Dynamic Global Illumination in Real-time Rendering

Speaker: DXT00

Outline

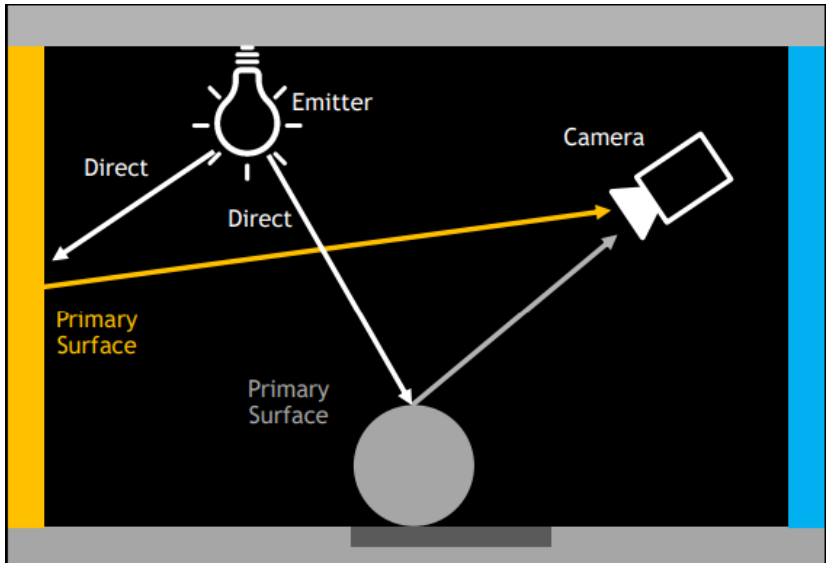
1. Background
2. Contribution of this thesis
3. Rendering Engine Designed
4. Dynamic GI Algorithm Designed
5. Conclusion
6. Future work

01

Chapter

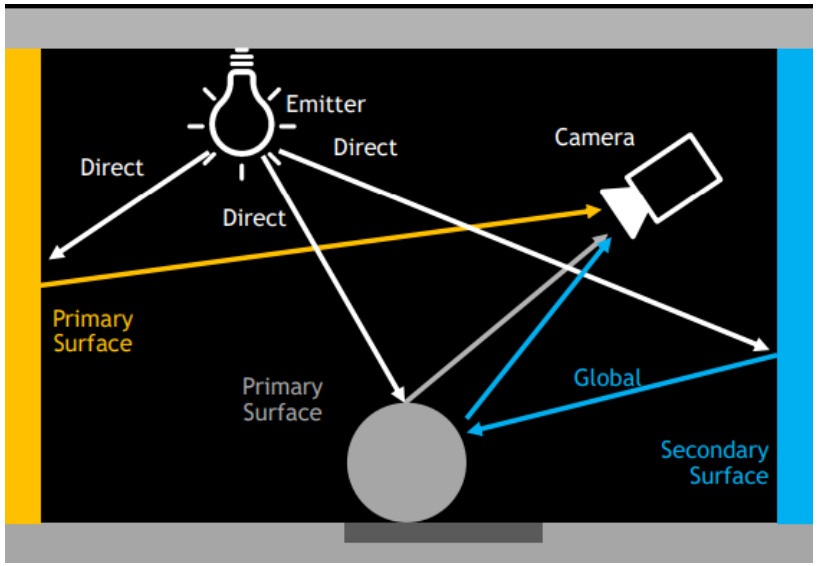
Background

Global illumination (GI) Definition



Direct illumination[1]

straight from the light emitter



Global illumination[1]

bounces off at least one other surface

Global illumination Definition

Direct light

+

Indirect diffuse light

+

Indirect specular light

Global illumination

Dynamic GI Challenge

Accurate and Realistic GI

↓

Ray tracing

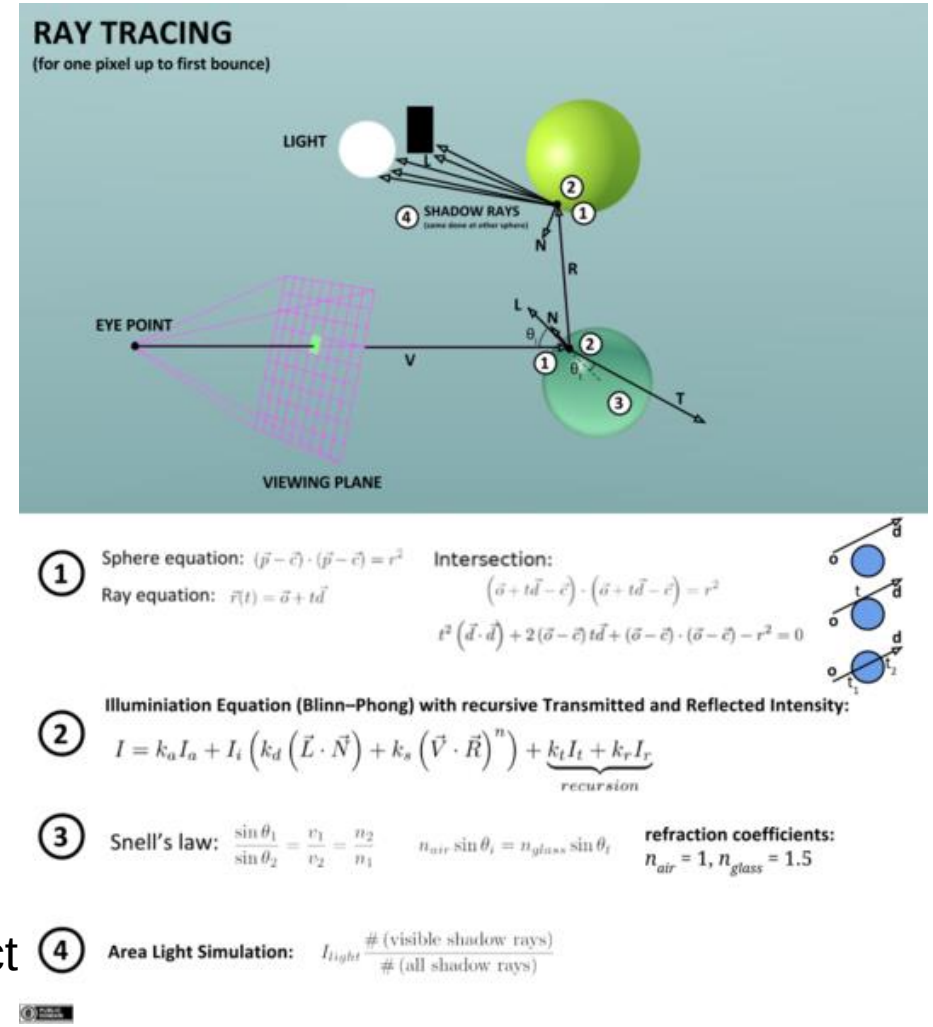
↓

Ray tracing is **expensive** and **time-consuming**

Target of Dynamic GI:

Find a efficient way to approximate GI
at real-time frame rate.

1. Rendering rate: real-time processing power (30fps~60fps)
2. Dynamic scene processing: dynamic light source, dynamic object
3. Scalability : large scene, multi-object processing
4. Rendering quality: convincing , realistic rendering



Ray tracing [1]

Dynamic scene processing

- Dynamic light source

the intensity ,position and color of the light source is variable, such as changes in light from morning to night

- Dynamic shadow

shadow changes with the change of the light source

- Dynamic object

object of various transformations, including the position, rotation and scale

- Dynamic scene

scene changes on the impact of lighting, such as window opening or blasting a hole through a wall.

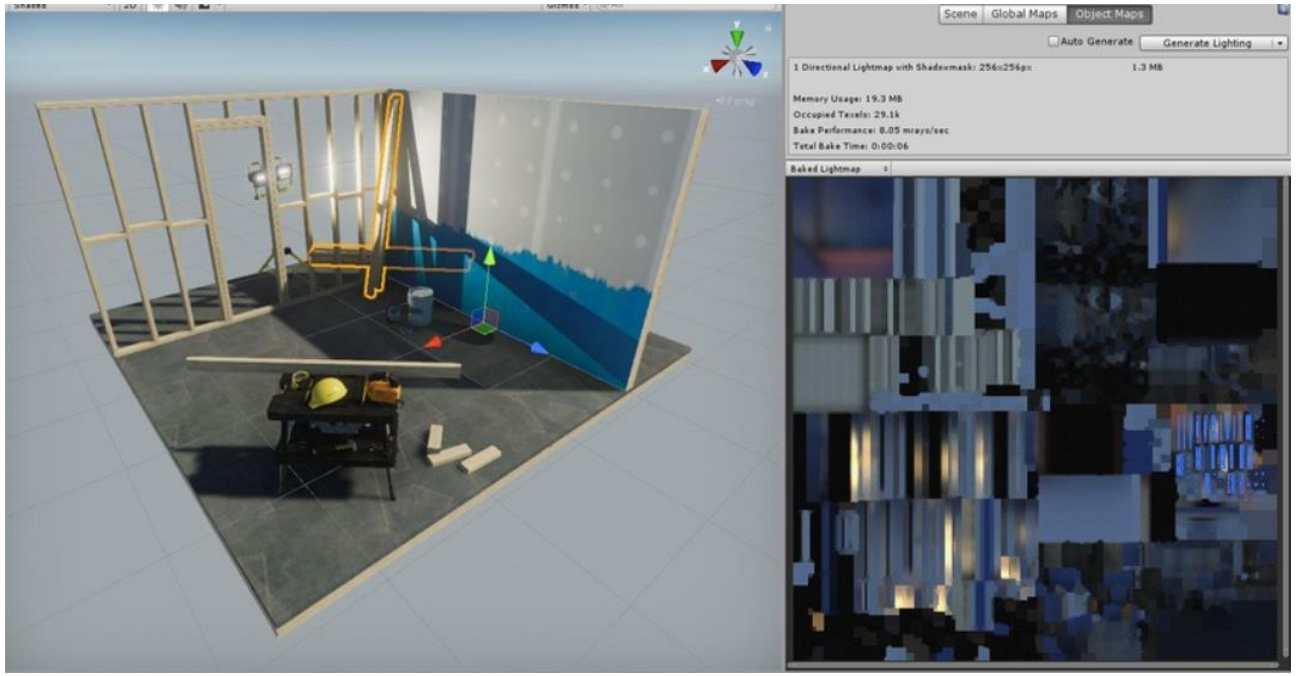
dynamic light source [1]

moving character in video game [3]

dynamic modifications [4]

Previous work

1. Baked GI



[7] backed lightmap

Unity Engine

1.Baked GI
Lightmapping
Preparing the Scene and baking the lightmaps

2.Precomputed Realtime GI
It precomputes all possible light bounces and encodes this information for use at runtime.

But both methods limited to static objects

Previous work

2. Light probe-based GI

[8] light probe cache

Unreal Engine

Precompute light probe cache

Dynamic objects get GI from probe

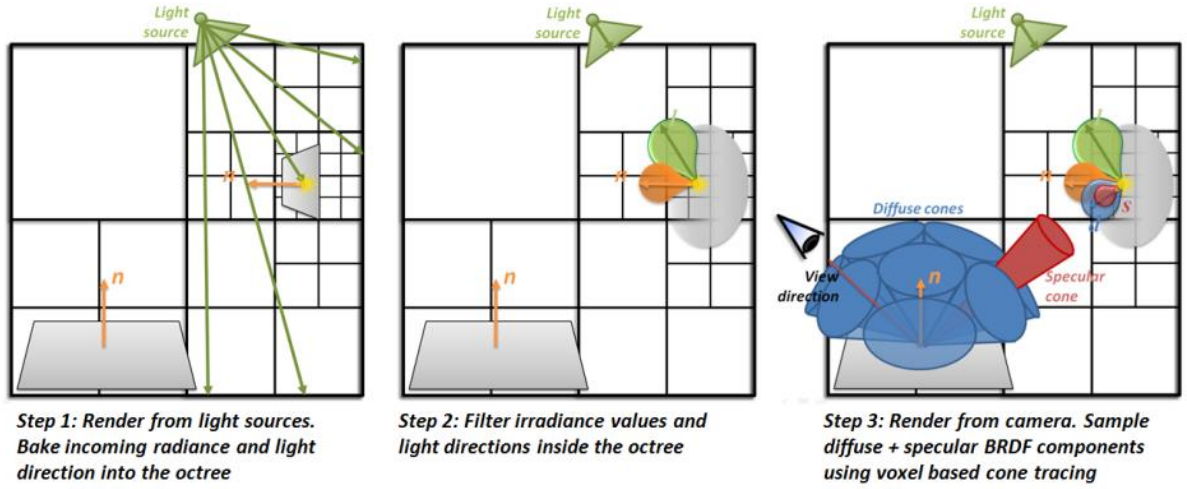
↓ But

Static object can not reflect dynamic object's light

Can not process dynamic environment light source, such as Day and night shifting.

Previous work

3. 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

[9]“Interactive Indirect Illumination Using Voxel Cone Tracing”

Sparse voxel octree

It is based on a hierarchical voxel octree representation generated and updated on the fly from a of the visibility and incoming energy. regular scene mesh coupled with an approximate voxel cone tracing that allows for a fast estimation

Build tree

Large memory consumption

02

Chapter

Contribution of this thesis

Contributions

1. **Designed a Rendering Engine** for Dynamic GI – BlackPearl Rendering Engine

2. Implemented and **Designed new rendering pipelines of** three kinds of dynamic GI algorithms: Image-based lighting GI, voxel cone tracing GI, lightprobe-based GI

3. **Optimization** of rendering efficiency and quality for **voxel cone tracing GI** and **light probe-based GI**

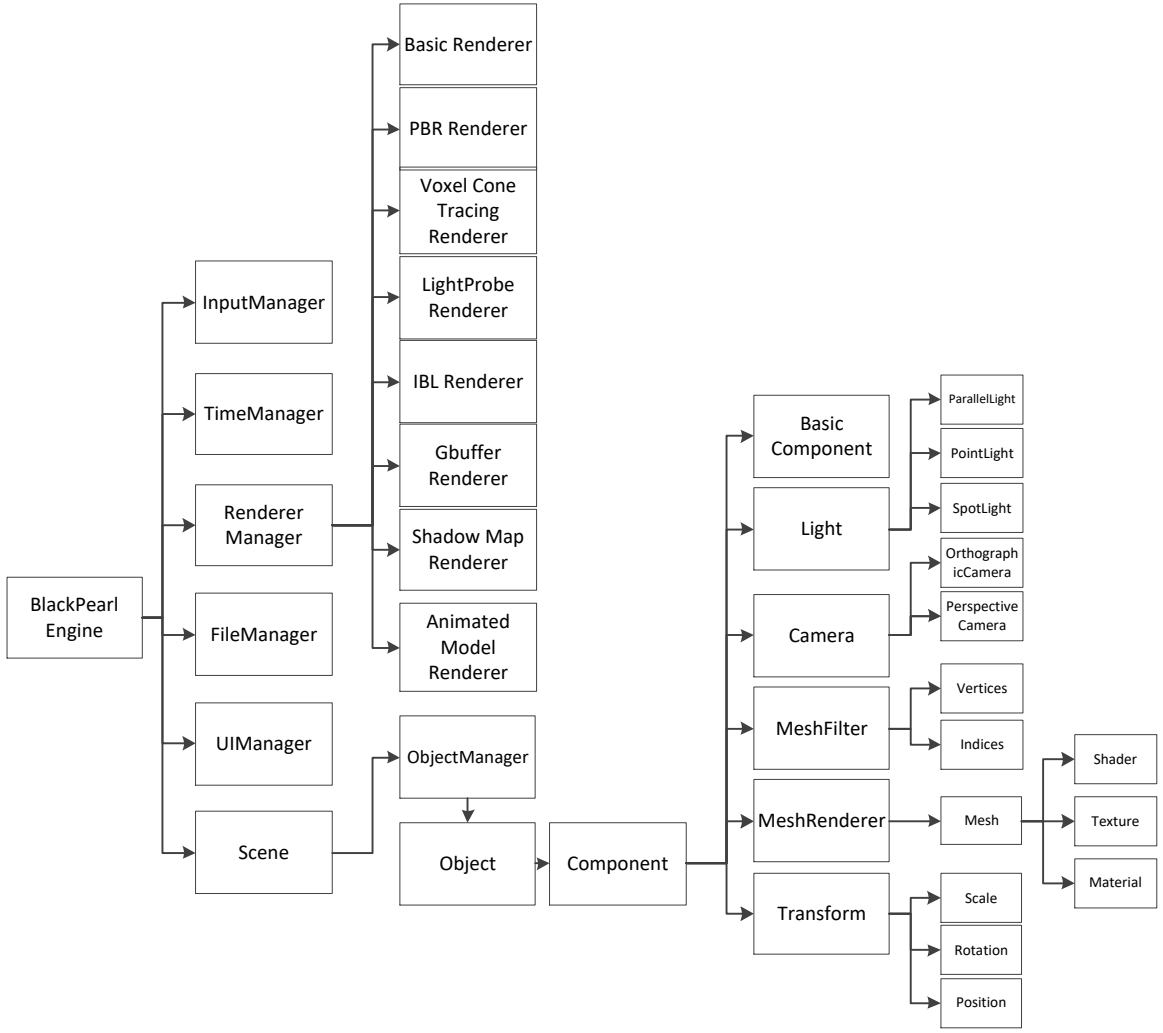
4. **Compare my work to previous work**

5. **Comparison** of three Dynamic GI algorithms, including storage data structure, applicable scenario, real-time performance and rendering quality.

