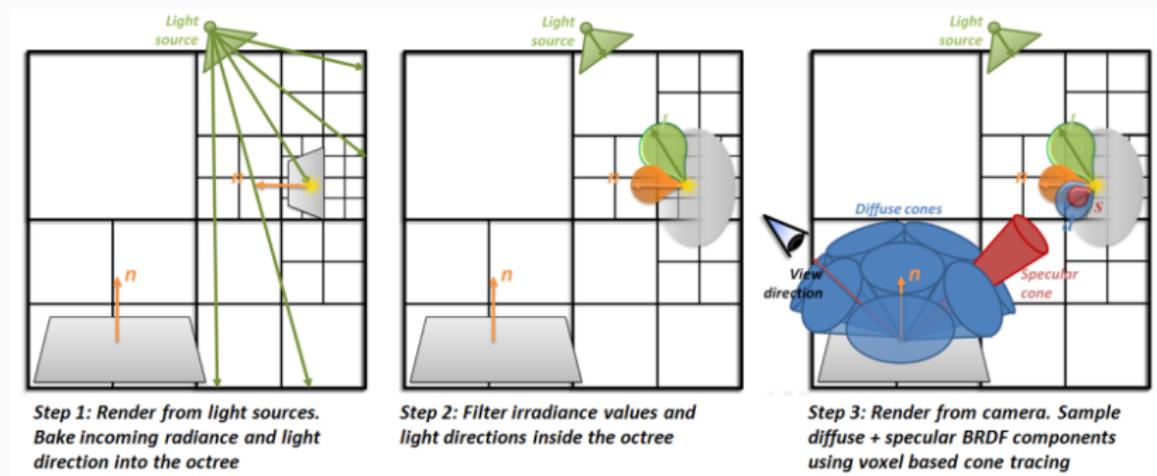
1. Scene representation? **Voxel grid (incomplete)**
2. Light computation? **Iterative propagation**
3. Light sampling? **Texture lookup**



# Related Work—Voxel Cone Tracing

Important parts of global illumination algorithms:

1. Scene representation? **Sparse voxel octree (or clipmap)**
2. Light computation? **Mipmaps**
3. Light sampling? **Voxel cone tracing**



## Related Work—Ours

Important parts of global illumination algorithms:

1. Scene representation? **Warped voxel grid**
2. Light computation? **Mipmaps**
3. Light sampling? **Voxel cone tracing**

# Conclusion

- Real-time implementation<sup>1</sup> of global illumination using voxel cone tracing
- Implementation and comparison of two voxelization methods
- Investigation into warped voxels

---

<sup>1</sup>Find the source here: [github.com/sfreed141/vct](https://github.com/sfreed141/vct)

## Future Work

- Cascaded sparse 3D textures
- Take advantage of tessellated voxelization to try to resolve temporal artifacts
- Spherical harmonics, anisotropic filtering, adaptive cone tracing quality, other miscellaneous optimizations

Thank you!

Questions?