2021 캡스톤 디자인 최종 발표

Team 6. Lit

Kookmin Univ. Software Dept. **Yang Kyowon**

Kookmin Univ. Software Dept. **Kwak Sangyeol**

Contents

- Motivation
- Framework
- Features
- Implementation
- Results (Demo Video)
- Future Works
- Q & A

Motivation

- Nature is full of beauty!

Motivation

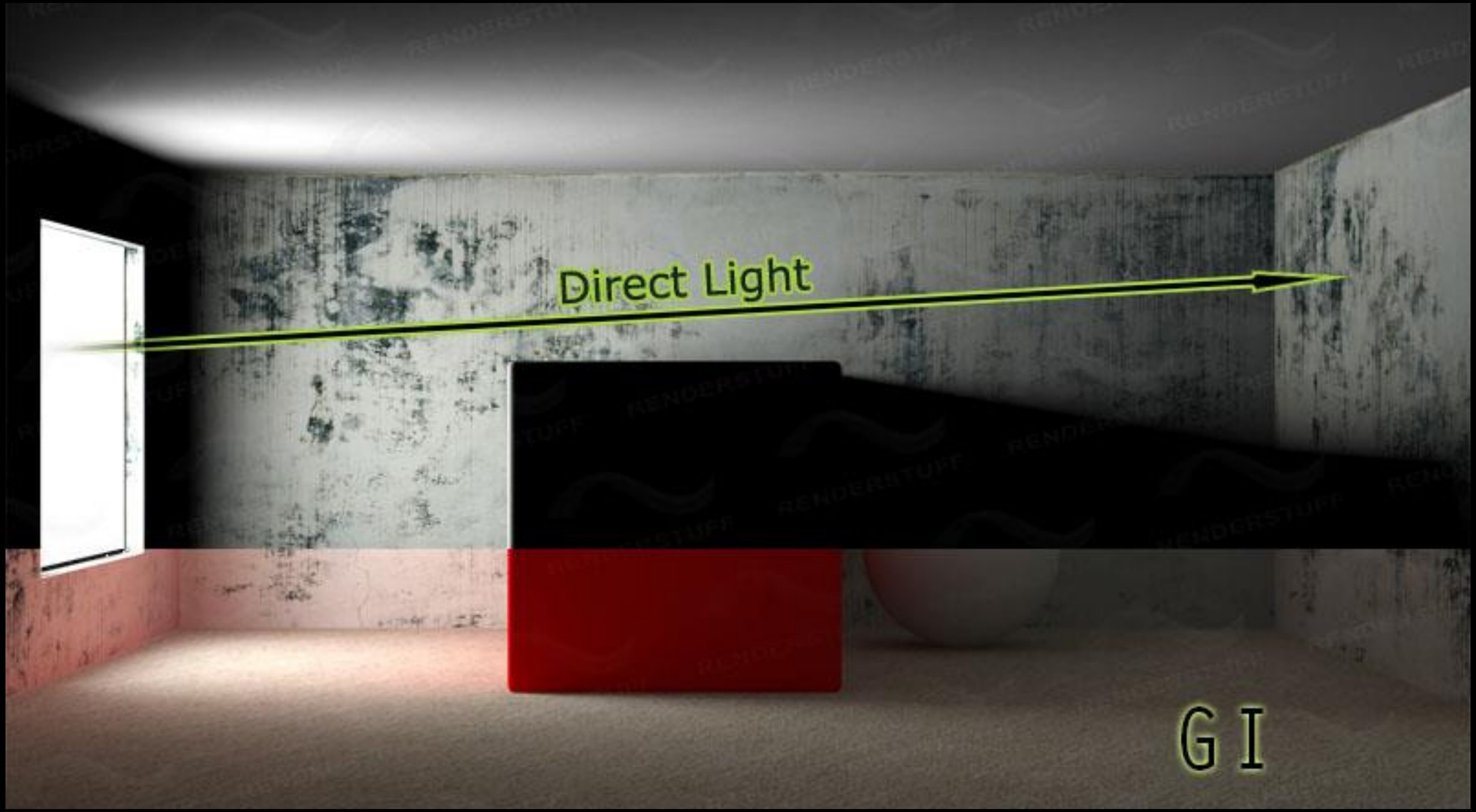
- But, Express this beauty in a real-time is still **challenging issues**

NVIDIA RTX

Cyberpunk 2077(CD PROJEKT)