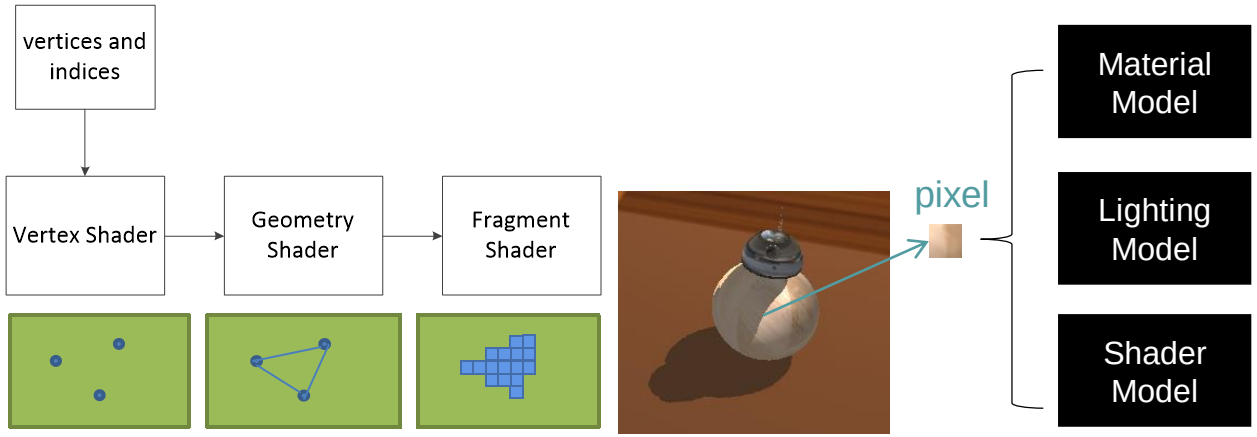
03 Chapter

Rendering Engine Designed for Dynamic GI

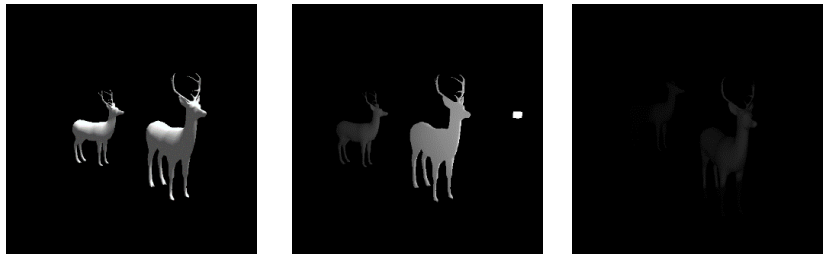
Rendering engine design



(1) Programmable rendering pipeline

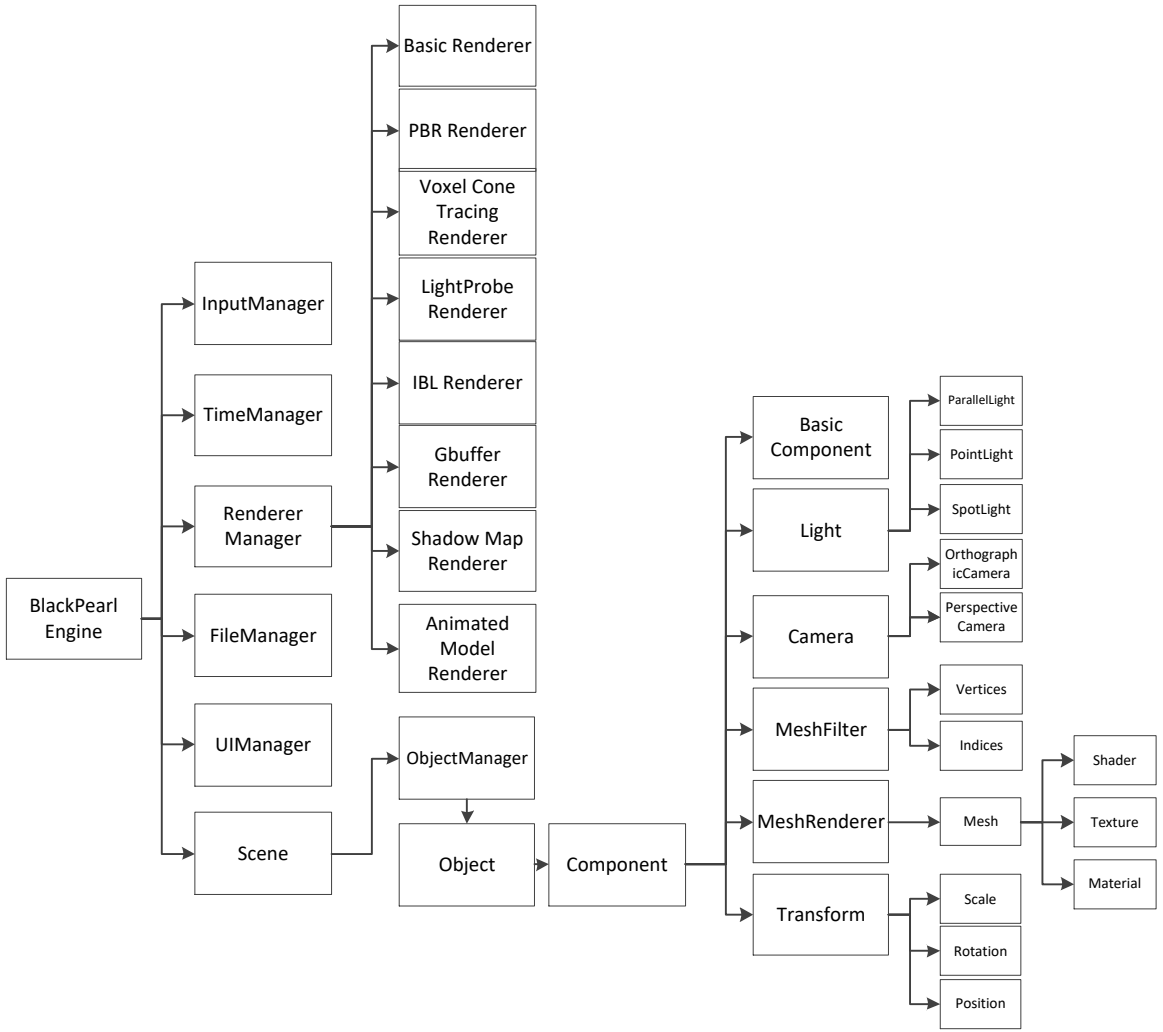


(2)Simulation of various light sources

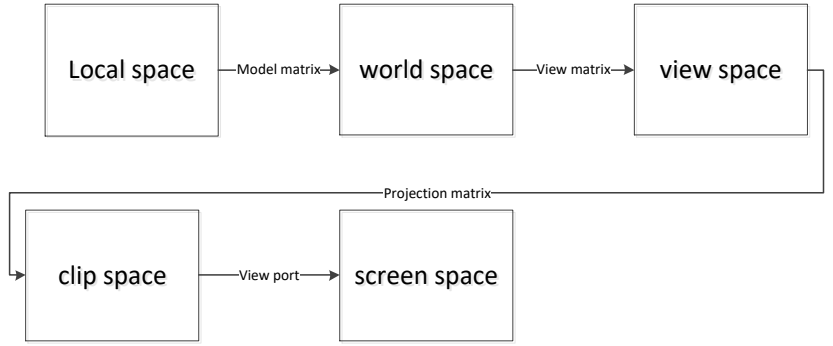


(3) Simulation of multiple rendering materials

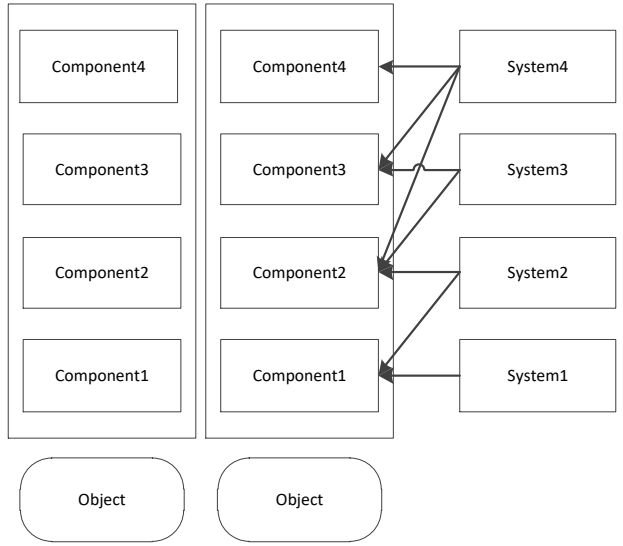
Rendering engine design



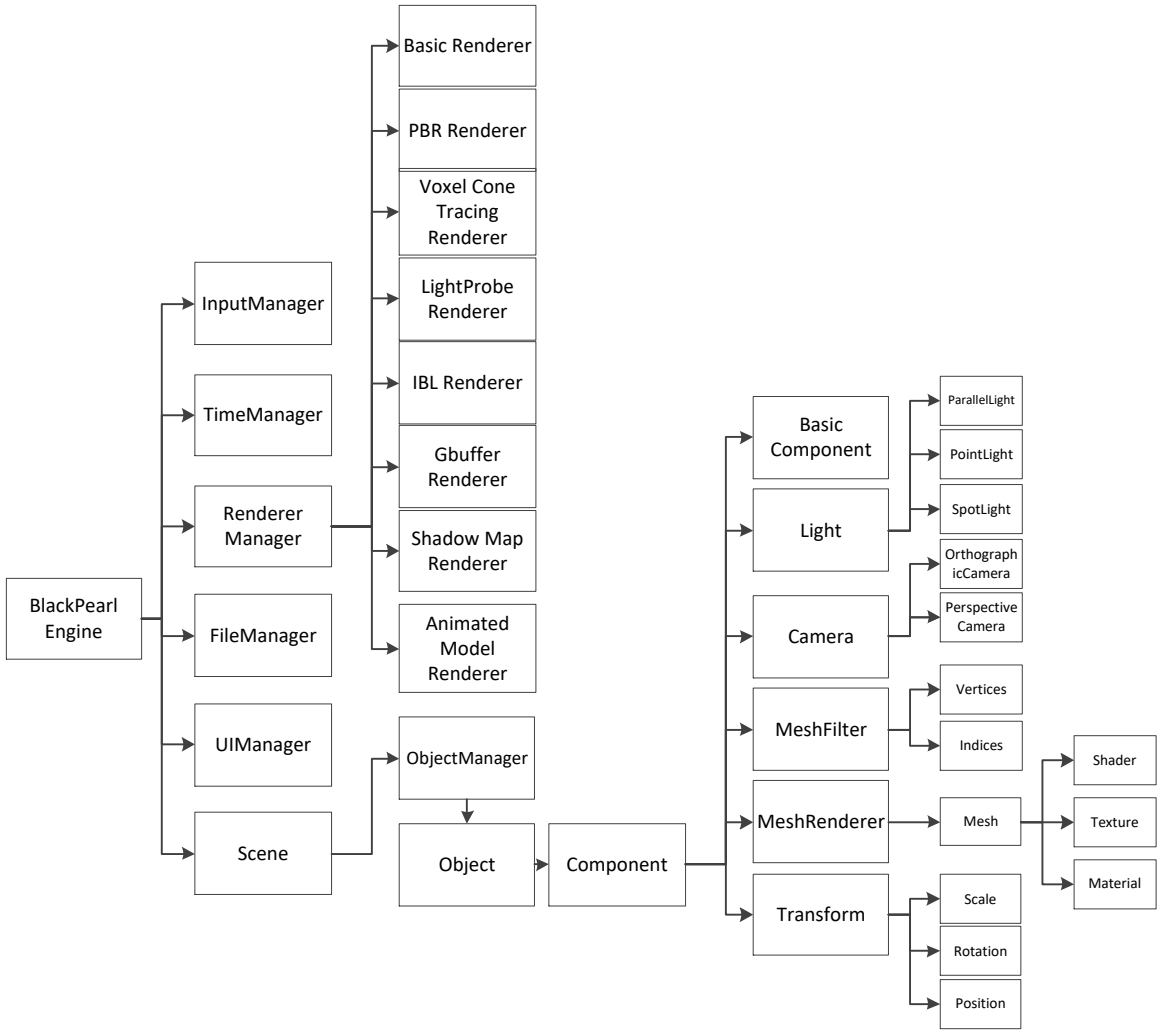
(4) Camera system and Motion control system



(5) Entity Component System(ECS) architecture



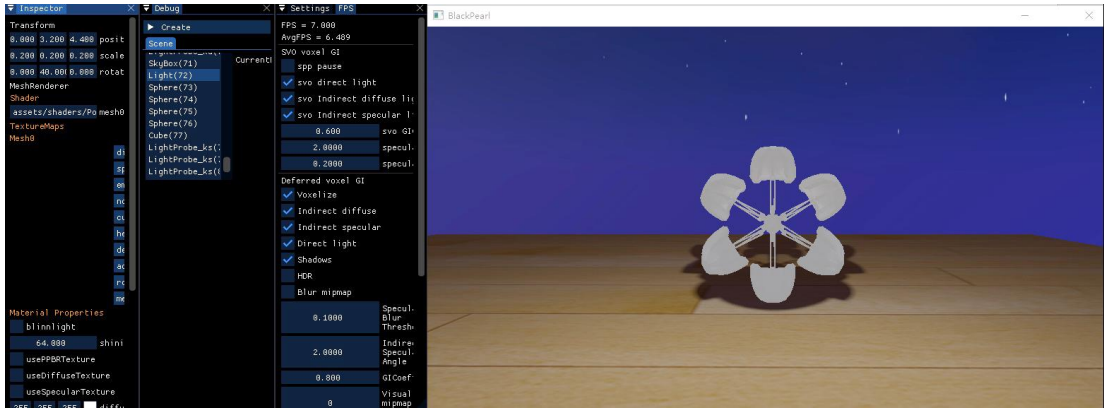
Rendering engine design



(6)Multiple rendering pipelines.

- Basic Renderer
- Physical Based Renderer
- Voxel Cone Tracing GI Renderer
- Light Probe Based GI Renderer
- Image Based Lighting GI Renderer
- ShadowMap Renderer

(7) Animation and UI



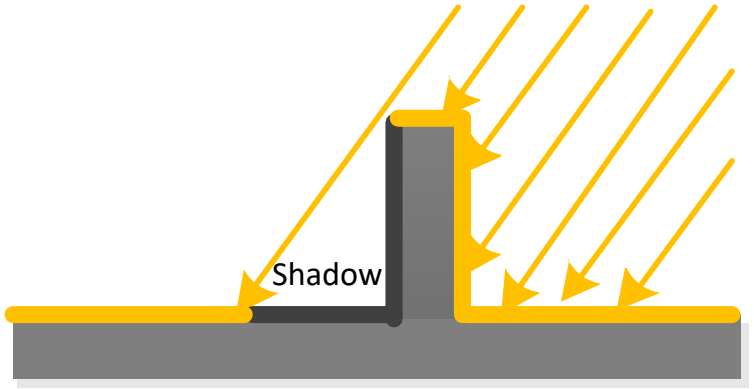
04

Chapter

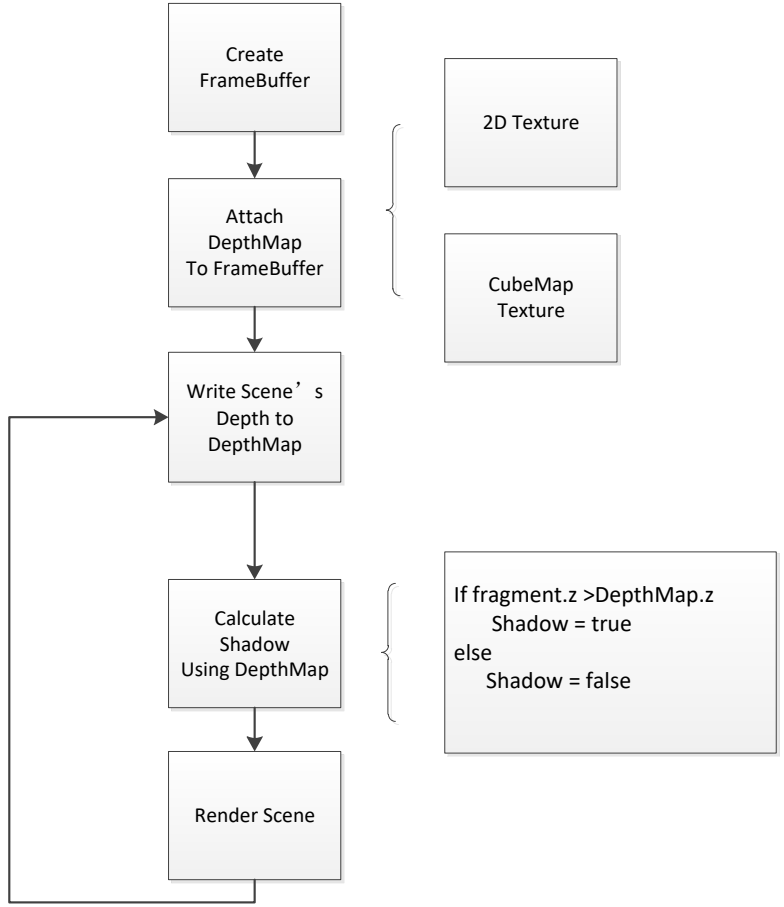
Dynamic GI Algorthm Designed

Dynamic Shadow – shadow map

Shadow map based GI algorithm: judge whether a fragment point is in the shadow



1.Stand at the position of the light source, observe the scene in the perspective of light, calculate the distance of objects in the scene from the light source, and store the depth to a shadow map
2. Use the obtained shadow map to identify the shadow. For some point p in the world, we can just transform its position to the light space and compare its depth under the light space to see if it's in shadow.



