Project report for the CG 100433 course

Project Title

Auto-moving Chess Game

Team member

1952114 荆宇泉
1950084 陈泓仰
1952112 赵颂霖
1952106 赵艺博
1952547 孔艺菲
1853047 孔庆晨

Abstract

As we know, chess has complex forms and delicate textures, which can well reflect detailed light effect. Because the emphasis is openGL, not games. We decided to work on Auto-moving Chess Game project. We used many techniques including Jason to analysis gltf model, Cubemaps technique to make skybox, bloom technique, PBR shader with IBL, Gamma Correction and HDR technique. And we implemented an integrated and logical frame to control each chess piece, realized moving, rocking and beating other chess pieces with well-organized camera controlling and moving.

Motivation

We like to play chess. Chess represents wisdom, friendship and Steady. And chess has complex forms and delicate textures, which can well reflect detailed light effect. Although the rule of chess is complex, simplified auto-moving chess game is easier to implement. In order to have enough time to learn more things about OpenGL, we choose Auto-moving Chess Game as our project.

The Goal of the project

The original goal of the project is to make dynamic wallpaper with beautiful chess scenes. We aimed at using a wide variety of techniques to render the scenes more vivid. Chess has to be moved according to rules and can eat other chess piece. Because of the limited time, we have not deploy it to the dynamic wallpaper.

The Scope of the project

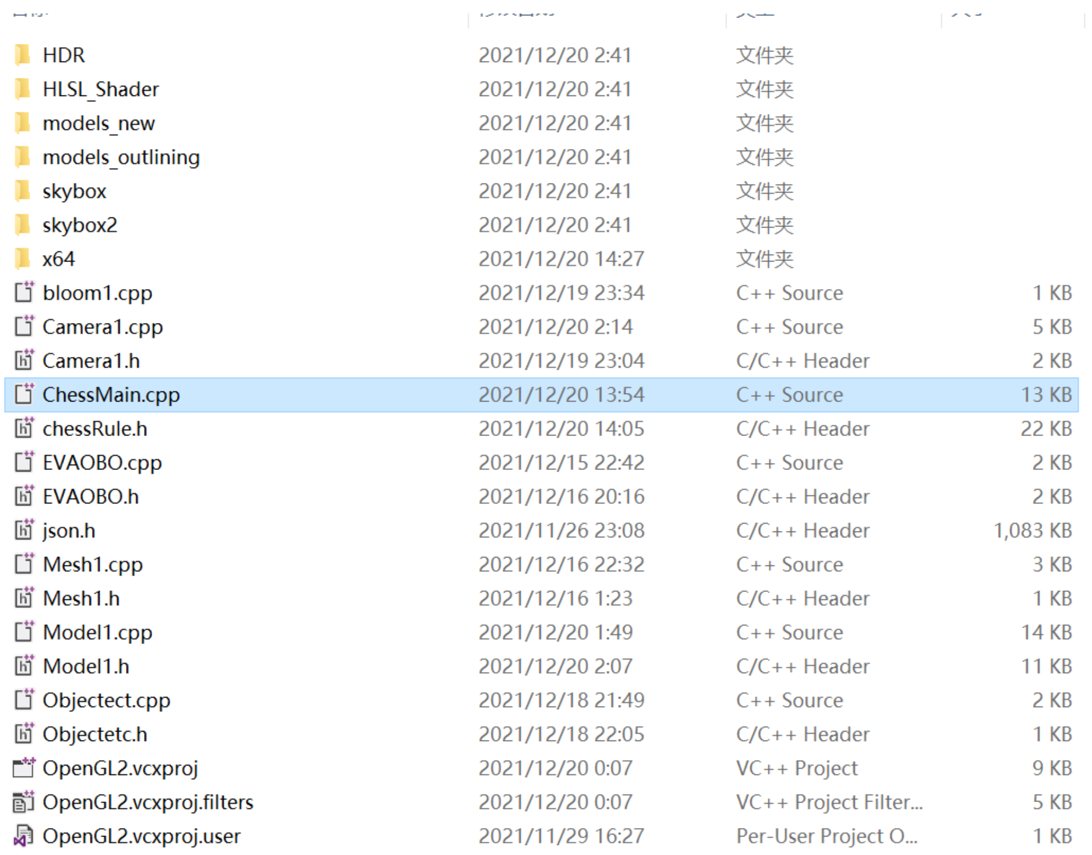
- Use Ray-Tracing technique to make the scenes more vivid.
- Add the chess AI to play chess with players automatically.

- Add friendly camera control. All control can be done by one mouse/touchpad/touchscreen.
- Add smooth animations to the chess when interacting and floating effect when it's still.