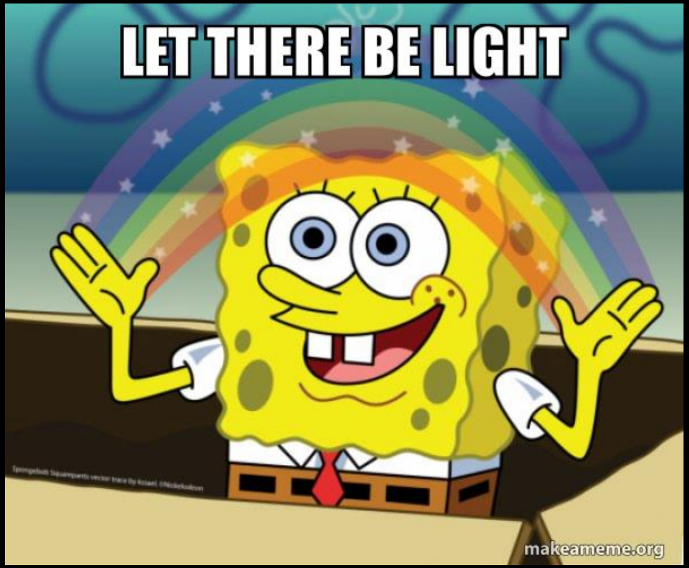
Motivation

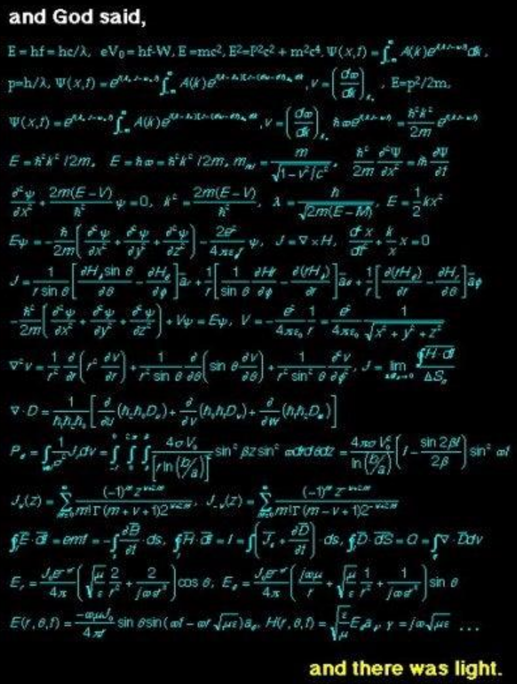
- Especially, **Global Illumination(indirect lights) effect** is major component to achieve high quality rendering results!



Lit : Let there be light!

- Lets implement **renderer** to synthesize realistic image in a **real time**!