Shadow map pipeline

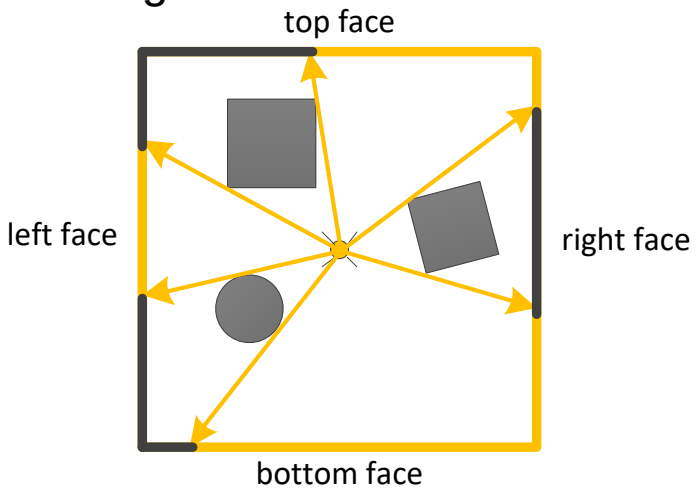
Dynamic Shadow – shadow map

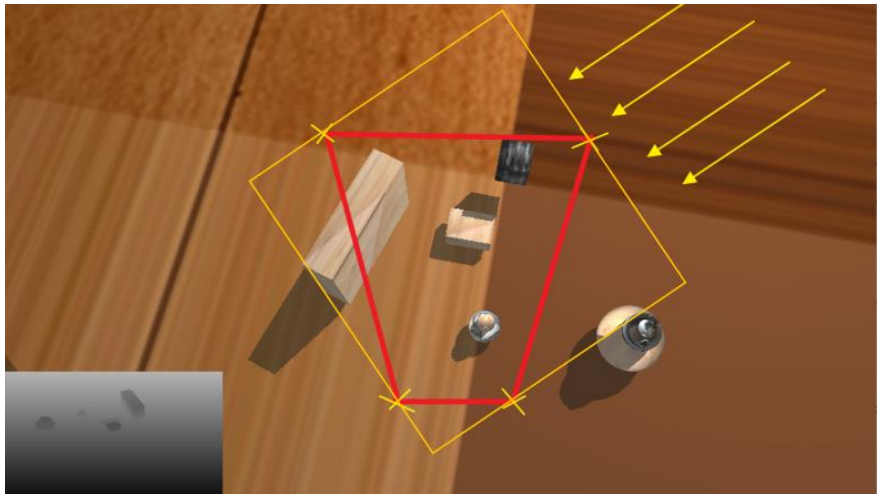
Direction light

2D texture

Point light

Cube map





Cast shadow inside the Camera’s view frustum(red trapezoid)

Optimize shadow generation range

In order to avoid redundant rendering scenes and improve the accuracy of Shadow map, the camera's view frustum
is calculated before each rendering
calculate the best Bounding Box, so as to render
only the objects in
the Bounding Box.

Image-based Lighting GI (IBL GI)

IBL Ideas: Obtain indirect light from image → Cube map reflection

It can be well combined with **physical based rendering(PBR)** algorithm

Render GI into a CubeMap

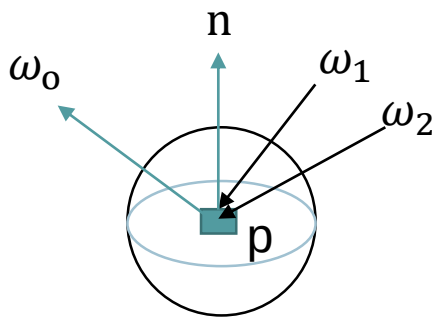
↓

Get GI from CubeMap

The Rendering Equation[17]

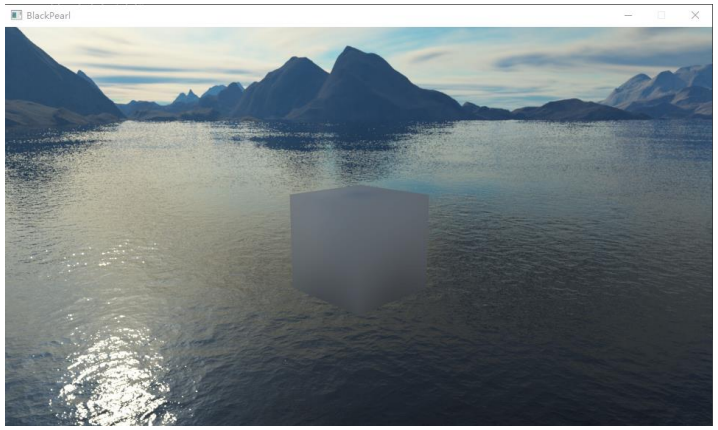
$$L_o(p,\omega_o) = L_e(p,\omega_o) + \int_\Omega f_r(p,\omega_i,\omega_o) L_i(p,\omega_i)(\omega_i \cdot n) d\omega_i$$

$$f_r = k_d f_{lambert} + k_s f_{cook-torrance}$$

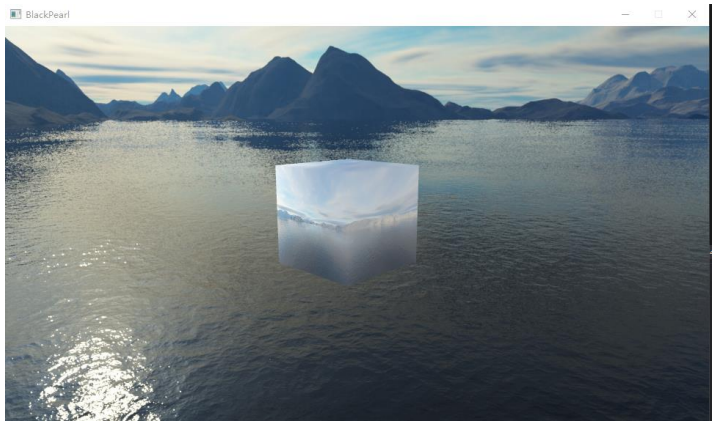


f_r is bidirectional reflective distribution function(BRDF)

Assuming that *L* denotes radiance,
we have that at each particular position and direction, the outgoing light (L_o) Is the sum of the emitted light (L_e) andthe reflected light.



diffuse irradiance cube map



mipmap = 1 prefilter specular cube map

Image-based Lighting GI (IBL GI)

$$L_o(p,\omega_o) = L_e(p,\omega_o) + \int_\Omega f_r(p,\omega_i,\omega_o) L_i(p,\omega_i)(\omega_i \cdot n) d\omega_i$$

$$f_r = k_d f_{lambert} + k_s f_{cook\text{-}torrance}$$

energy conservation

$$k_s = 1 - k_d$$

$$f_{lambert} = \frac{c}{\pi} \qquad f_{cook\text{-}torrance} = \frac{DFG}{4(\omega_o \times n)(\omega_i \times n)}$$

Direct light → only one incident ray for each point light

Indirect light → The direction of the Indirect light is from any direction of the fragment.
We need to **integrate** the hemisphere.

$$L_o(p,\omega_o) = \int_\Omega (k_d \frac{c}{\pi} + k_s \frac{DFG}{4(\omega_o \cdot n)(\omega_i \cdot n)}) L_i(p,\omega_i)(\omega_i \cdot n) d\omega_i$$

$$= \int_\Omega k_d \frac{c}{\pi} L_i(p,\omega_i)(\omega_i \cdot n) d\omega_i + \int_\Omega k_s \frac{DFG}{4(\omega_o \cdot n)(\omega_i \cdot n)} L_i(p,\omega_i)(\omega_i \cdot n) d\omega_i$$

D is a normal vector distribution function,
F is a Fresnel function,
G is a geometric function.
Together, these three functions simulate the effect of different roughness materials reflected light

$$D(h) = \frac{\alpha^2}{\pi((n \cdot h)^2(\alpha^2 - 1) + 1)^2}$$

α = $roughness^2$

h = $normalize(l + v)$

$$G(l,v,k) = (\frac{n \cdot v}{(n \cdot v)(1-k)+k})(\frac{n \cdot l}{(n \cdot l)(1-k)+k})$$

$$k_{direct} = \frac{(\alpha+1)^2}{8}$$

$$k_{IBL} = \frac{\alpha^2}{2}$$

$$F(h,v,F_0) = F_0 + (1-F_0)(1-(h \cdot v))^5$$

$$F_0 = 0.04 * metalness + diffuse.rgb * (1 - metalness)$$

Cook R L, Torrance K E,A Reflectance Model for Computer Graphics,Computer Graphics (SIGGRAPH ’81 Proceedings),
pp. 307–316, July 1981.

Image-based Lighting GI (IBL GI)

Indirect diffuse light

$$\int_{\Omega} k_d \frac{c}{\pi} L_i(p,\omega_i)(\omega_i \cdot n) d\omega_i$$

$$= k_d \frac{c}{\pi} \int_{\Omega} L_i(p,\omega_i)(\omega_i \cdot n) d\omega_i$$

c ---Material diffuse color

Input: Environment Cubemap

Output: Diffuse irradiance Cubemap

solve the integral in the hemisphere Ω by getting fixed number of discrete samples and averaging results.

Indirect specular light

$$\int_{\Omega} k_s \frac{DFG}{4(\omega_o \cdot n)(\omega_i \cdot n)} L_i(p,\omega_i)(\omega_i \cdot n) d\omega_i$$

Split Sum Approximation[10]

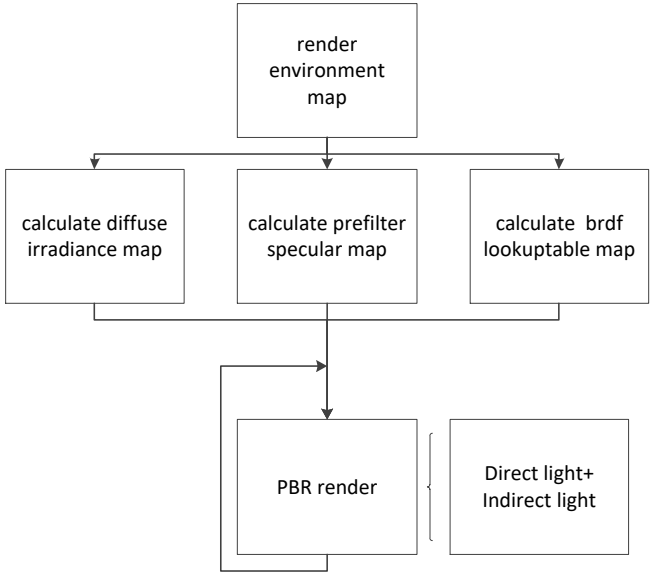
$$\int_{\Omega} \frac{DFG}{4(\omega_o \cdot n)(\omega_i \cdot n)} L_i(p,\omega_i)(\omega_i \cdot n) d\omega_i$$

$$= \int_{\Omega} L_i(p,\omega_i) d\omega_i \int_{\Omega} \frac{DFG}{4(\omega_o \cdot n)(\omega_i \cdot n)} (\omega_i \cdot n) d\omega_i$$

Input: Environment Cubemap

Output: Specular irradiance Cubemap

Image-based Lighting GI (IBL GI)



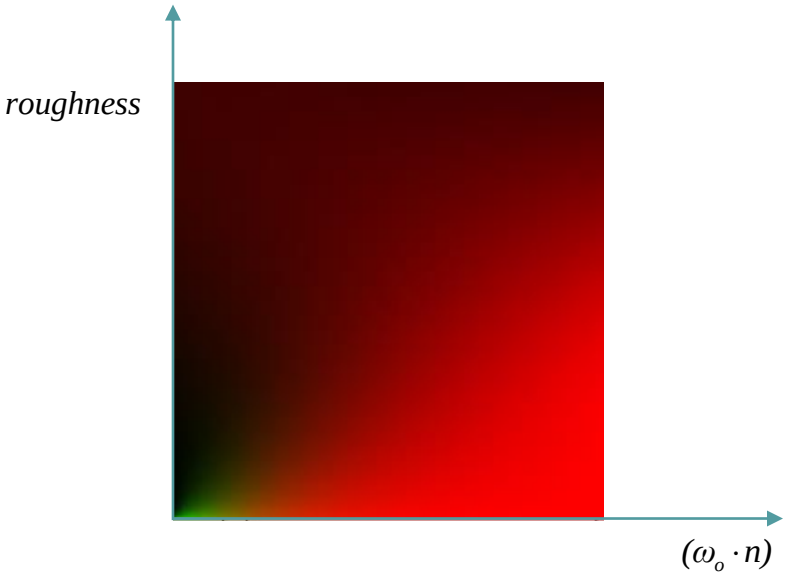
$$\text{brdf}(\omega_i, n, \alpha, \omega_o)$$

$$= \int_{\Omega} \frac{DFG}{4(\omega_o \times n)(\omega_i \times n)} (\omega_i \times n) d\omega_i$$

$$= F_0 \int_{\Omega} \frac{DG}{4(\omega_o \times n)(\omega_i \times n)} (1 - (1 - \omega_i \times h)^5)(\omega_i \times n) d\omega_i + \int_{\Omega} \frac{DG}{4(\omega_o \times n)(\omega_i \times n)} (1 - \omega_i \times h)^5 (\omega_i \times n) d\omega_i$$

brdf.r

brdf.g



brdf look up table

Input：fragment normal vector **N**，Camera View Vector **V**, fragment attribute: **roughness**,**albedo**, **ao**, **metallic**，Reflection Vector **R**, Diffuse irradiance Cubemap **irradianceMap**，Specular irradiance Cubemap **prefilterMap,**
BRDF look up table **brdfLUTMap**

Output：indirect light **indirectColor**

```
1   F =  FresnelSchlickRoughness(max(dot(N, V), 0.0), F0, roughness)
2   Ks = F
3   Kd = vec3(1.0)-Ks
4   Kd *= (1.0 - metallic)
5   environmentIrradiance = texture(irradianceMap,N).rgb;
6   diffuse = environmentIrradiance*albedo
7   MAX_LOD = 4.0
8   prefileredColor=sampler(prefilterMap,R,roughness*MAX_LOD).rgb
9   brdf = sampler(brdfLUTMap,vec2(max(dot(N,V),0.0),roughness)).rg
10  specular = prefileredColor * (F*brdf.x+brdf.y)
11  indirectColor = (Kd*diffuse+specular) * ao
```

image base lighting algorithm

Image-based Lighting GI (IBL GI)

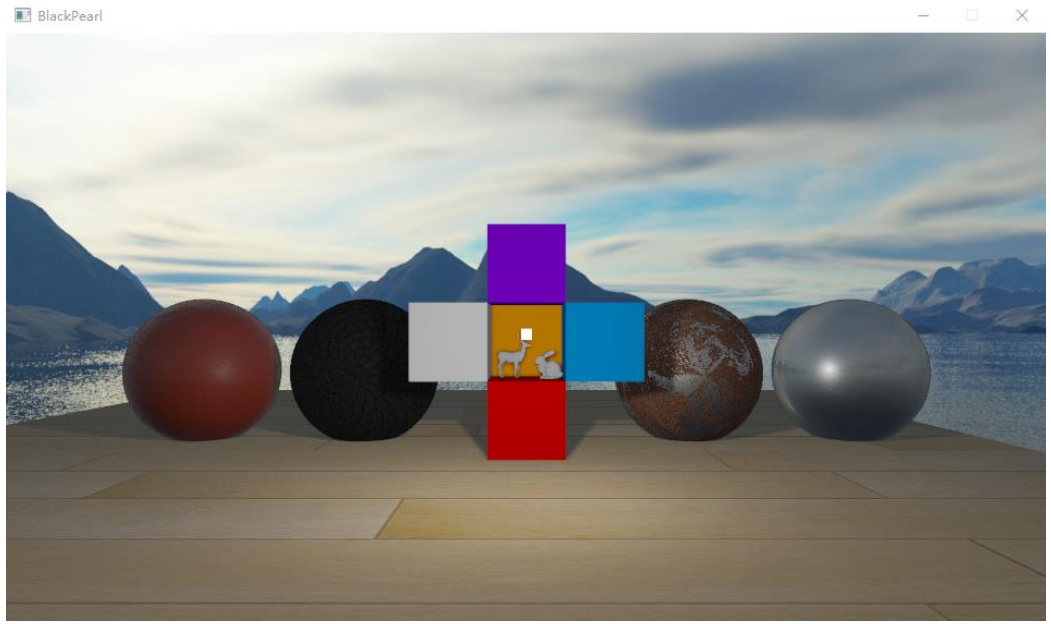
Experiment

Scene settings

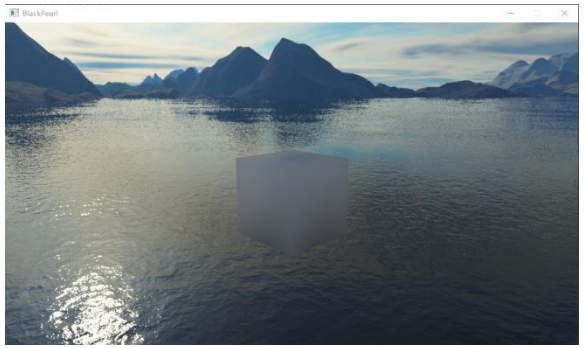
Object	position	scale	rotation	diffuse color	specular color
deer	-0.5,0.0,2.4	0.003	0,54.2,0.00	153,153,153	0
bunny	0.6,0.0,3.0	0.5	0,-30,0	153,153,153	0
cube1	2,1,3	1	0	0,75,255	0
cube2	0,1,1	1	0	255,68,0	0
cube3	-2,1,3	1	0	255,255,255	0
cube4	0,3,3	1	0	52,0,255	0
cube5	0,-1,3	1	0	255,0,0	0
point light	0,1.25,9.0	0.2	0	255,255,255	point light
camera	0,1.387,22.012	-	0,-90,0	-	camera
sphere1	10,0,0	1.5	0	PBR texture	
sphere2	5,0,0	1.5	0	PBR texture	
sphere4	-10,0,0	1.5	0	PBR texture	
plane	-2,-2,0	16,0,16	0	2D texture	
skybox	0,1.387,22.012	1	0,-90,0	Cube map	

Resources usage

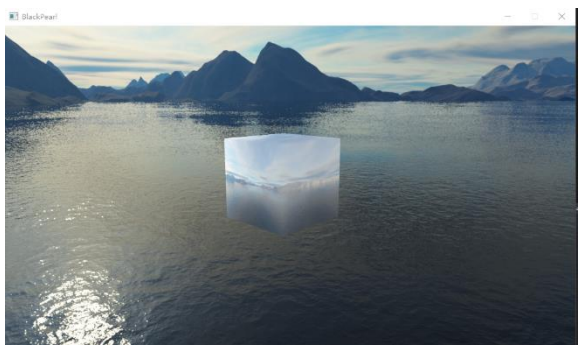
CPU %	GPU %	FPS frame/s	Memory MB	Video Memory MB
13.9	21.2	60.0	64.5	514.816