Involved CG techniques

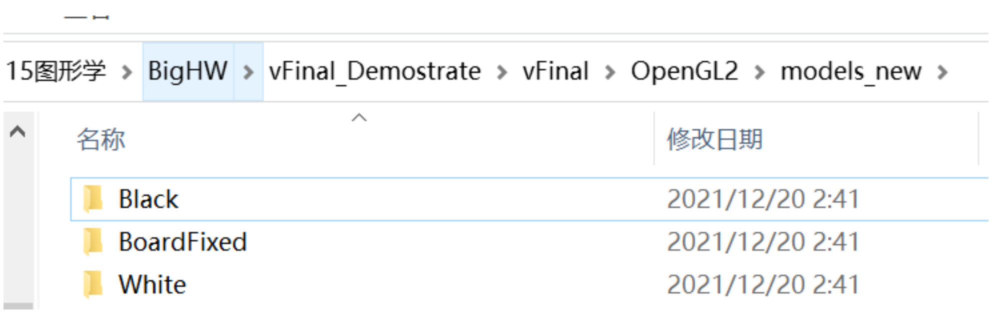
- Skybox —— remote background that won't translate with the camera.
- HDR —— Show details even if the scene is too bright or too dark
- PBR —— Using physical based render principle to render liver scenes.
- IBL —— collect lights from images to make chess pieces reflect the light of Skybox.
- Bloom —— using Gaussian blur to make object have blurred halo and look like emitting lights.

Project contents

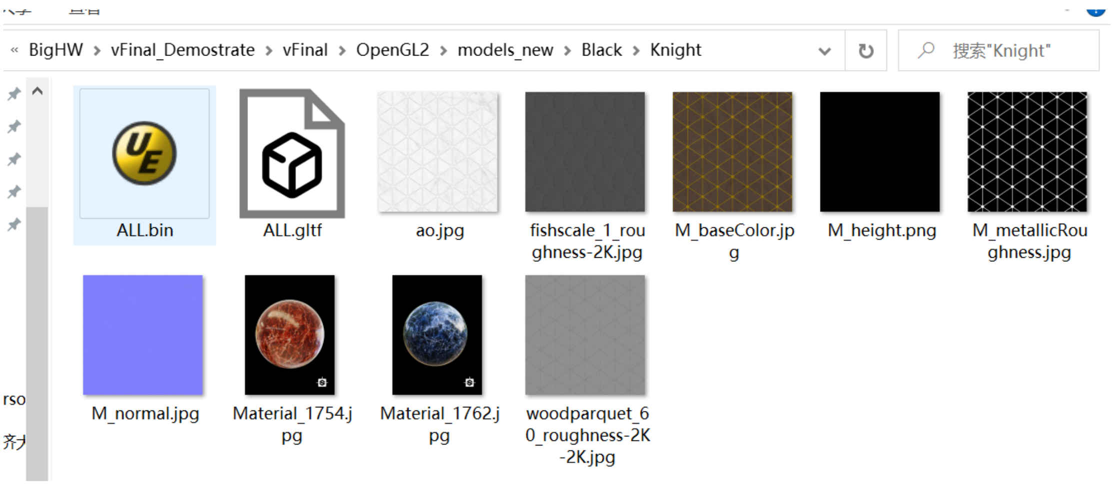


HDR	2021/12/20 2:41	文件夹	
HLSL_Shader	2021/12/20 2:41	文件夹	
models_new	2021/12/20 2:41	文件夹	
models_outlining	2021/12/20 2:41	文件夹	
skybox	2021/12/20 2:41	文件夹	
skybox2	2021/12/20 2:41	文件夹	
x64	2021/12/20 14:27	文件夹	
bloom1.cpp	2021/12/19 23:34	C++ Source	1 KB
Camera1.cpp	2021/12/20 2:14	C++ Source	5 KB
Camera1.h	2021/12/19 23:04	C/C++ Header	2 KB
ChessMain.cpp	2021/12/20 13:54	C++ Source	13 KB
chessRule.h	2021/12/20 14:05	C/C++ Header	22 KB
EVAOBO.cpp	2021/12/15 22:42	C++ Source	2 KB
EVAOBO.h	2021/12/16 20:16	C/C++ Header	2 KB
json.h	2021/11/26 23:08	C/C++ Header	1,083 KB
Mesh1.cpp	2021/12/16 22:32	C++ Source	3 KB
Mesh1.h	2021/12/16 1:23	C/C++ Header	1 KB
Model1.cpp	2021/12/20 1:49	C++ Source	14 KB
Model1.h	2021/12/20 2:07	C/C++ Header	11 KB
Objectect.cpp	2021/12/18 21:49	C++ Source	2 KB
Objectetc.h	2021/12/18 22:05	C/C++ Header	1 KB
OpenGL2.vcxproj	2021/12/20 0:07	VC++ Project	9 KB
OpenGL2.vcxproj.filters	2021/12/20 0:07	VC++ Project Filter...	5 KB
OpenGL2.vcxproj.user	2021/11/29 16:27	Per-User Project O...	1 KB

- HDR floder: including HDR file that using in IBL
- HLSL_Shader floder: including all vertices shader and fragment shader
- Skybox2 floder: including the skybox textures.
- Models_new floder: including the board and chess model in form of gltf


15图形学 > BigHW > vFinal_Demostrate > vFinal > OpenGL2 > models_new >

名称	修改日期
Black	2021/12/20 2:41
BoardFixed	2021/12/20 2:41
White	2021/12/20 2:41


« BigHW > vFinal_Demostrate > vFinal > OpenGL2 > models_new > Black > Knight

In each model, we have 9 textures.
List and explain the contents of your projects.