And God said…

$$\oiint \vec{E}\,\partial\vec{s} = \frac{Q}{\epsilon_0}$$

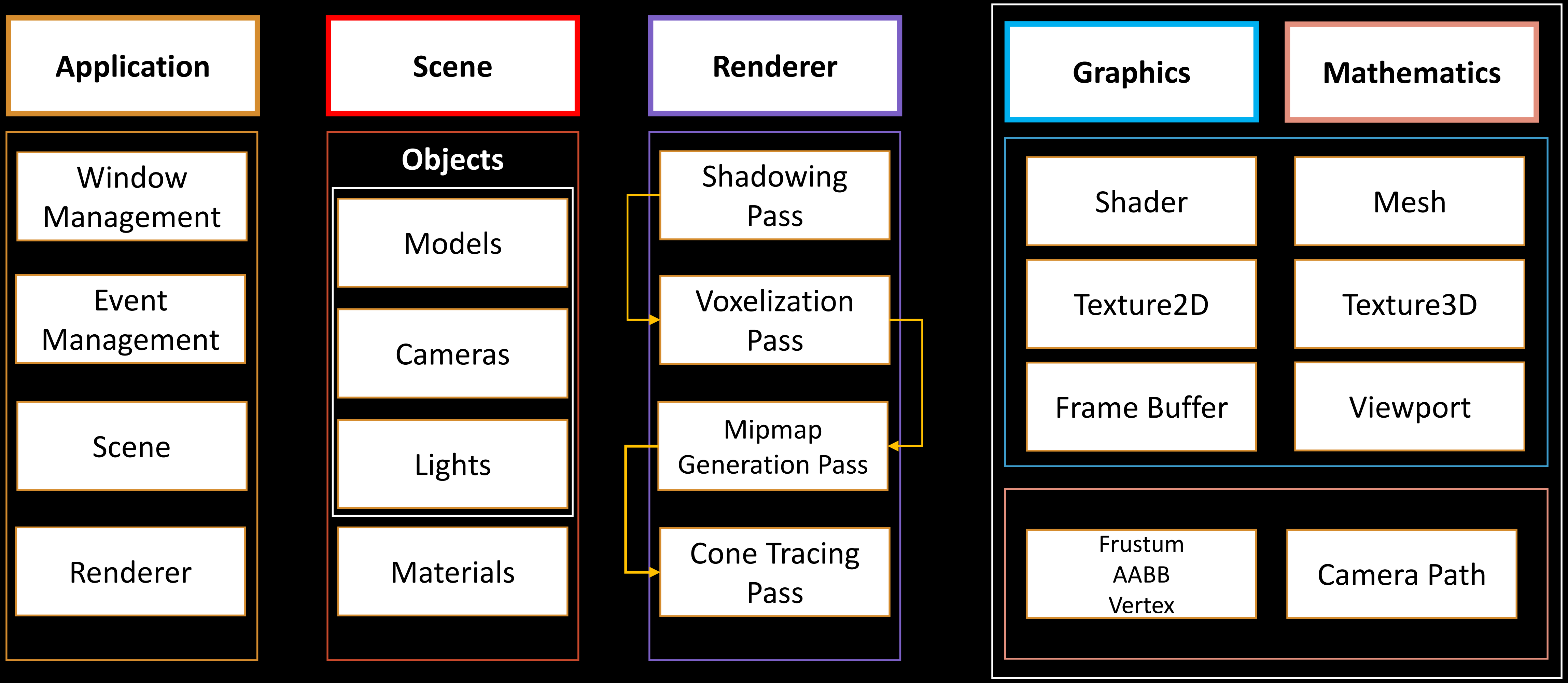
$$\oiint \vec{B}\,\partial\vec{s} = 0$$

$$\oint \vec{E}\,\partial\vec{l} = \oiint \frac{\partial\vec{B}}{\partial t}$$

$$\oint H\,\partial\vec{l} = i + \epsilon\,\frac{\partial\vec{B}}{\partial t}$$

…and there was light.

Framework


Application
Window Management
Event Management
Scene
Renderer
Scene
Objects
Models
Cameras
Lights
Materials
Renderer
Shadowing Pass
Voxelization Pass
Mipmap Generation Pass
Cone Tracing Pass
Graphics
Mathematics
Shader
Mesh
Texture2D
Texture3D
Frame Buffer
Viewport
Frustum
AABB
Vertex
Camera Path