IBL GI results



diffuse irradiance cube map



mipmap = 1 prefilter specular cube map

Dynamic Global illumination Algorithm design and implementation

Rendering pipeline

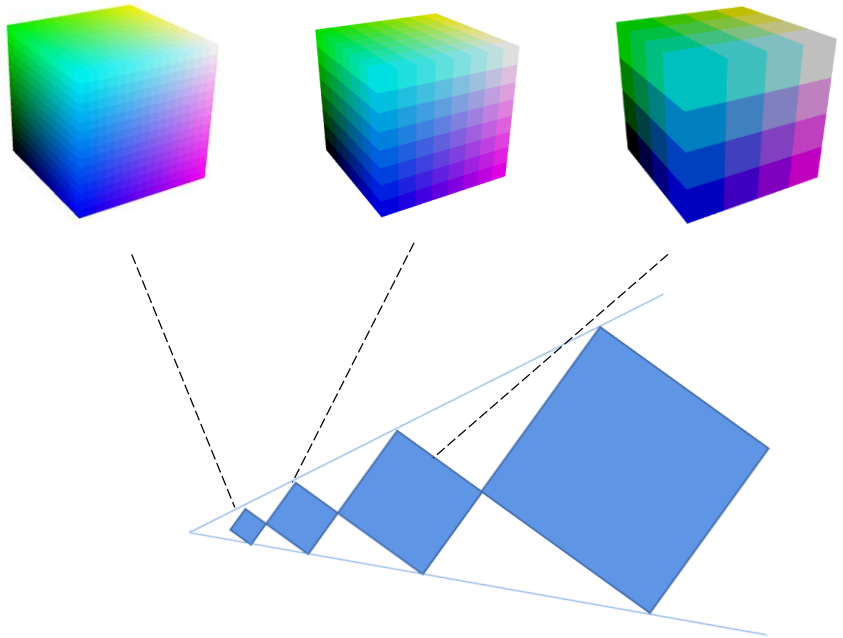
- Image based lighting GI
- Light probe-based GI
- Voxel cone tracing GI

Rasterization

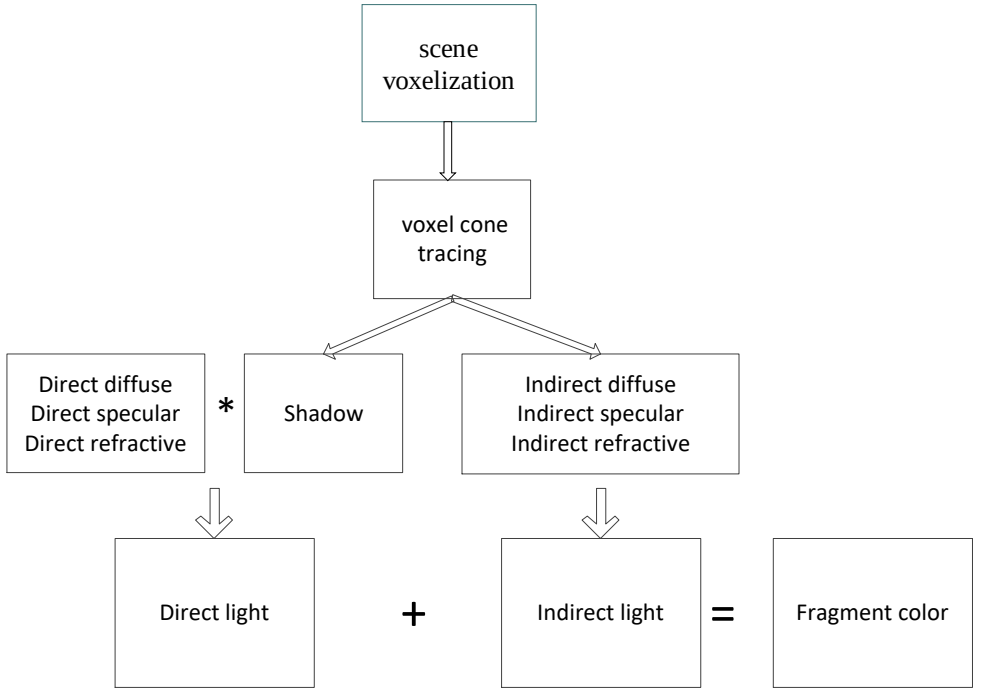
Ray Tracing Approximation

Voxel cone tracing GI Design and Implementation

Voxel structure selection: 3D mipmap texture



Mipmap structure



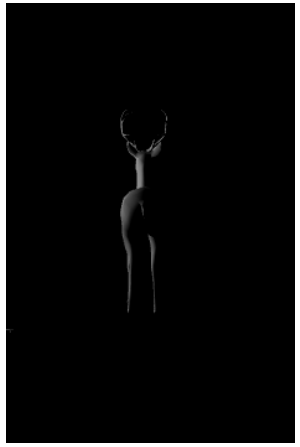
Voxel cone tracing GI rendering pipeline

VS Sparse voxel octree

1.Has no preprocessing (build tree and prefiltering)
2.Voxelization is fast and simpler for dynamic objects
(No tree reconstruction and prefiltering is required for dynamic objects)
3. Consuming more storage space

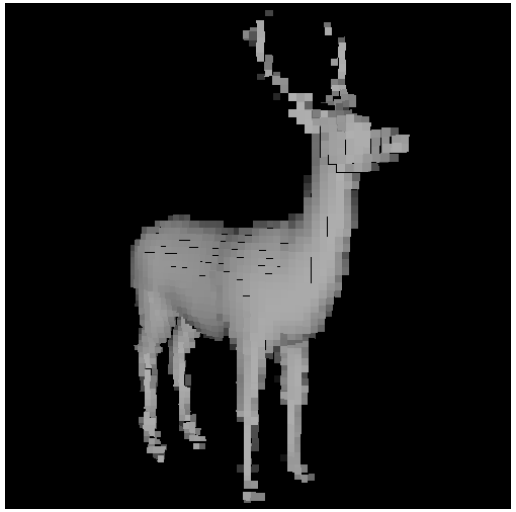
Voxel cone tracing GI Design and Implementation

Step1 : Voxelize the triangle meshes



3D model

Triangle mesh projection



Voxel visualization

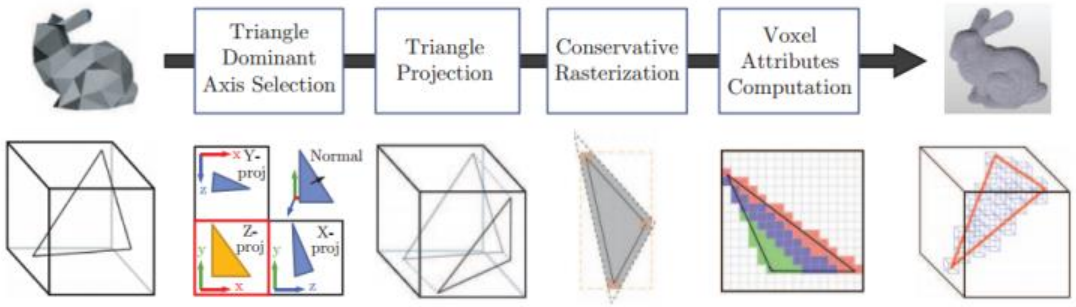


Illustration of the voxelization scheme described by Crassin et al. Image on courtesy of Cyril Crassin. [11]

Input from vertex shader: Triangle world position and normal vector **v_worldPosition[3], v_normal[3]**
Input from Uniform variables： Camera position **u_CameraViewPos**， Virtual Cube size **u_CubeSize**
Output： output to fragment shader: **g_worldPosition, g_normal**

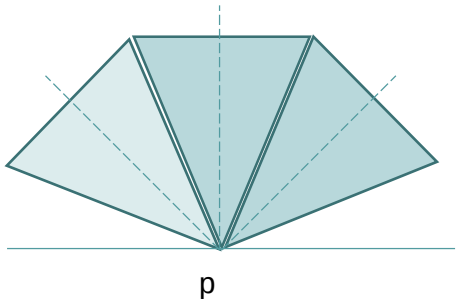
```
1   p = abs(cross(v_worldPosition[1] - v_worldPosition[0], v_worldPosition[2] - v_worldPosition[0]));
2   for i in range(0,2)
3       g_worldPosition=v_worldPosition[i]
4       worldPosition=(worldPosition[i]-u_CameraViewPos)/u_CubeSize
5       g_normal= normal[i]
6       If p.z > p.x and p.z > p.y
7           gl_Position = vec4(worldPosition.x, worldPosition.y, 0, 1)
8       else if  p.x > p.y and p.x > p.z
9           gl_Position = vec4(worldPosition.y,worldPosition.z, 0, 1)
10      else
11          gl_Position = vec4(worldPosition.x, worldPosition, 0, 1)
12      EmitVertex()
13  EndPrimitive()
```

Voxelization algorithm in geometry shader

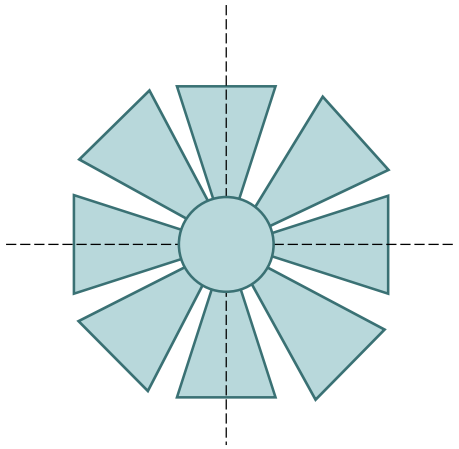
Voxel cone tracing GI Design and Implementation

Step2 : Voxel Cone tracing

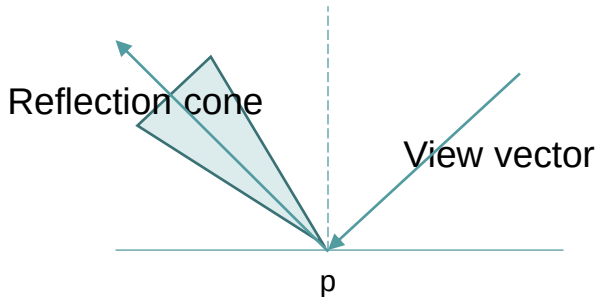
Diffuse cone tracing



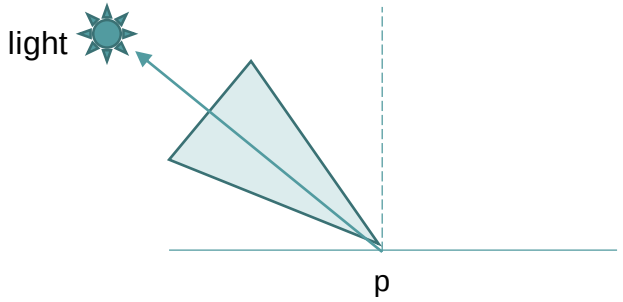
9 cones

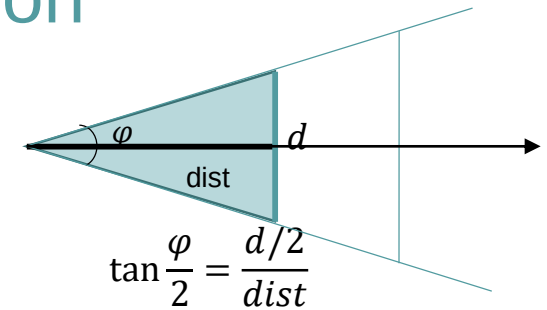


Specular cone tracing



Shadow cone tracing





$$\tan\frac{\varphi}{2} = \frac{d/2}{dist}$$

$$Mipmap\ level = log_2(\frac{d}{voxel\ size})$$

$$voxel\ size = \frac{1.0}{3D\ texture\ resolution}$$

Input: fragment position in unit cube **fragment_position**，cone direction **direction**，3D texture **texture3D**，voxel size **voxel_size,** The number of voxel steps taken ahead of time voxel_size,**count_offset**

Output: indirect color(diffuse or specular) **color**

```
1   tanHalfAngle = tan(radians(coneAngle/2.0)),
2   offset =voxel_size
3   origin = fragment_position + offset*direction
4   color = 0,dist=count_offset* voxel_size
3   distance= length(vec3(1.0))
4   while t<distance and color.alpha<1 then
5       ray = origin + dist*direction
6       if ray is not inside unit cube
7           break
8       diameter = max(voxel_size,2.0* tanHalfAngle*dist)
9       level = log2(diameter / voxel_size)
10      voxel = sampler(texture3D,ray,level)
11      color = color + (1-color.alpha)*voxel
12      dist+ = 0.5*diameter
13  return color
```

Voxel Cone tracing algorithm

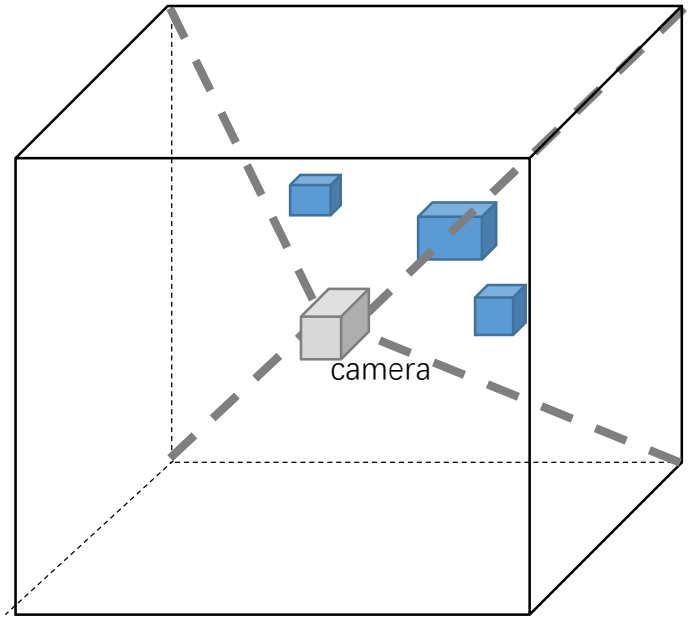
Voxel cone tracing GI Optimization

Optimization More elaborate quality

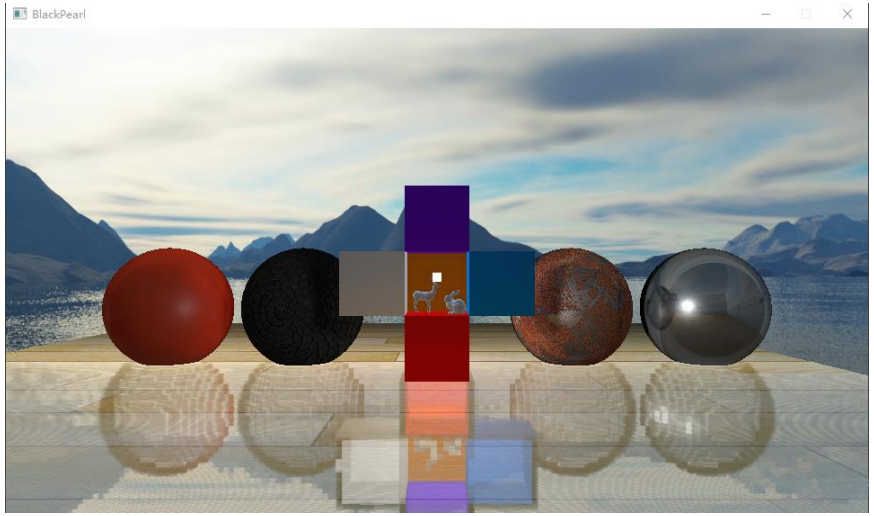
1.Memory and DrawCall number optimized: Only voxelize objects around the camera

2.Rendering Quality optimized: improve Glossy Reflection quality by applying gaussian filtering to voxels

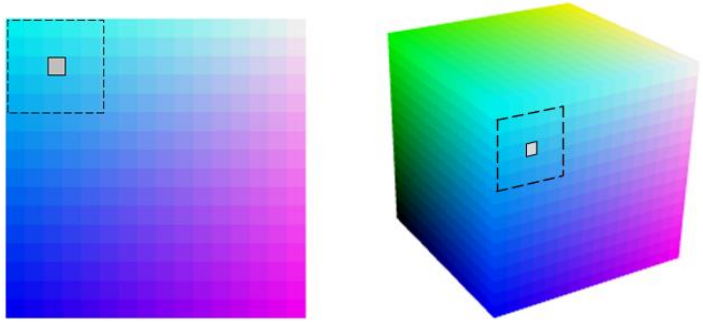
Virtual Cube



Camera centered voxelizatioin



Due to the **discrete** nature of the voxel storage, the mirror-like material produces a discrete rendering effect.
As the cone angle size decreases, the geometry visibility is more obvious