Framework – Low Level APIs

Application

Graphics

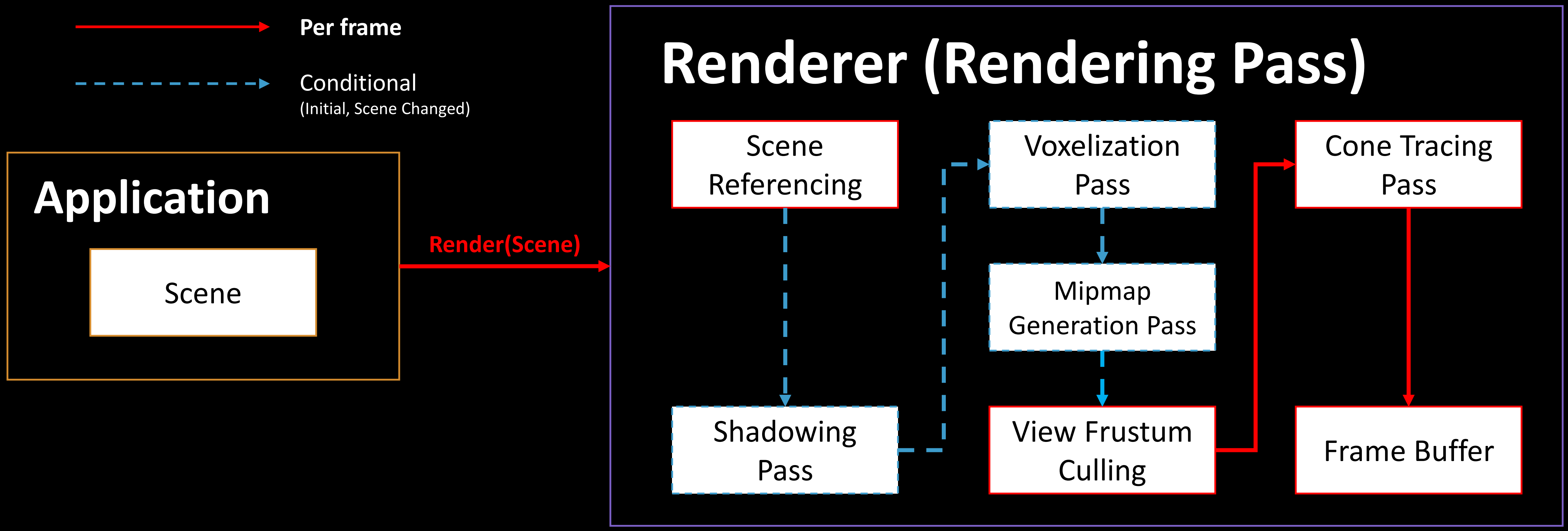
Mathematics

Platforms

Features

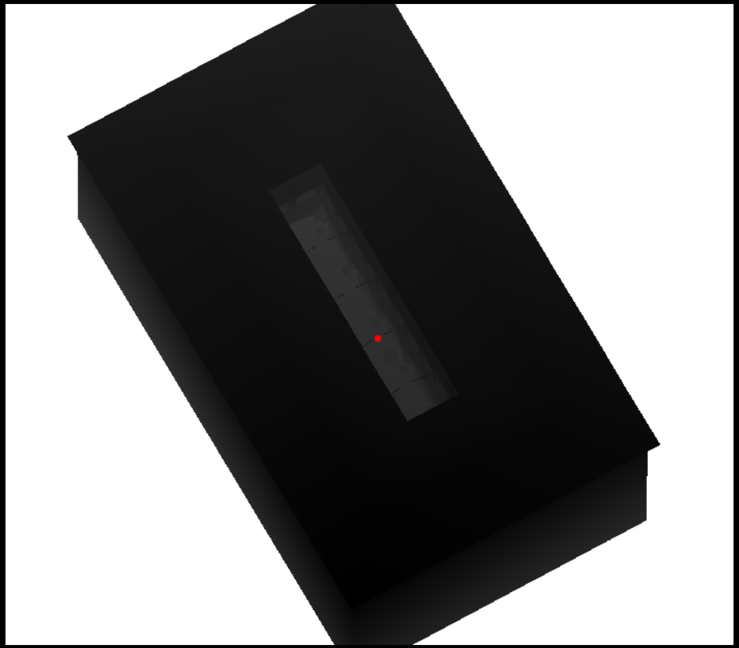
- Scene Management
  - Objects
  - Cameras
  - Lights
- Camera Path Animator
- Physically Based Workflow
- View Frustum Culling
- **Real-time Global Illumination Effects (Voxel Cone Tracing)**

Implementation (Renderer)



Shadowing Pass

- Render Depth Map(Shadow Map) from Light Source's view

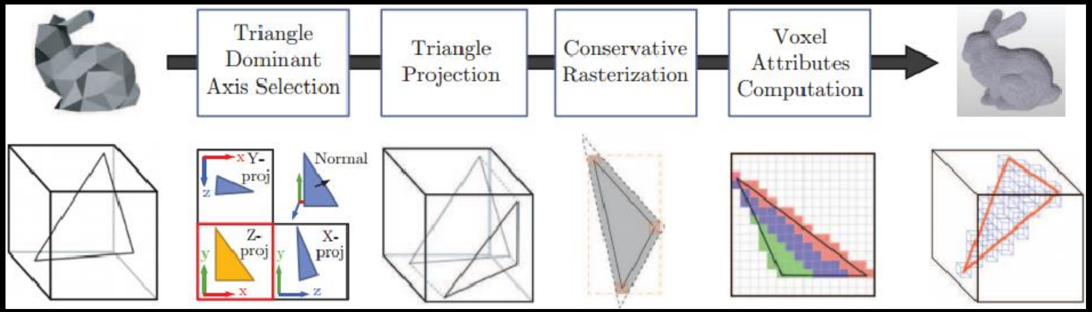


Shadow Map (Depth Map via Light Source)

Lambertian Diffuse with Shadow

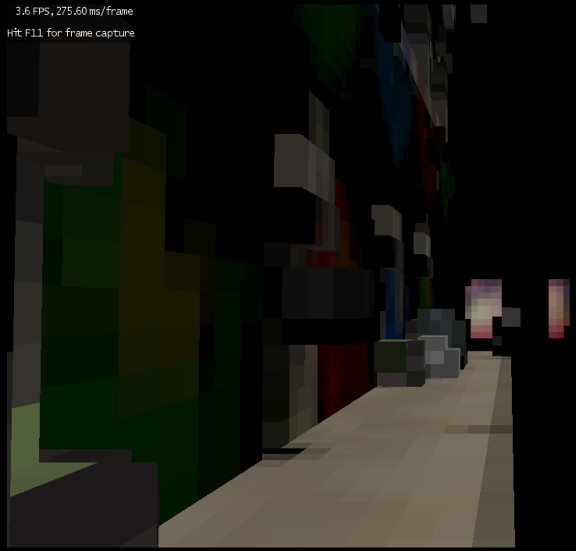
Voxelization Pass

- Voxelize entire scene objects through geometry shader to 3D Texture



Voxelization Algorithms

Voxelize Scene and store radiance at 3D texture
(Diffuse Lambertian reflectance with Shadow)



Voxelized Sponza Scene (Lambertian Diffuse)
< Our renderer used 512^3 RGBA8 3D Texture >

Mipmap Generation Pass

- Problem! : Built-in Mipmap generation methods is too slow!

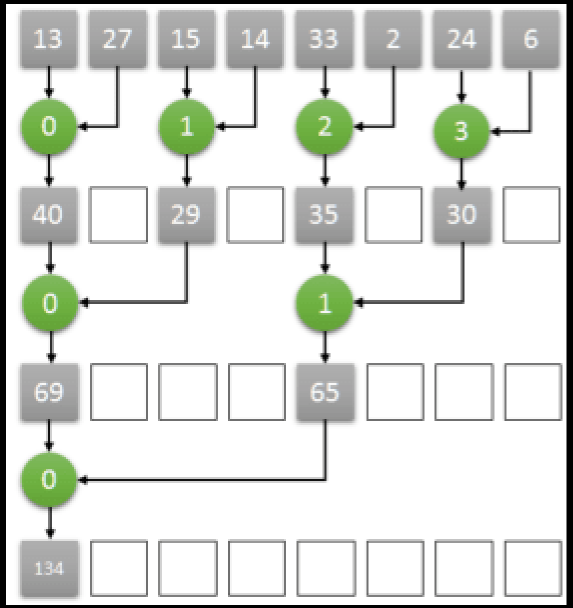
API Call	Count	Avg CPU ms	Σ CPU ms	Avg GPU ms	Σ GPU ms
glBindTexture()	1,874	<0.01	0.43	0.00	0.00
glActiveTexture()	1,872	<0.01	0.09	0.00	0.00
glUniform1i()	1,734	<0.01	0.19	0.00	0.00
glUniform1f()	838	<0.01	0.08	0.00	0.00
glBindVertexArray()	686	<0.01	0.42	0.00	0.00
glUniform3fv()	461	<0.01	0.06	0.00	0.00
glDrawElements()	343	<0.01	1.31	0.01	4.96
glUniform4fv()	252	<0.01	0.06	0.00	0.00
glUniformMatrix4fv()	48	<0.01	0.02	0.00	0.00
glEnable()	27	<0.01	<0.01	0.00	0.00
glBindFramebuffer()	4	<0.01	0.04	0.00	0.00
glDisable()	4	<0.01	<0.01	0.00	0.00
glUseProgram()	3	<0.01	0.01	0.00	0.00
glViewport()	3	<0.01	<0.01	0.00	0.00
glClearColor()	2	<0.01	<0.01	0.00	0.00
glClear()	2	0.04	0.09	0.01	0.02
glColorMask()	2	<0.01	<0.01	0.00	0.00
glCullFace()	1	<0.01	<0.01	0.00	0.00
glFrontFace()	1	<0.01	<0.01	0.00	0.00
glGetIntegerv()	1	<0.01	<0.01	0.00	0.00
glClearTexImage()	1	0.04	0.04	0.00	0.00
glBindImageTexture()	1	<0.01	<0.01	0.00	0.00
glGenerateTextureMipmap()	1	73.62	73.62	0.00	0.00
glBindRenderbuffer()	1	<0.01	<0.01	0.00	0.00
SwapBuffers()	1	0.54	0.54	0.00	0.00

- glGenerateTextureMipmap
  - AMD Ryzen 2700x (16 threads)
  - NVIDIA GeForce RTX 2080
  - 3D Texture (256^3) : RGBA8
  - 73.62 ms to generate entire levels
  - ~14 FPS

Mipmap Generation Pass

- Perform mipmap generation using Parallel Reduction

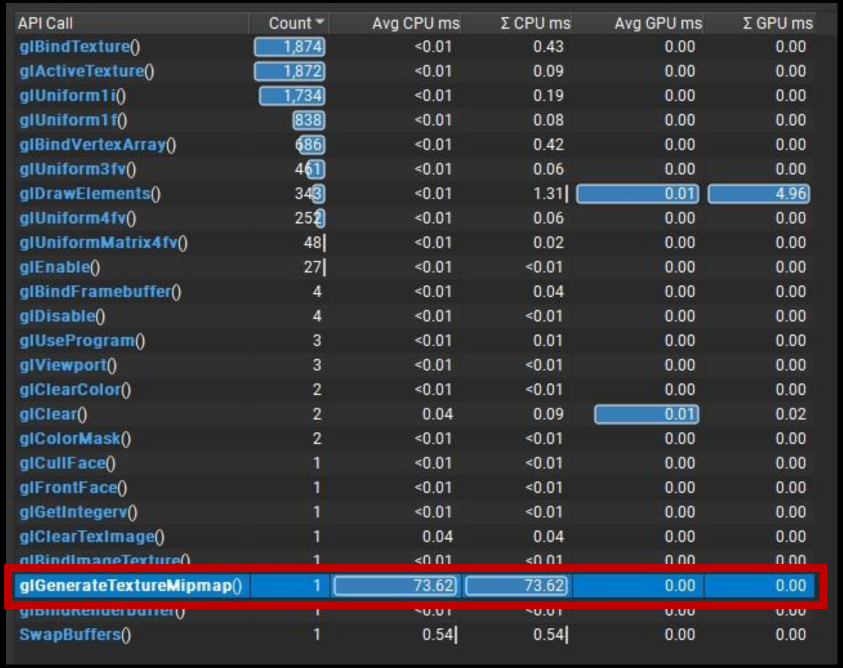
Mipmap Chains (2D)