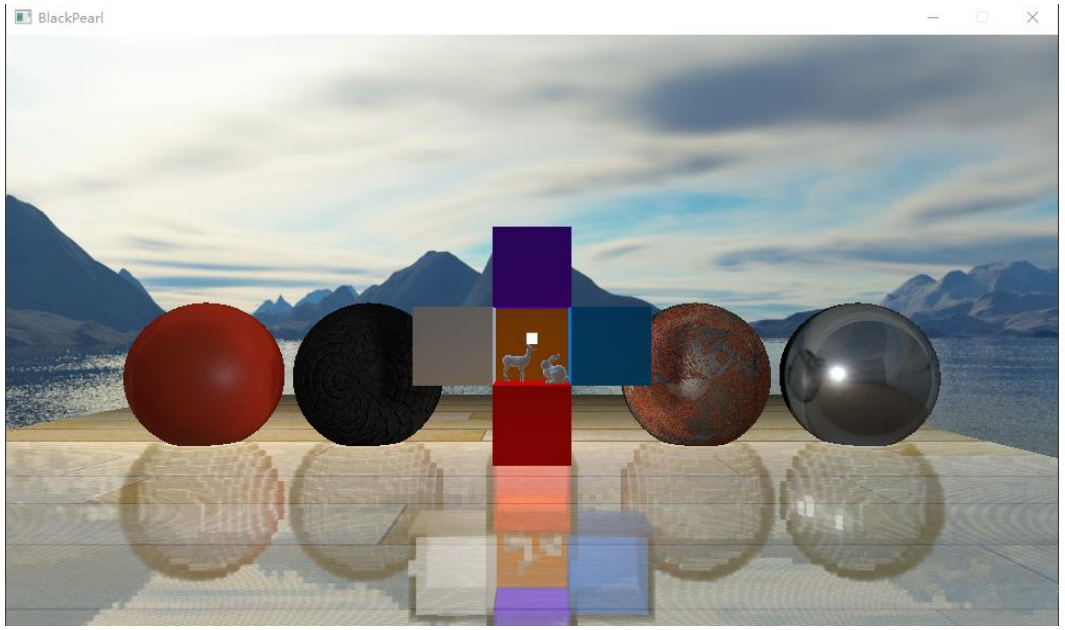


Apply a Gaussian filter to 3D mipmap texture

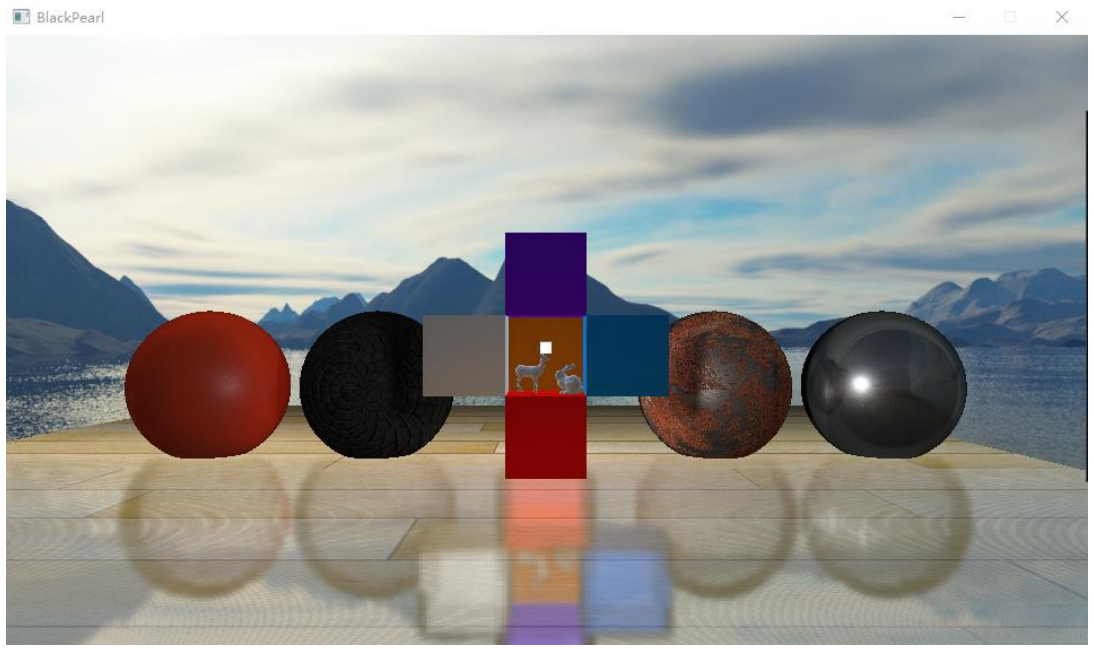
Use a Gaussian kernel of size 5

Voxel cone tracing GI Optimization

Optimization
1.Memory and DrawCall number optimized: Only voxelize objects around the camera
2.Rendering Quality optimized: improve Glossy Reflection quality by applying gaussian filtering to voxels



Without Gaussian Blurred



With Gaussian Blurred

This results in significantly better quality and smoother transitions at geometric discontinuities!

Voxel cone tracing GI Optimization

3.Reduce voxel flickering phenomenon

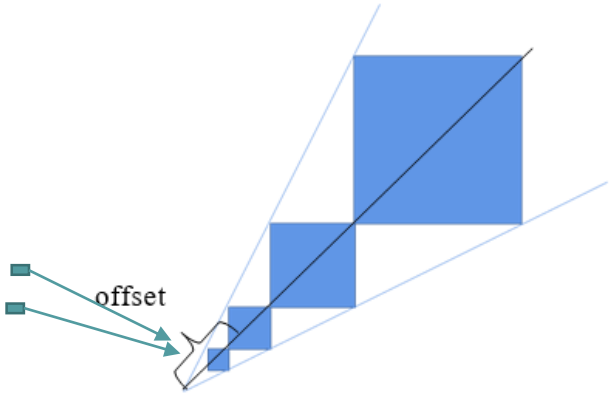
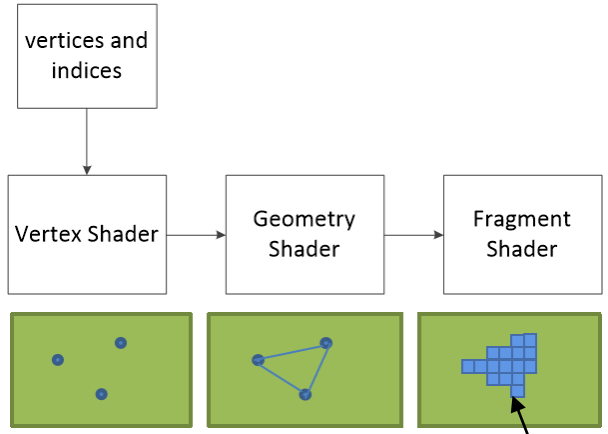
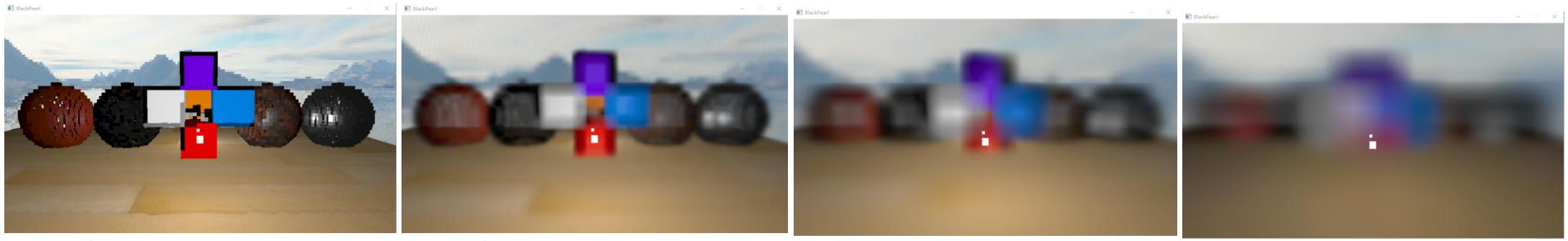


Figure 4- 9 Cone tracing offset value offset

Due to the voxel resolution, two fragments will be projected onto the same voxel. However, the operation of fragment shader is not sequential.
As a result, when two consecutive frames are voxelized, flickering will occur in some small objects or the boundary of objects.
Therefore, when voxel cone tracing, If the mipmap level reached by the light is small, these randomly colored voxels will be collected.



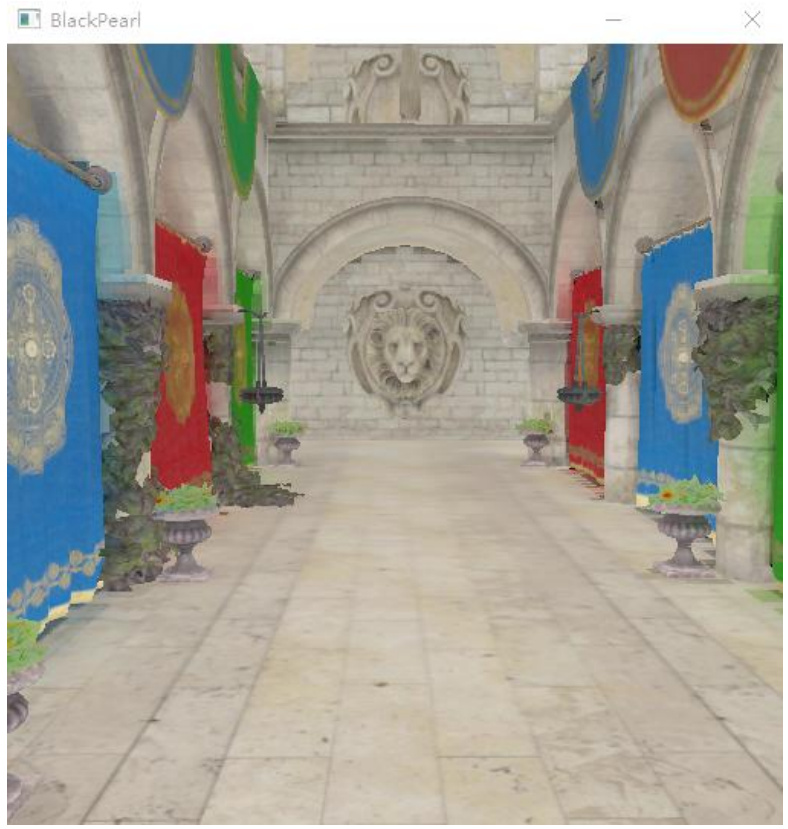
Parallel process



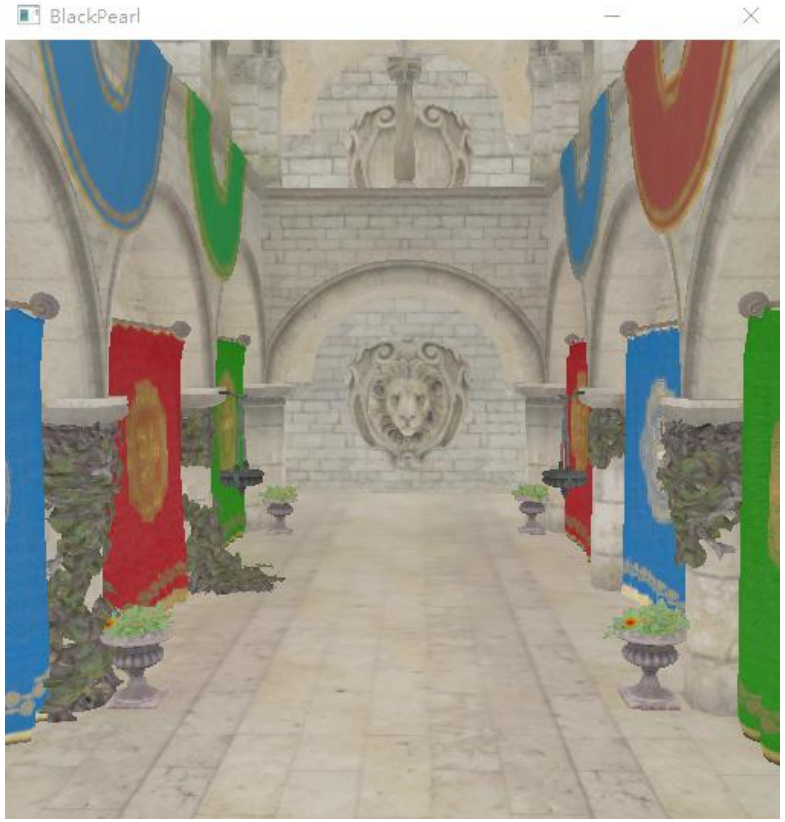
（a）mipmap level = 0　（b）mipmap level = 1　(c) mipmap level =2　(d) mipmap level = 3

Figure 4-10 3D texture mipmap level from low to high voxelized image

Voxel cone tracing GI Optimization



Flickering phenomenon



Add offset value, no flicker

Voxel cone tracing GI Experiment

Rendering scene settings

scene1

Object	position	scale	rotation	diffuse color	specular color
deer	-0.5,0.0,2.4	0.003	0,54.2,0.00	153,153,153	0
bunny	0.6,0.0,3.0	0.5	0,-30,0	153,153,153	0
cube1	2,1,3	1	0	0,75,255	0
cube2	0,1,1	1	0	255,68,0	0
cube3	-2,1,3	1	0	255,255,255	0
cube4	0,3,3	1	0	52,0,255	0
cube5	0,-1,3	1	0	255,0,0	0
point light	0.0,0.6,3.8	0.1	0	255,255,255	0
camera	0.0,0.95,5.9	-	0,-90,0	-	-

scene2

Object	position	scale	rotation	diffuse color	specular color
deer	-0.5,0.0,2.4	0.003	0,54.2,0.00	153,153,153	0
bunny	0.6,0.0,3.0	0.5	0,-30,0	153,153,153	0
cube1	2,1,3	1	0	0,75,255	0
cube2	0,1,1	1	0	255,68,0	0
cube3	-2,1,3	1	0	255,255,255	0
cube4	0,3,3	1	0	52,0,255	0
cube5	0,-1,3	1	0	255,0,0	0
point light	0,1.25,9.0	0.2	0	255,255,255	point light
camera	0,1.387,22.012	-	0,-90,0	-	camera
sphere1	10,0,0	1.5	0	PBR texture	
sphere2	5,0,0	1.5	0	PBR texture	
sphere3	-5,0,0	1.5	0	PBR texture	
sphere4	-10,0,0	1.5	0	PBR texture	
plane	-2,-2,0	16,0,16	0	2D texture	
skybox	0,1.387,22.012	1	0,-90,0	Cube map	

scene3

Object	position	scale	rotation	diffuse color	specular color
point light	0,1.25,9.0	0.2	0	255,255,255	point light
camera	0,1.387,22.012	-	0,-90,0	-	camera
church	0,0,10	0.01	0,-90,0	2D texture	
skybox	0,1.387,22.012	1	0,-90,0	Cube map	

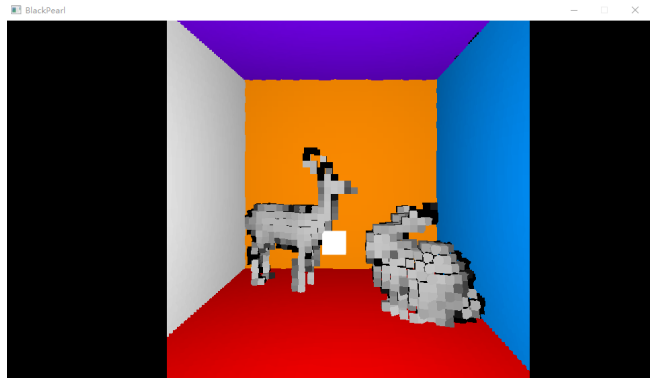
Scene	Triangles number
scene1	6598
scene2	7624
scene3	262267