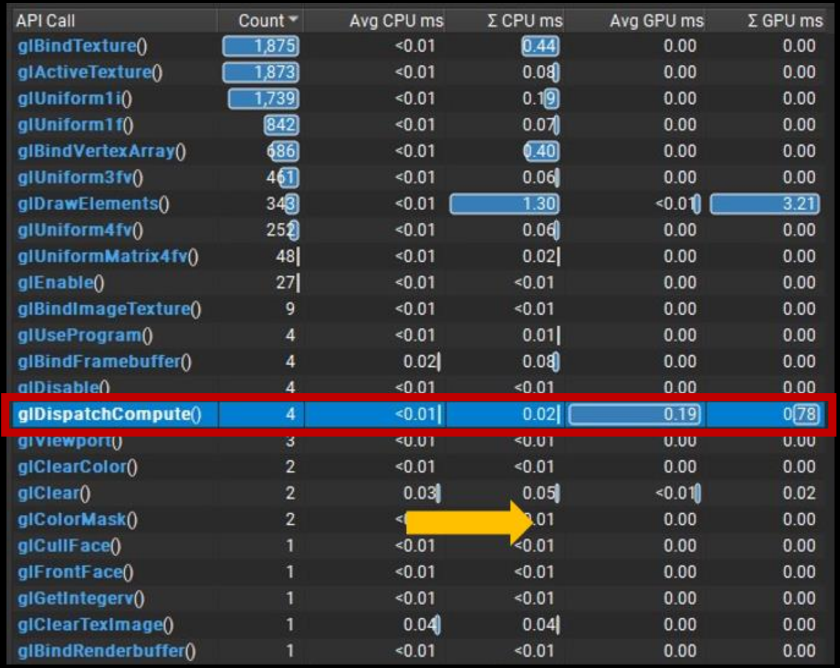
- Parallel Reduction based Mipmap Generation
  - Compute Shader
  - Generate two levels per dispatch
  - Optimized to NVIDIA Pascal Architecture

Mipmap Generation Pass

- Perform mipmap generation using Parallel Reduction Compute Shader



API Call	Count	Avg CPU ms	Σ CPU ms	Avg GPU ms	Σ GPU ms
glBindTexture()	1,874	<0.01	0.43	0.00	0.00
glActiveTexture()	1,872	<0.01	0.09	0.00	0.00
glUniform1i()	1,734	<0.01	0.19	0.00	0.00
glUniform1f()	838	<0.01	0.08	0.00	0.00
glBindVertexArray()	686	<0.01	0.42	0.00	0.00
glUniform3fv()	461	<0.01	0.06	0.00	0.00
glDrawElements()	343	<0.01	1.31	0.01	4.96
glUniform4fv()	252	<0.01	0.06	0.00	0.00
glUniformMatrix4fv()	48	<0.01	0.02	0.00	0.00
glEnable()	27	<0.01	<0.01	0.00	0.00
glBindFramebuffer()	4	<0.01	0.04	0.00	0.00
glDisable()	4	<0.01	<0.01	0.00	0.00
glUseProgram()	3	<0.01	0.01	0.00	0.00
glViewport()	3	<0.01	<0.01	0.00	0.00
glClearColor()	2	<0.01	<0.01	0.00	0.00
glClear()	2	0.04	0.09	0.01	0.02
glColorMask()	2	<0.01	<0.01	0.00	0.00
glCullFace()	1	<0.01	<0.01	0.00	0.00
glFrontFace()	1	<0.01	<0.01	0.00	0.00
glGetIntegerv()	1	<0.01	<0.01	0.00	0.00
glClearTexImage()	1	0.04	0.04	0.00	0.00
glBindImageTexture()	1	<0.01	<0.01	0.00	0.00
glGenerateTextureMipmap()	1	73.62	73.62	0.00	0.00
glBindRenderbuffer()	1	<0.01	<0.01	0.00	0.00
SwapBuffers()	1	0.54	0.54	0.00	0.00



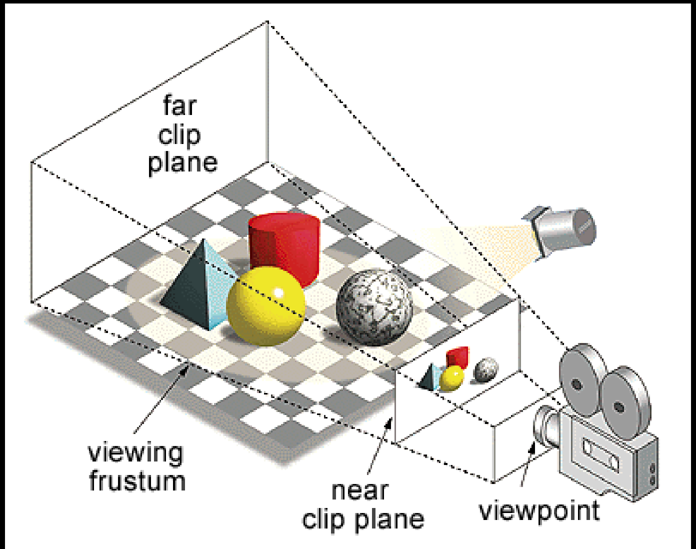
API Call	Count	Avg CPU ms	Σ CPU ms	Avg GPU ms	Σ GPU ms
glBindTexture()	1,875	<0.01	0.44	0.00	0.00
glActiveTexture()	1,873	<0.01	0.08	0.00	0.00
glUniform1i()	1,739	<0.01	0.19	0.00	0.00
glUniform1f()	842	<0.01	0.07	0.00	0.00
glBindVertexArray()	686	<0.01	0.40	0.00	0.00
glUniform3fv()	461	<0.01	0.06	0.00	0.00
glDrawElements()	343	<0.01	1.30	<0.01	3.21
glUniform4fv()	252	<0.01	0.06	0.00	0.00
glUniformMatrix4fv()	48	<0.01	0.02	0.00	0.00
glEnable()	27	<0.01	<0.01	0.00	0.00
glBindImageTexture()	9	<0.01	<0.01	0.00	0.00
glUseProgram()	4	<0.01	0.01	0.00	0.00
glBindFramebuffer()	4	0.02	0.08	0.00	0.00
glDisable()	4	<0.01	<0.01	0.00	0.00
glDispatchCompute()	4	<0.01	0.02	0.19	0.78
glViewport()	3	<0.01	<0.01	0.00	0.00
glClearColor()	2	<0.01	<0.01	0.00	0.00
glClear()	2	0.03	0.05	<0.01	0.02
glColorMask()	2	[illegible]	[illegible]	0.00	0.00
glCullFace()	1	<0.01	<0.01	0.00	0.00
glFrontFace()	1	<0.01	<0.01	0.00	0.00
glGetIntegerv()	1	<0.01	<0.01	0.00	0.00
glClearTexImage()	1	0.04	0.04	0.00	0.00
glBindRenderbuffer()	1	<0.01	<0.01	0.00	0.00

Almost **100 times faster** than built-in mipmap generation method at same configurations!

$$\text{Dispatch Count} = \frac{\log_2 256}{2}$$

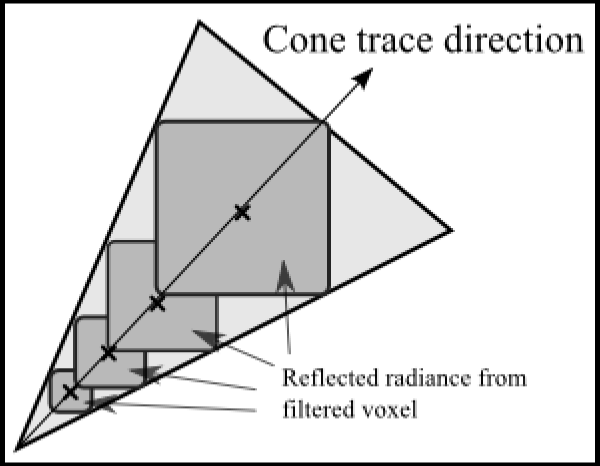
View Frustum Culling

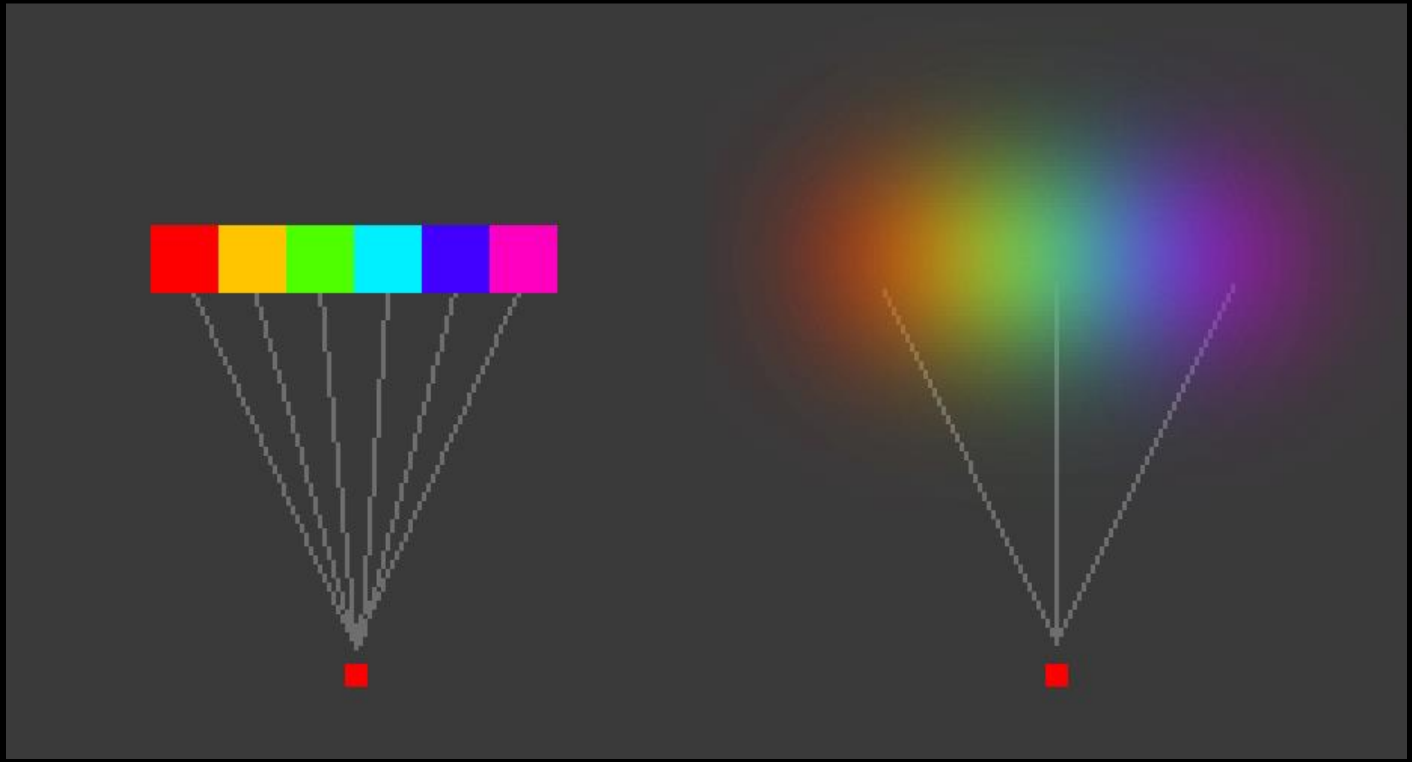
- To decrease cone tracing and draw call overheads, cull objects which not visible to viewer