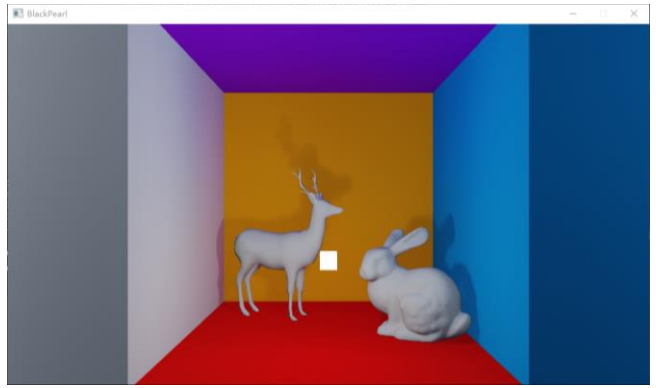
Voxel cone tracing GI Experiment

scene1

Cube size = 10

CPU %	GPU %	FPS frame/s	Memory MB	Video Memory MB
5.1	27.3	54.1	45.91	246.53

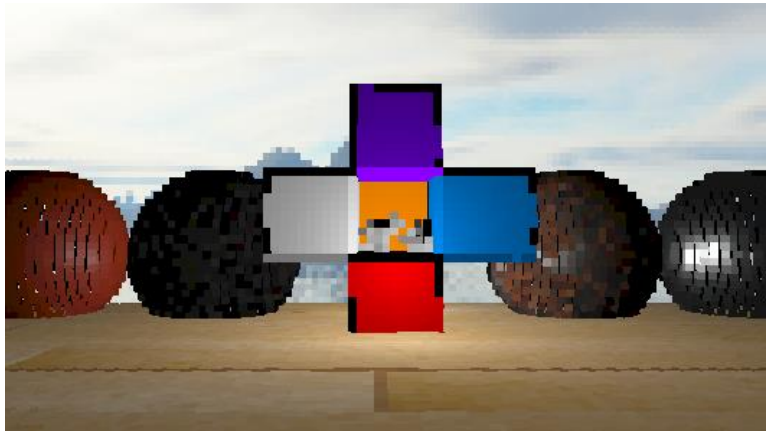


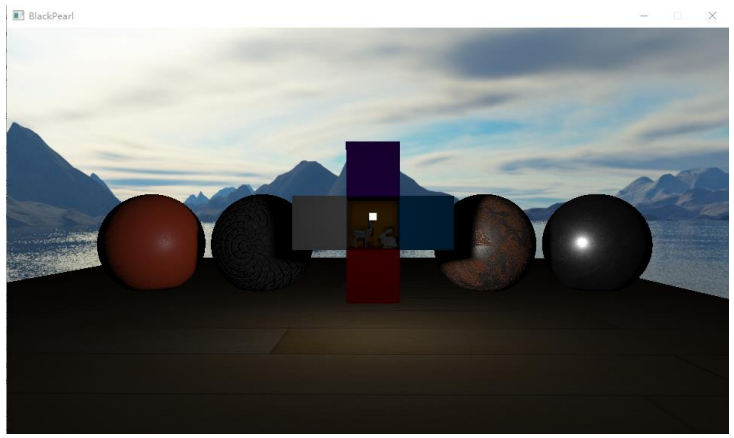


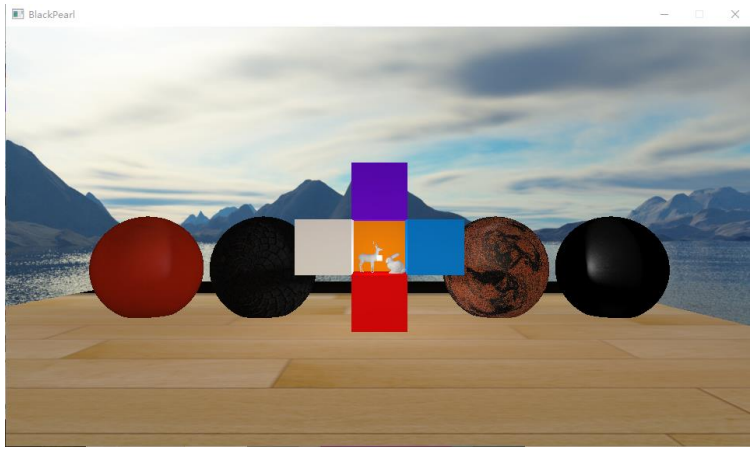
scene2

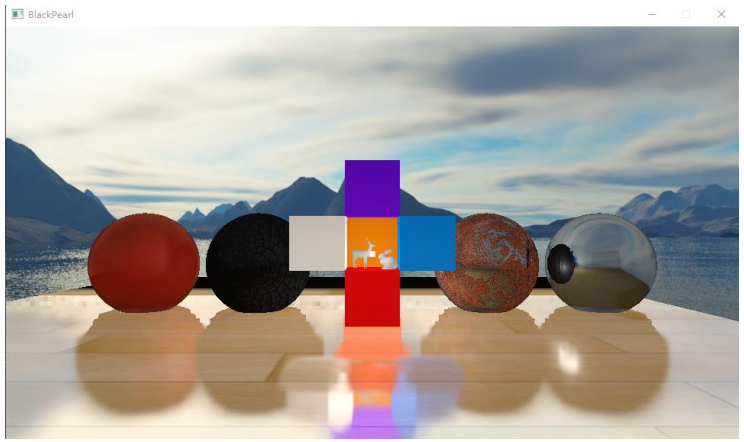
Cube size = 30

CPU %	GPU %	FPS frame/s	Memory MB	Video Memory MB
11.8	49.4	44.5	43.03	582.06









Voxel cone tracing GI Experiment

scene3

Cube size = 35

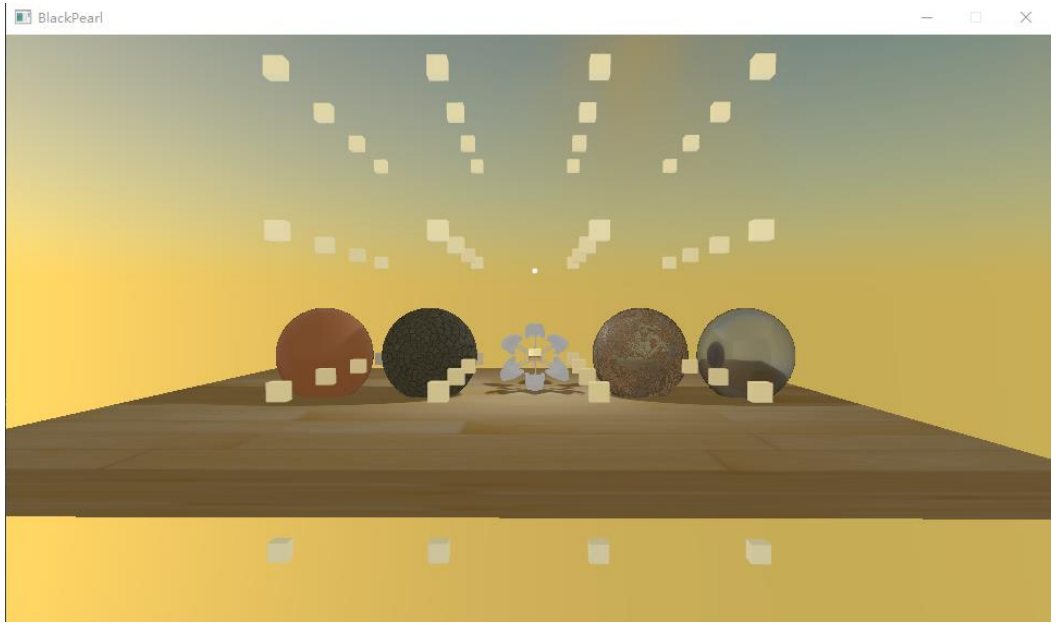
CPU %	GPU %	FPS frame/s	Memory MB	Video Memory MB
16.2	44.6	9	118.74	627.269

Light probe-based GI theory

Image based lighting GI →**one** environment map for the whole scene → Imprecise for large scene GI

Light probe-based GI →**multiple compressed** environment maps

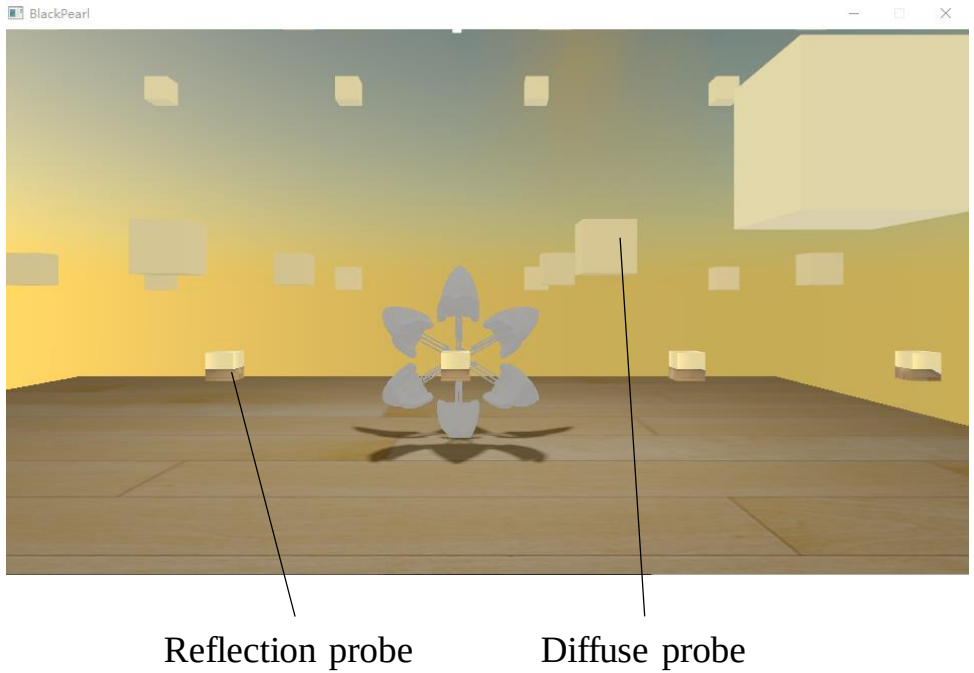
Ideas:
1.Store GI information in multiple compact data structure – light probe
2.Get GI information by interpolation from probes around the object



Light probe-based GI – probe structure design

probe structure design

Diffuse probe	Reflection probe
9 spherical harmonic coefficients	specular irradiance cubemap



storage space optimization:

Spherical harmonic function encodes indirect diffuse light

diffuse irradiance cubemap → 9 spherical harmonic coefficients

256*256*6*RGB16F =2.25MB

9*RGB16F=54B

According to “An Efficient Representation for Irradiance EnvironmentMaps” ,SIGGRAPH01[12]

Diffuse part of the rendering function
can be represented by 9 spherical harmonic coefficients

$$\int_{\Omega} L_i(p,\omega_i)(\omega_i \cdot n)d\omega_i$$

$$\mathrm{E}(n) = \int_{\Omega} L(\omega)(\omega \cdot n)d\omega$$

Convert to the spherical coordinate system and expand the spherical harmonic function of order 3:

$$\mathrm{E}(\theta,\phi) = \sum_{l,m} E_{lm} Y_{lm}(\theta,\phi)$$

$$l \le 2, -l \le m \le l$$

$$Y_{00}(\theta,\phi)$$

$$Y_{1,-1}(\theta,\phi),\ Y_{1,0}(\theta,\phi),\ Y_{1,1}(\theta,\phi)$$

$$Y_{2,-2}(\theta,\phi),\ Y_{2,-1}(\theta,\phi),\ Y_{2,0}(\theta,\phi),\ Y_{2,1}(\theta,\phi),\ Y_{2,2}(\theta,\phi)$$

Light probe-based GI – probe structure design

According to the literature [13,14,15], the nine spherical harmonic functions can be expressed in the cartesian coordinate system as follows:

$$(x, y, z) = (\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\theta)$$

$$Y_{00}(\theta,\phi) = 0.282095$$

$$Y_{1,1}(\theta,\phi), Y_{1,0}(\theta,\phi), Y_{1,-1}(\theta,\phi) = 0.488603(x; z; y)$$

$$Y_{2,1}(\theta,\phi), Y_{2,-1}(\theta,\phi), Y_{2,-2}(\theta,\phi) = 1.092548(xz; yz; xy)$$

$$Y_{2,0}(\theta,\phi) = 0.315392(3z^2-1)$$

$$Y_{2,2}(\theta,\phi) = 0.546274(x^2 - y^2)$$

$$let\ A = (\omega \cdot n)$$

$$A(\theta) = \max(\cos\theta, 0) = \sum_l A_l Y_{l0}(\theta)$$

$$\mathrm{E}(\theta,\phi) = \sum_{l,m} \hat{A}_l L_{lm} Y_{lm}(\theta,\phi)$$

$$A_l = 2\pi \int_0^{\pi/2} Y_{l0}(\theta)\cos(\theta)\sin(\theta)d\theta$$

$$A_l = 2\pi\sqrt{\frac{2l+1}{4\pi}}\int_0^1 P_l(\mu)P_1(\mu)d\mu$$

$$l = 1, A_l = \sqrt{\frac{\pi}{3}}$$

$$l > 1 \quad odd, A_l = 0$$

$$l \quad even, A_l = 2\pi\sqrt{\frac{2l+1}{4\pi}}\frac{(-1)^{l/2-1}}{(l+2)(l+1)} \times \left(\frac{l!}{2^l (l!/2)^2}\right)$$

$$\hat{A}_l = \sqrt{\frac{4\pi}{2l+1}} A_l$$

$$\hat{A}_0 = 3.14159, \hat{A}_1 = 3.14159, \hat{A}_2 = 3.14159$$

$$c_1 = 0.282095\hat{A}_0$$

$$c_2 = 0.488603\hat{A}_1$$

$$c_3 = 1.092548\hat{A}_2$$

$$c_4 = 0.315392\hat{A}_2$$

$$c_5 = 0.546274\hat{A}_2$$

c1=0.886288， c2=1.023328,c3=0.858086,c4=0.247708,c5=0.429043

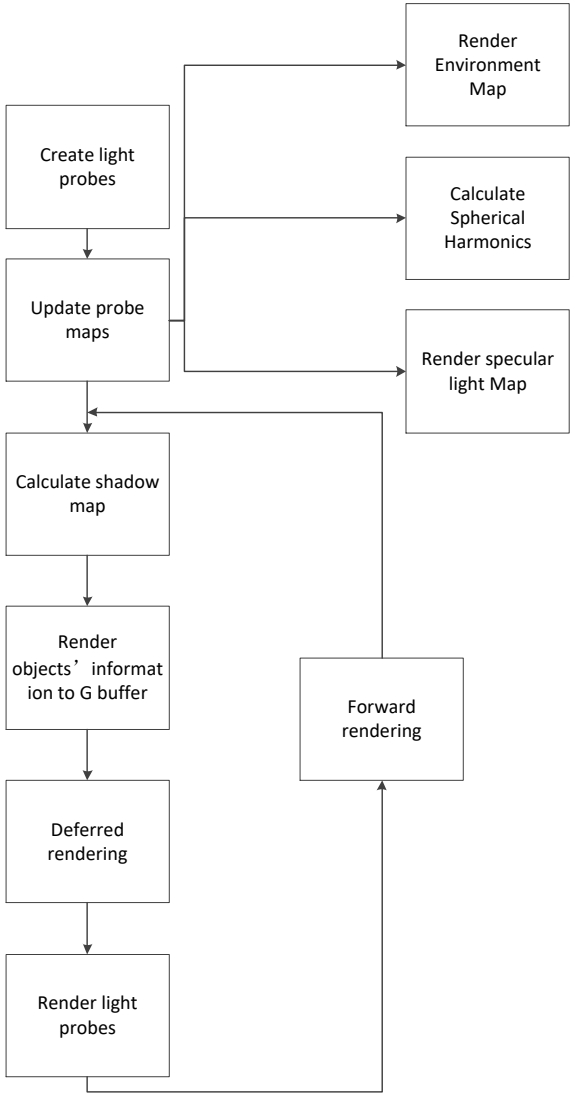
$$\mathrm{E}(n) = c_1 L_{0,0} + c_2(L_{1,1}x + L_{1,0}z + L_{1,-1}y) + c_3(L_{2,1}xz + L_{2,-1}yz + L_{2,-2}xy) + c_4 L_{2,0}(3z^2-1) + c_5 L_{2,2}(x^2 - y^2)$$

$$\mathrm{L}_{lm} = \int_{\theta=0}^{\pi}\int_{\phi=0}^{2\pi} L(\theta,\phi)Y_{lm}(\theta,\phi)\sin\theta d\theta d\phi$$

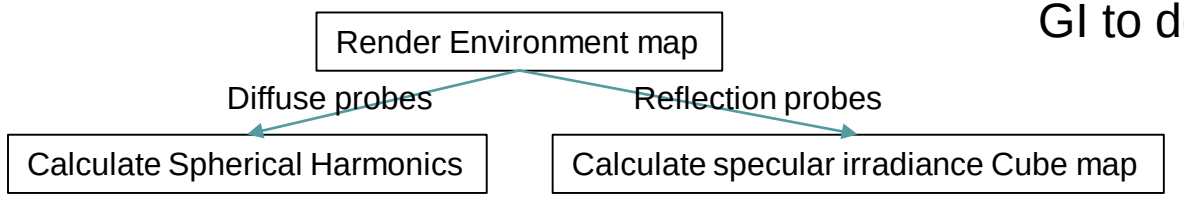
Input: get $L(\theta, \varphi)$ from Environment cube map

Output: 9 spherical harmonic coefficients: $L_{0,0}, L_{1,1}, L_{1,0}, L_{1,-1}, L_{2,1}, L_{2,-1}, L_{2,-2}, L_{2,0}, L_{2,2}$

Light probe-based GI – Rendering pipeline design



Light probe-based GI pipeline design



Combine light probe GI to deferred rendering

1.DrawCall number optimization: Deferred Rendering

Only draw the fragment we can see in screen space

Do not need to calculate the GI for each fragment!

Forward rendering → O(Objects number * light number)

Deferred rendering → O(Objects number + light number)

Only the fragment we can see in Screen

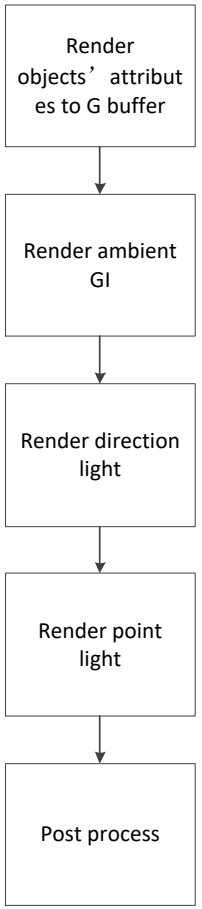
Store objects attributes in multiple 2D textures

G buffer

Material attribute	Data type	attachment	target
position	R16FG16FB16F	ATTACHMENT0	TEXTURE_2D
normal	R16FG16FB16F	ATTACHMENT1	
diffuse color + roughness	R16FG16FB16F+A16F	ATTACHMENT2	
specular color + metallic	R16FG16FB16F+A16F	ATTACHMENT3	
ambientGI + ao	R16FG16FB16F+A16F	ATTACHMENT4	
normal map	R16FG16FB16F+A16F	ATTACHMENT5	
depth buffer	24F	DEPTH_ATTACHMENT	RENDERBUFFER

Light probe-based GI – Optimization

Deferred rendering passes



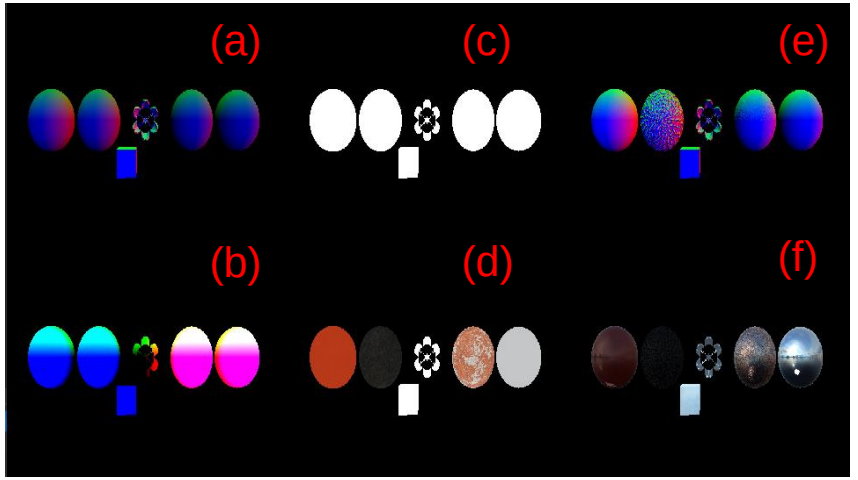
Input: weight array for k light probes **u_ProbeWeight[],**
fragment normal vector **N**,fragment color **albedo**,
Number of adjacent lightprobe **k**
Output: indirect diffuse color **Indirect_diffuse**

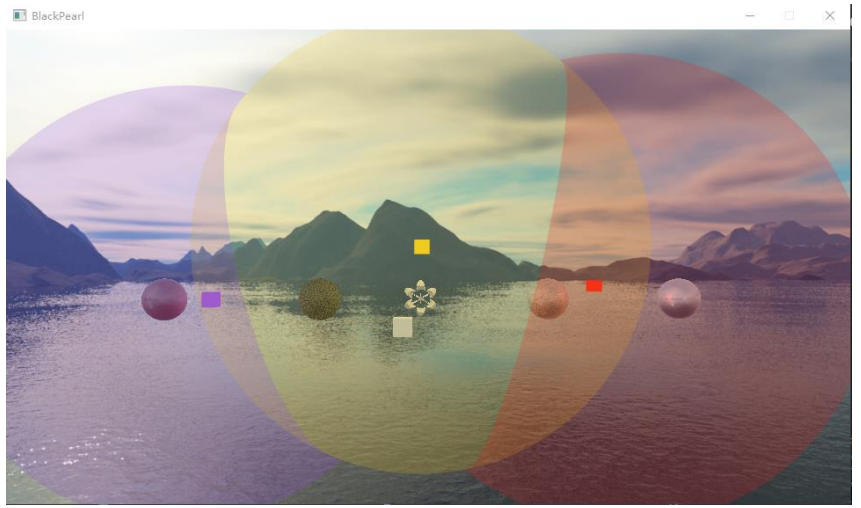
```
diffuse_irradiance=vec3(0.0)
for i in range(0, k)
    diffuse_irradiance+=u_ProbeWeight[i]*SHDiffuse(i,N)
end
Indirect_diffuse = diffuse_irradiance * albedo
```

```
1  Get Objects'Attribute from GBuffer
2  Draw(quad,direction_light_shader)
```

HDR and tone mapping

(a) normal;(b) position ;(c)Specular color + metallic;
(d)Diffuse color + roughness ;(e) normalMap(f)AmbientGI + ao






```
1  Enable(blend)
2  for each point light
3      Calculate bounding sphere's radius
4      Draw(bounding_sphere,pointlight_pass_shader)
5  end
```

Light probe-based GI – Probe selection improvement

1.Area management

Each area maintains a probe list

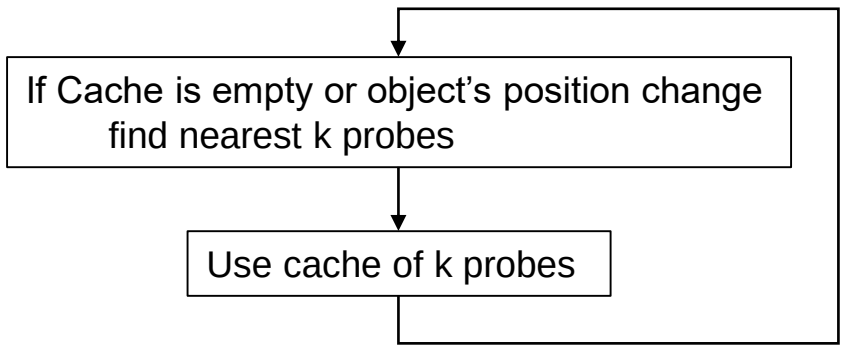
For each object

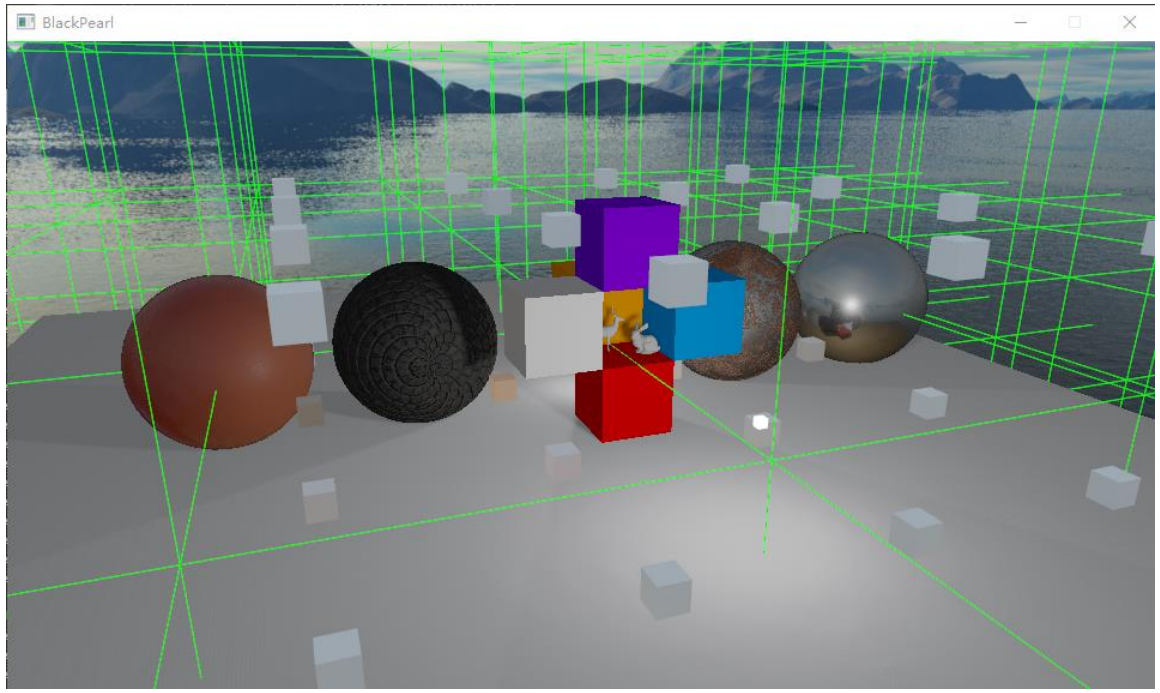
Find the k probes closest to the object

↓

Only sort probes in nearby areas

2.Cache



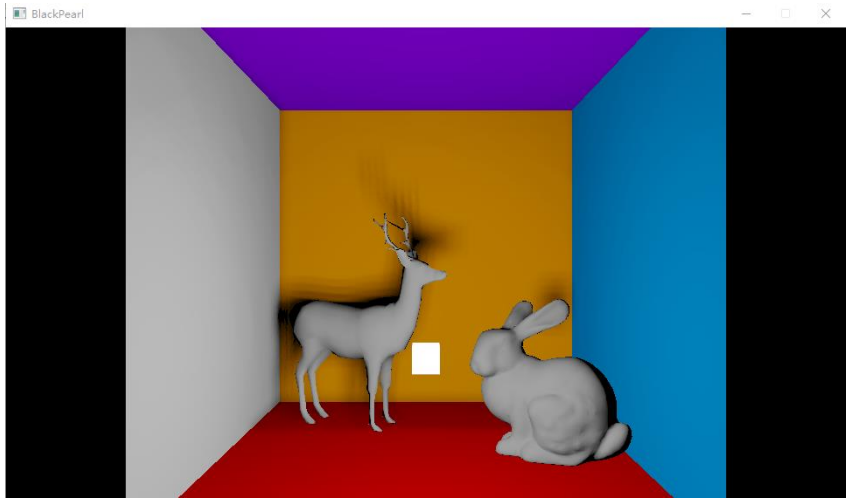


Area partition

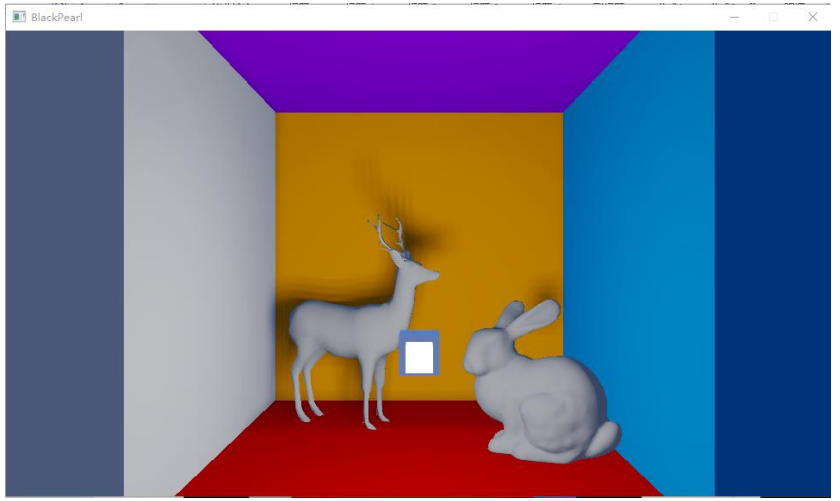
Light probe-based GI – Experiment

diffuse light probe： 1
Scene size ： 3*3*3 m^3

CPU %	GPU %	FPS frame/s	Memory MB	Video Memory MB
12.6	14.2	59.15	55.7	116.49



Light probe off

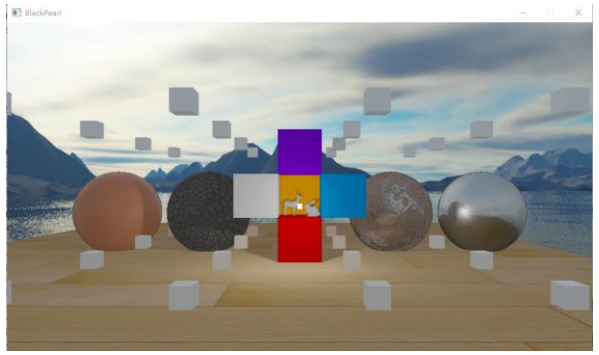


Light probe on

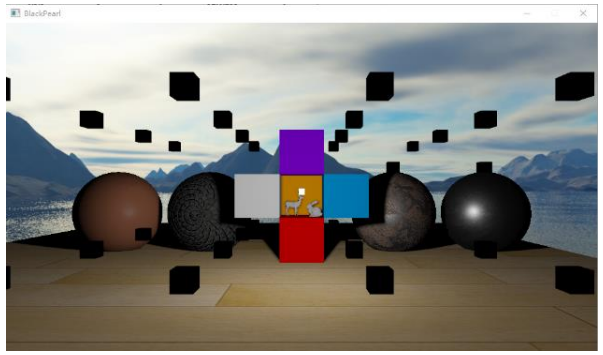
Light probe-based GI – Experiment

diffuse light probe grid：4*2*4
specular light probe grid：3*1*1
light probe space ： 5
Scene size：16*16*16 m^3

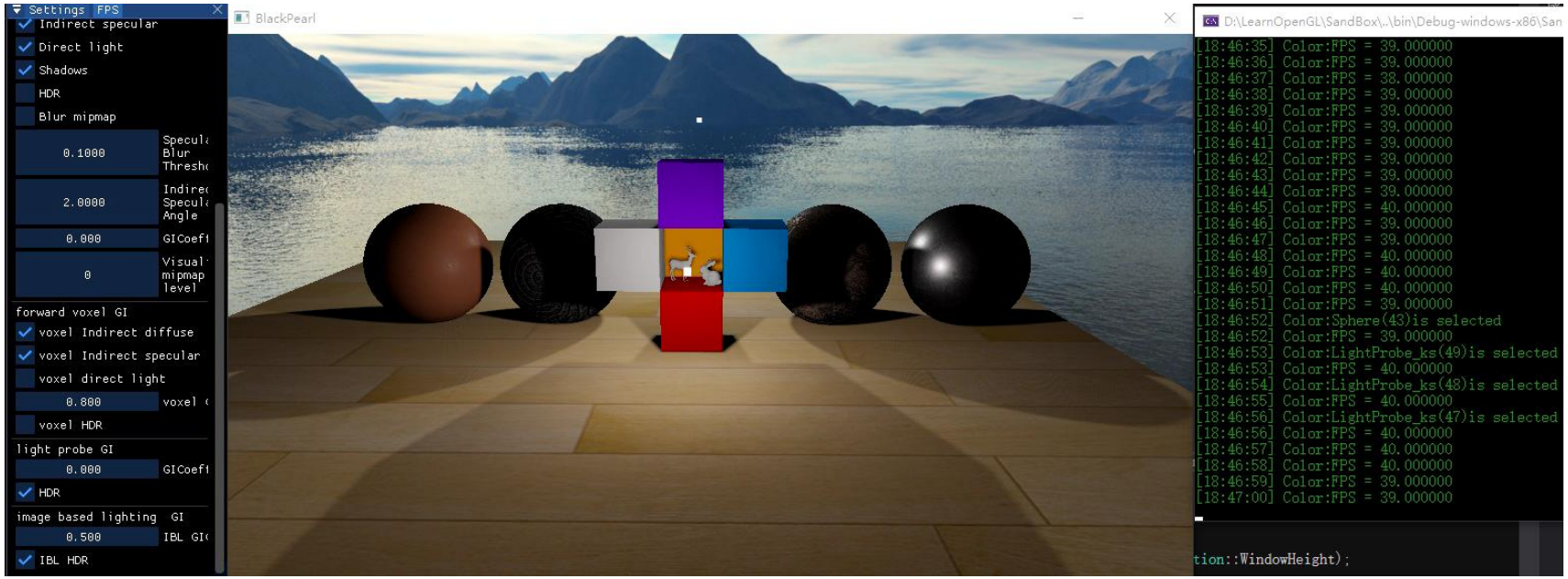
CPU %	GPU %	FPS frame/s	Memory MB	Video Memory MB
15.8	4.5	40.00	63.0	543.37

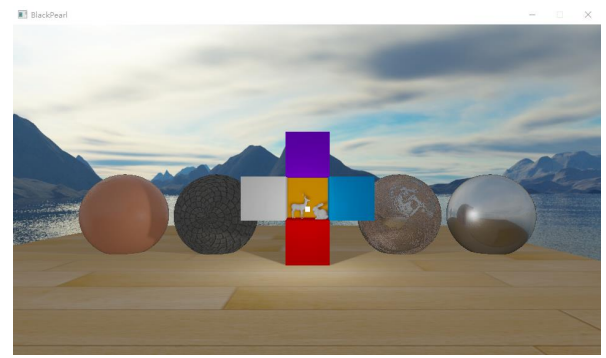


Light probe grid



Light probe off





Light probe on

tricks : update shadow map and specular probe only when objects or light changing!

Light probe-based quality improvement

1. Update a diffuse probe each frame
2. Add specular probe to each specular texture object

Enable

Dynamic Indirect light:

Changing environment light--→day and night

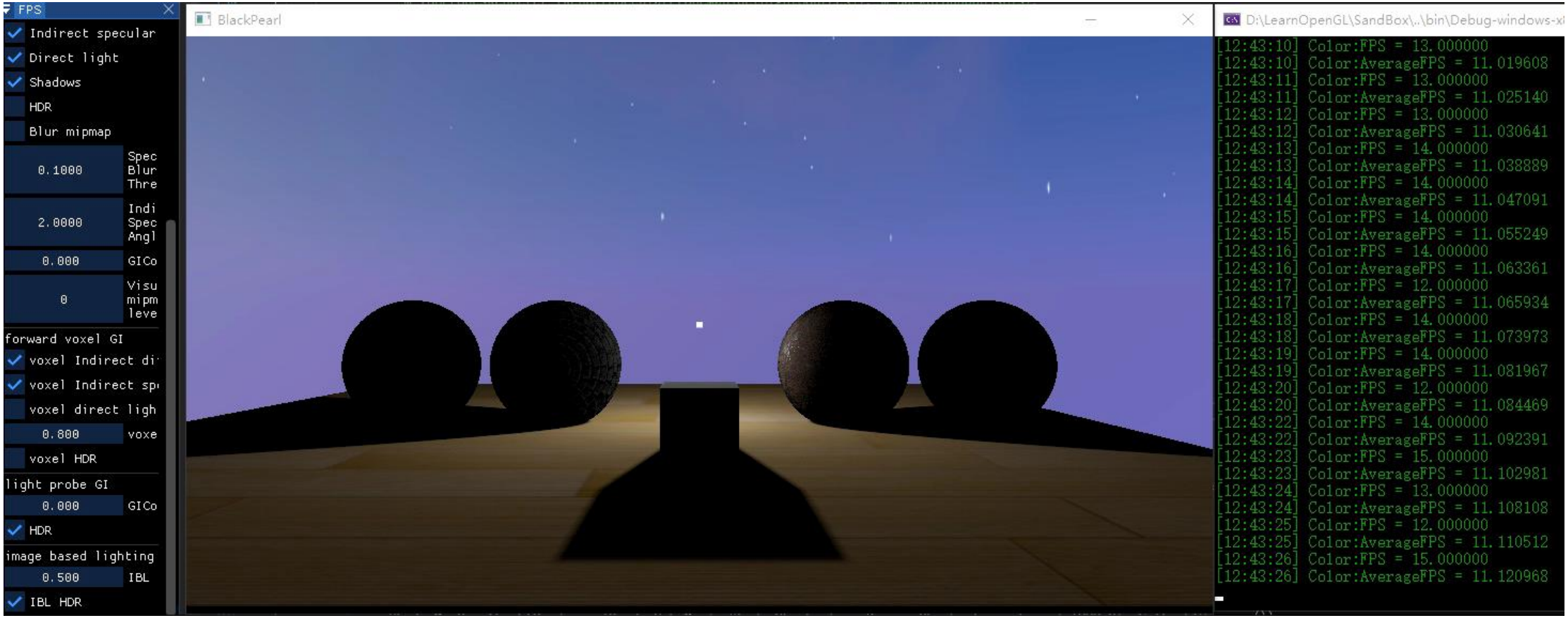
Moving specular object

can handle **semi-real-time** dynamic ambient lighting scenes

dynamic light source [1]

Light probe-based quality improvement

Dynamic indirect diffuse light+ dynamic indirect specular light



CPU %	GPU %	FPS frame/s	Memory MB	Video Memory MB
6.6	24.0	14.1	58.16	634.77

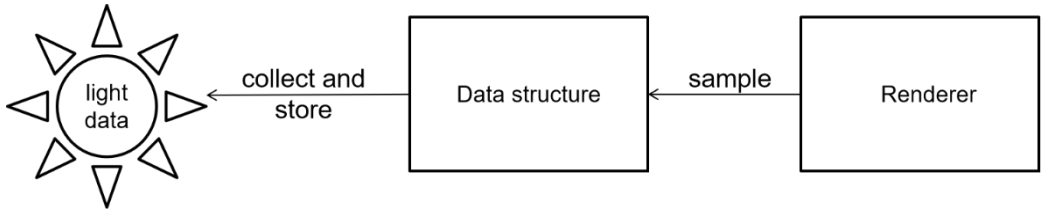
Dynamic GI Algorithm Comparison

1. Data structure
2. Real-time performance
3. Rendering quality
4. Applicable scenarios

Comparison is based on BlackPearl Rendering Engine and the three kinds of Dynamic GI this thesis designed!