Hierarchical AABBs (Model-Meshes)

Cone Tracing

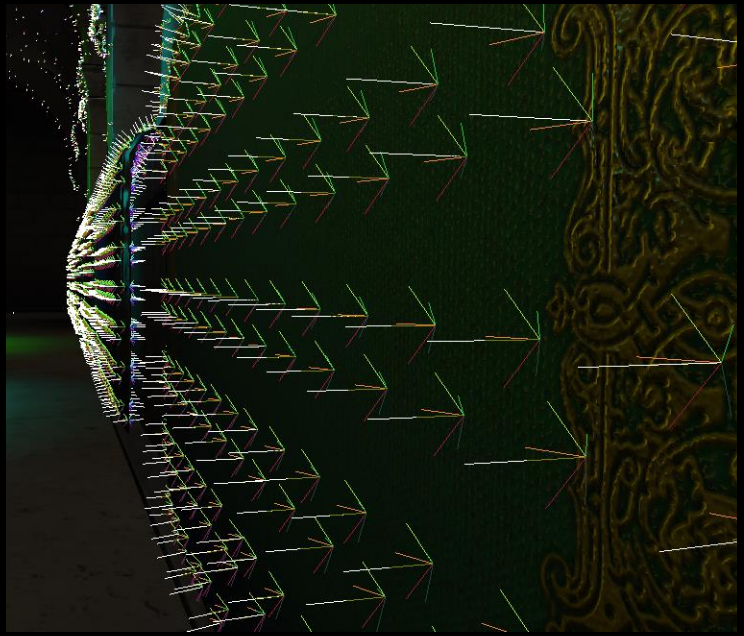


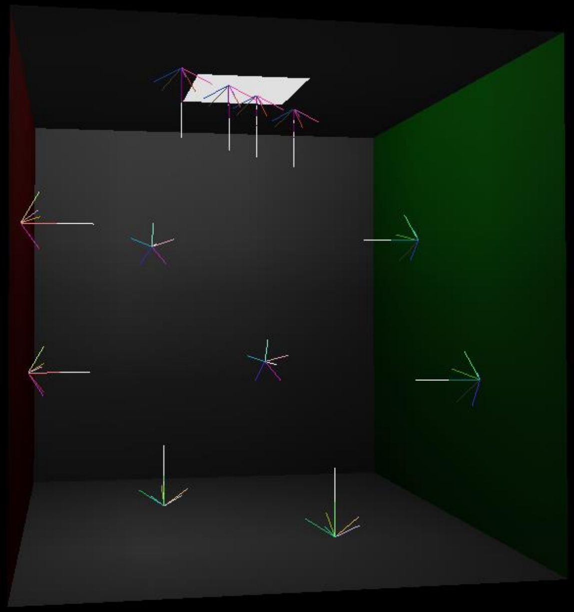


Big cone aperture sampling wider area of the scene.
It will be done through mipmap generation and trilinear interpolation of 3D Textures(Voxel Volume)

Cone Tracing Pass

- Final gathering using voxel based cone tracing!





Visualize diffuse Cone's directions

- Gathering Indirect Lights using Cone Tracing
  - Direct Diffuse : Lambertian BRDFs
  - Direct Specular : Cook-Torrance BRDFs
  - Indirect Diffuse : Trace 6 Cones
    - 60° per cone
    - Also compute ambient occlusion
  - Indirect Specular : GGX Importance Sampling
    - Aperture of cone is vary on material roughness
    - 2~4 Samples to achieve real-time performance
  - Linear Attenuation ($Light\ Energy \propto \frac{1}{Distance}$)

Results (Demo Video)

Future Works

- **Extend Indirect Light Bounces**
  - Using LPV(Light Propagation Volume) for 1st bounce to extend 2nd bouncing at VCT.
  - Or through compute shader to simulate (N-1) times light bouncing.
- **Improve Voxelization Method**
  - Clip-map based Voxelization to reducing memory footprint.
  - Split Geometry data(Normal, Albedo, Opacity, ..) to handle more complex scenes.

    (ex. Light has physical quantity units)
- **Implement Post-Process Effects**
  - To get more beautiful final result, we need to consider about post-process effects like DOF, Bloom, Exposure, Bokeh, etc…
- **Find more flexible and physically plausible BSDFs (Not a BRDF)**

Q & A