GI Algorithm Comparison--Data structure

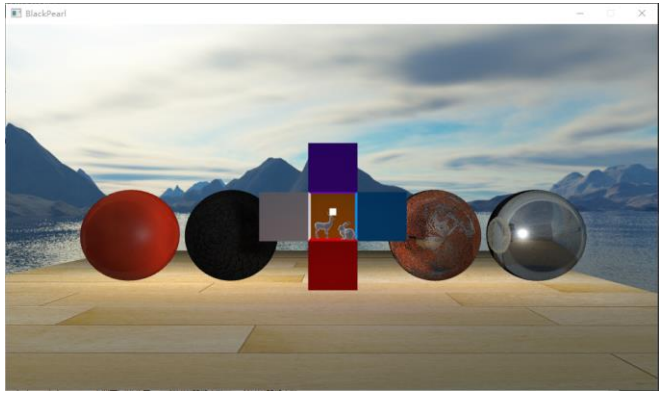
All three algorithms have something in common:



Theoretical value of storage space

Dynamic global illumination algorithm	Data structure that stores light information	Video memory of each structure	Total Video memory
voxel cone tracing GI	5 LOD 256^3 RGBA8 mipmap 3D texture	73.140MB	73.140MB
image base lighting GI	256^3 RGB16F Indirect diffuse irradiance cube map	2.25MB	5.5MB
	5 LOD mipmap 256^3 RGB16F Prefilter specular cube map	3MB	
	256^2 RG16F BRDF look up table 2D texture	256KB	
light probe based GI	5 LOD mipmap 256^3 RGB16F Prefilter specular cube map	3MB	Ks*3MB+Kd*54B (Ks- number of reflection probe ,Kd- number of diffuse probe)
	9* RGB16F Shpere Harmonic coefficients	54B	

Experiment results



voxel cone tracing GI

CPU %	GPU %	FPS frame/s	Memory MB	Video Memory MB
11.8	49.4	44.5	43.03	582.06

image base lighting GI

CPU %	GPU %	FPS frame/s	Memory MB	Video Memory MB
13.9	21.2	60.0	64.5	514.816

light probe based GI

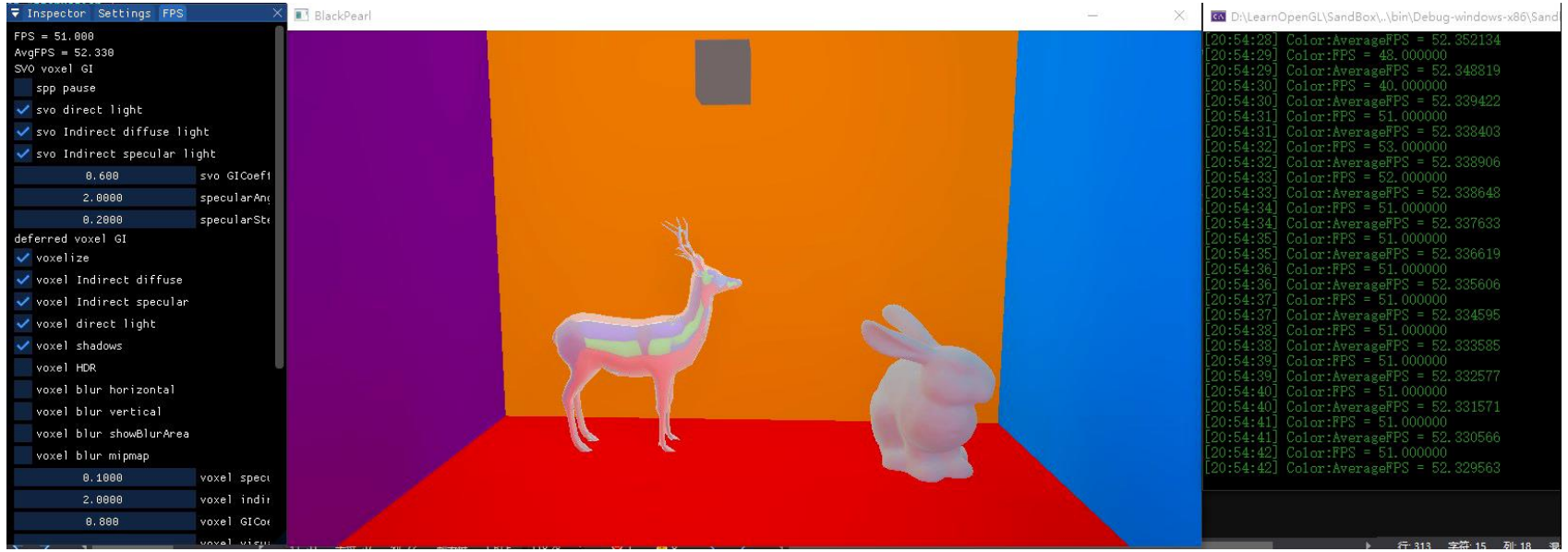
CPU %	GPU %	FPS frame/s	Memory MB	Video Memory MB
15.8	4.5	21.00	63.0	543.37

GI Algorithm Comparison--Real-time performance

Real-time performance contrast

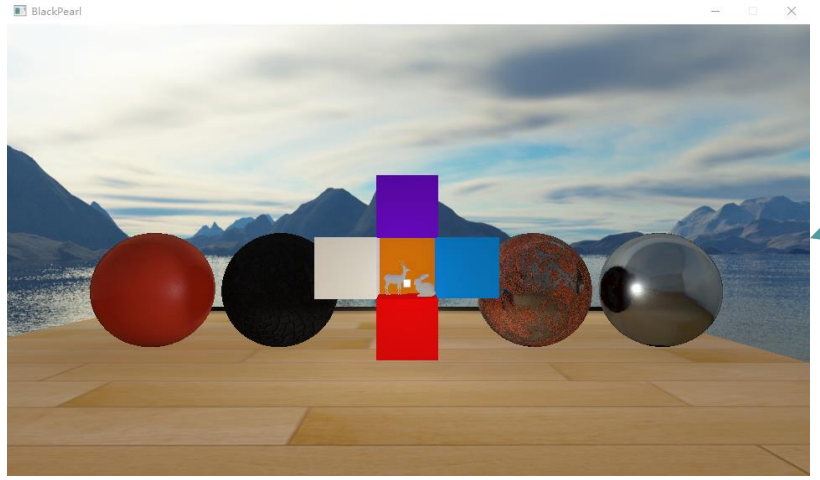
Dynamic global illumination scene	Image based lighting GI	Light probe based GI	voxel cone tracing GI
Dynamic indirect light	×	√ but slow	√
Dynamic direct light	√	√	√
Moving object	√	√	√
A sudden change of scene	×	×	√
Dynamic shadow	shadow map	shadow map	√

voxel cone tracing GI



GI Algorithm Comparison--Rendering quality

Rendering quality depends light probe distribution

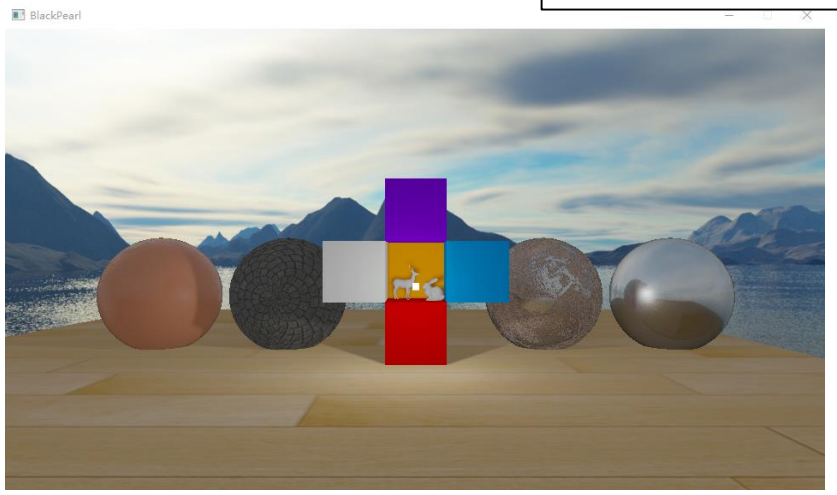


a) voxel cone tracing GI

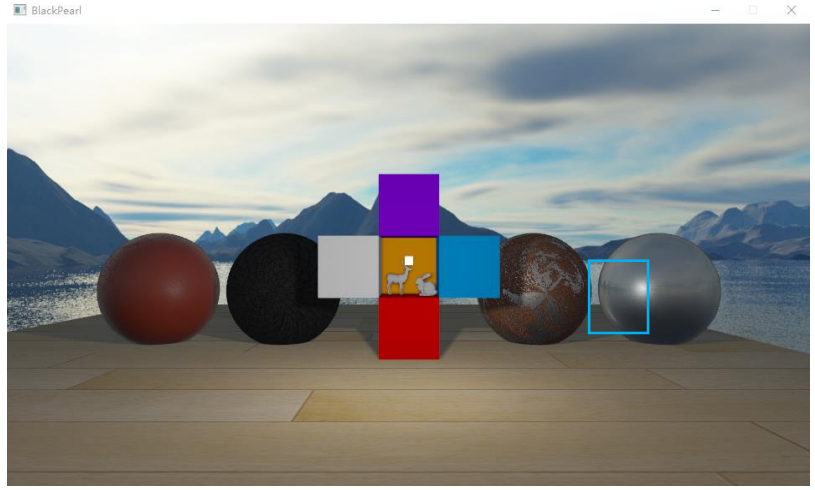
more realistic , restores more details of the illumination

Rendering quality is related to voxel resolution

There may be a light leak → needs thick walls

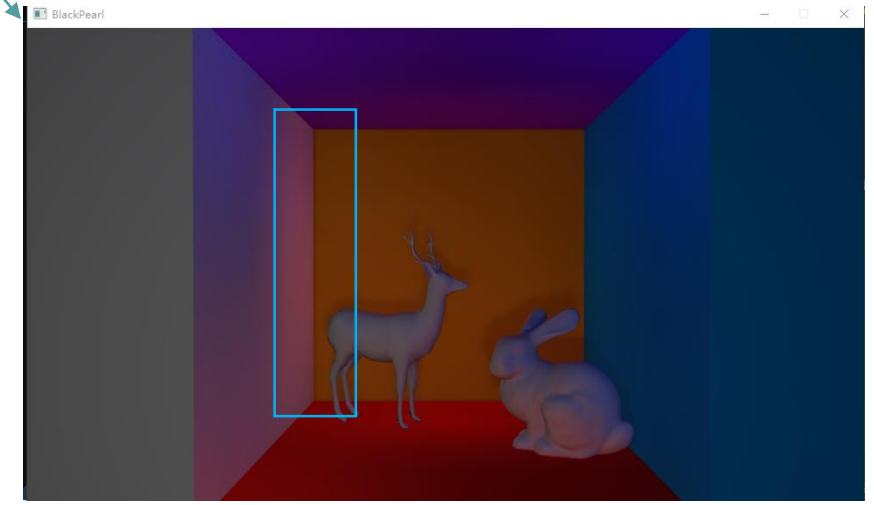


b) light probe based GI



c) image based lighting GI

Lack of rendering detail



Light leak

GI Algorithm Comparison--Application scenarios

256^3->73MB
512^3->512MB

The higher the resolution, the more accurate the scene rendering→Large amount of video memory

voxel cone tracing GI

Same resolution 3D texture, Small scene has more voxels to represent a model

↓

Smaller scene rendering

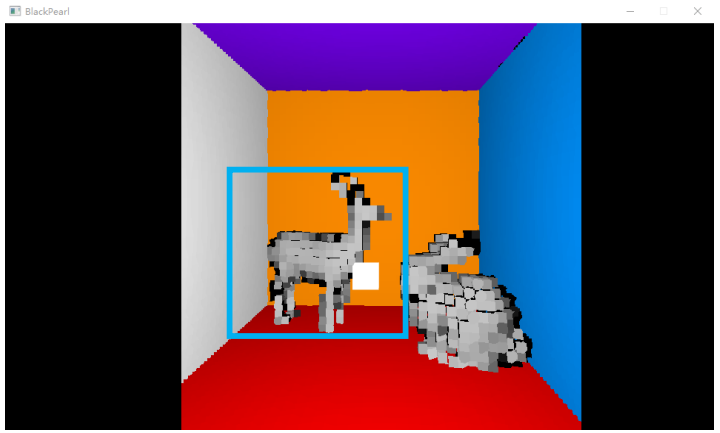


image base lighting GI

Good for Real-time rendering, but Lack of rendering detail, It is suitable for simulating the scene where the ambient light of all objects is generally consistent, such as the sky and sunlight.

light probe based GI

It needs very little memory and is suitable for large scene rendering with a lot of diffuse reflection

05

Chapter

Conclusion

Innovation and Conclusion

1. **Designed a Rendering Engine** for Dynamic GI

2. **Designed rendering pipeline of** three kinds of dynamic GI algorithms:
   Image-based lighting GI, voxel cone tracing GI, lightprobe-based GI

3. **Optimization** of rendering efficiency and quality for voxel cone tracing GI and light probe-based GI

Optimization	voxel cone tracing GI	light probe-based GI
DrawCall number	Limit render range, camera-centered voxelization	Deferred rendering pipeline
Memory		Spherical harmonic function encodes diffuse illumination
Rendering Quality	1.Gaussian Blurred for 3D mipmap texture 2.Avoid diffuse light flickering by taking multiple voxels ahead of time	Add reflection probes
Real-time performance	1.Voxelize scene only when obeject moves or light changes	1.Cache nearest probes and Area Manager 2.Update Shadowmap and reflection probe only when obeject moves

Innovation and Conclusion

4. **Comparison** of three Dynamic GI algorithms,
   including storage data structure, usage scene, real-time performance
   and rendering quality.

06

Chapter

Future work

Improve real time performance

Light probe-based GI -→Calculate Spherical Harmonic Coefficient in GPU

Batch rendering

Reduce storage space

Voxel cone tracing GI -→try clipmap

References

[1] GDC Presentation 2019, Ray-traced Irradiance Fields, McGuire

[2] NVIDIA's GTC 2013,Voxel Cone Tracing and Sparse Voxel Octree for Real-time Global Illumination,Cyril Crassin

[3] GDC Presentation 2014,In-Game and Cinematic Lighting of The Last of Us, Ding

[4] GDC Presentation 2018,Dynamic Global Illumination

[5] Real-time Global Illumination using Irradiance Probes, Simon Sedlacek

[6] WIKI:https://en.wikipedia.org/wiki/File:Ray_Tracing_Illustration_First_Bounce.png

[7] Unity Documentation. https://docs.unity3d.com/Manual/Lightmapping.html

[8] Unreal Engine 4 Documentation. Indirect Lighting Cache ://docs.unrealengine.com/en-US/Engine/Rendering/LightingAndShadows/IndirectLightingCache/index.html

[9] James T K. The Rendering Equation. ACM Siggraph Computer Graphics, 20(4):143–150, 1986.

[10] Karis B, Epic Games.Real Shading in Unreal Engine 4 SIGGRAPH,2013.

https://cdn2.unrealengine.com/Resources/files/2013SiggraphPresentationsNotes-26915738.pdf

[11] Cyril Crassin and Simon Green. Octree-based sparse voxelization using the gpu hardware rasterizer. OpenGL Insights, 2012.

Reference

[12] Ramamoorthi R, Hanrahan P.An Efficient Representation for Irradiance EnvironmentMaps.SIGGRAPH01,497–500,2001.http://graphics.stanford.edu/papers/envmap/

[13] Cabral B, Max N, Springmeyer R.Bidirectional Reflection Functions from Surface Bump Maps. In SIGGRAPH 87, pages 273–281,1987.

[14] MacRobert T M. Spherical Harmonics: An Elementary Treatise on Harmonic Functions, with Applications. Dover Publications,1948.

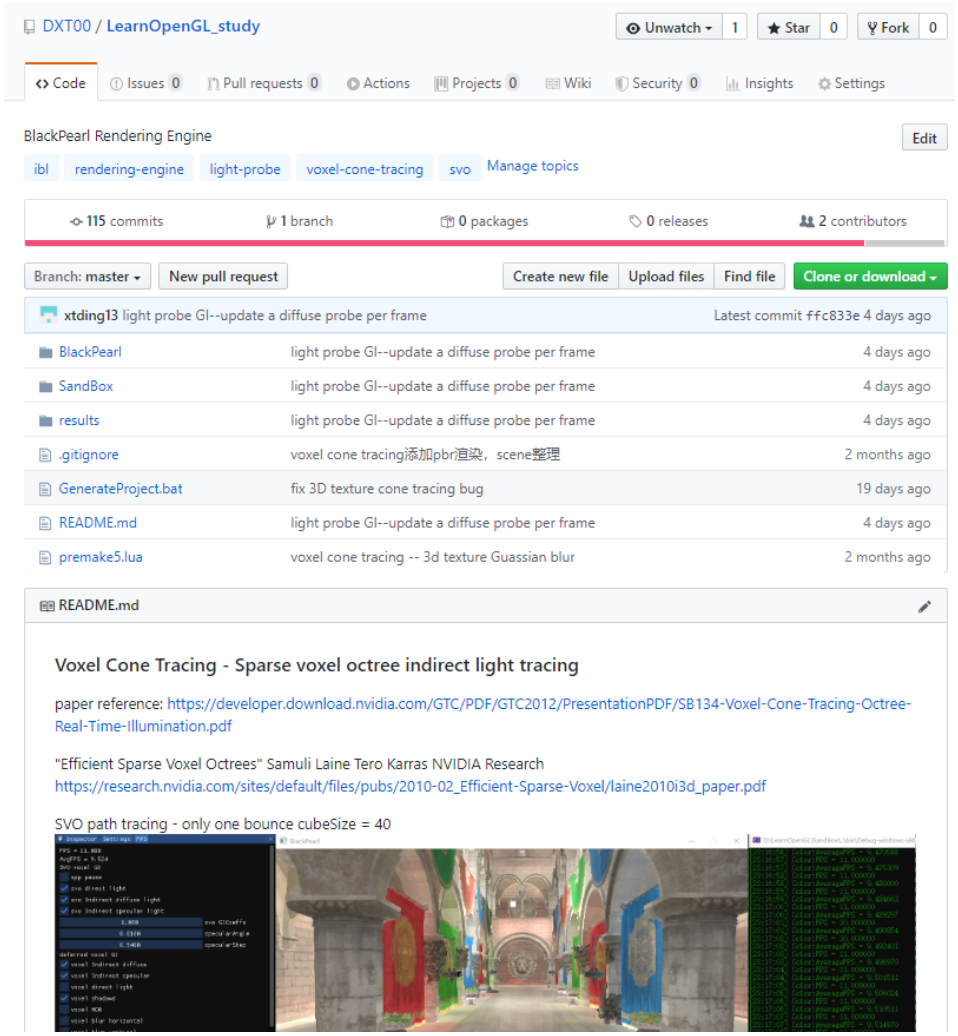
[15] Sillion F X, Arvo J, Westin S H, et al. A Global Illumination Solution forGeneral Reflectance Distributions. In SIGGRAPH 91, pages 187–196.

[16] Kalle Bladin ,“Real Time Global Illumination Using Voxel Cone Tracing”, TSBK03

Techniques for Advanced Computer Games,2015
http://kbladin.se/dmt_projects/docs/TSBK03_report_kalle_bladin.pdf

Project Code

GitHub:
https://github.com/DXT00/LearnOpenGL_study



Thanks