Implementation

- Model loading: it's surprised that gltf format is not unified. When we changed models, we have to change the program to adapt to each gltf model.
- Code restructure: First, we use YouTube's mesh and model class, but soon we found their functions are limited. We add texture items to adapt PBR's need.
- Texture loading: We used a parameter to determine the texture item, which let us load all kinds of texture in one function.
- FBO: FBO is not a set of data, but a container. When we use it, we must attach more than one set of data on it.
- Chess move algorithm: Specialized for better display effect. Step with chess or a larger movement range are prior to take.
- Animations: Add smooth animations to the chess when interacting and floating effect when it's still.
- Camera control: All control can be done by one mouse/touchpad/touchscreen.
- Debug: Because OpenGL program is hard to debug, we use professional tool to debug the program.

Details for animations:

Control logic

Control by frame is not stable enough due to the variation of the frame rate, so we choose to control by time: record the time when the animation starts, and calculate which frame the animation should render according to [current time-start time].

x = float(info.currentFrame) / float(max_frame[info.moveType]);（∈[0, 1]）

Packaging logic：

最外层：传入模型、shader、动画与时间。

计算层：根据动画种类和运动方程计算棋子（相对棋盘的）坐标。

渲染层：根据棋盘坐标计算空间坐标并渲染。

Animation: Float effect

给静止的棋子浮动效果。

棋子位置：0.5~1~0.5~0~0.5

使用 sin 函数使运动更加平滑：

height = 0.5 * (1 + sin(x * 2 * PI));

Animation: Move

x=[0,0.25)　　升起

height=0.5+2.5*sin(PI * 2 * x) [0.5~3]

x=[0.25,0.75)　移动

height=3

X=oxpos+(xpos-oxpos)*[(x-0.25)*2]

Y=oypos+(ypos-oypos)*[(x-0.25)*2]

（X 和 Y 也使用 sin 来使运动平滑）

x=[0.75,1]　　降落

height=3-2.5*sin(PI * 2 * (x - 0.75)) [3~0.5]

Animation: Disappear

动画的前一半：闪烁

color_r = (rand() % 3) / 2;

动画的后一半：从白色开始渐变为黑色

color_r = (1 - x) * 2;

针对黑色棋子的颜色修正：

lightColor_1 = glm::vec4(color_r / 2, color_r / 3, color_r / 4, 1.0f);

白色棋子消失时闪烁蓝白色，黑色棋子消失时显示深红色。

Details for camera control:

Location and orientation

Inspiration obtained when using Blender to observe the model.

Take the chessboard (0,0,0) as the center, fix a radius to make a sphere, and let the camera move on the surface of the sphere. Meanwhile, make the camera always point to the

center of the chessboard.
Pros:

Only need to calculate the camera position, all other parameters can be derived through matrix transformation.

相机朝向 = 位置取反：Orientation = glm::normalize(-Position);

右向量 = Position × (0,1,0)

No misoperation: the camera won’t point to or go to a wrong place, which is harder to reset.

Convenient to limit the radius range and the rotation range, thus avoid mold wear.

Simplified operation logic: a mouse, or touch pad, or touch screen can complete all operations.

Zoom

Use the scroll wheel to zoom.

Zooming is achieved by simultaneously ×0.95 or ÷0.95 to the camera's three-dimensional coordinates.

Calculate the radius from the coordinates. If the radius exceeds the limit range after zooming, cancel it.

Auto move

After every 3 moves, the lens will rotate 180° to the opposite angle of view.

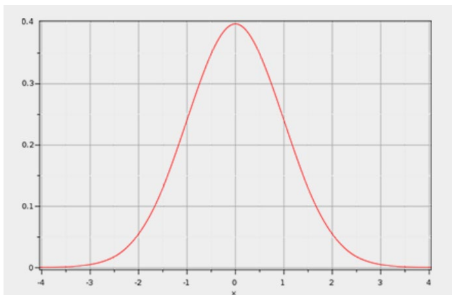
Details for bloom effect:

Bloom is an effect used to reproduce and imaging artifact of real-world cameras. It is an aura-like color around bright lights that gives them the appearance of being brighter than they really are in reality. The effect produces feathers of light, which extends from the edge of lighting area in a picture, leading to the illusion of an extremely bright light, which overwhelming the camera or eye capturing the scene.

In practice, we implement this effect by generally 3 steps:

Firstly, we render our model as we did it before, except we also render all the bright spots of our model in another texture.

Secondly, we implement the blur in post-processing. Now that these two are separated, we can blur the second one. The algorithm we use to blur the model and the steps we do that will decide the quality of our bloom effect. In our project, we use a Gaussian blur. Gaussian blur is based on the Gaussian curve which is commonly described as a bell-shaped curve giving high values close to its center that gradually wear off over distance. The Gaussian curve can be mathematically represented in different forms, but generally has the following shape:

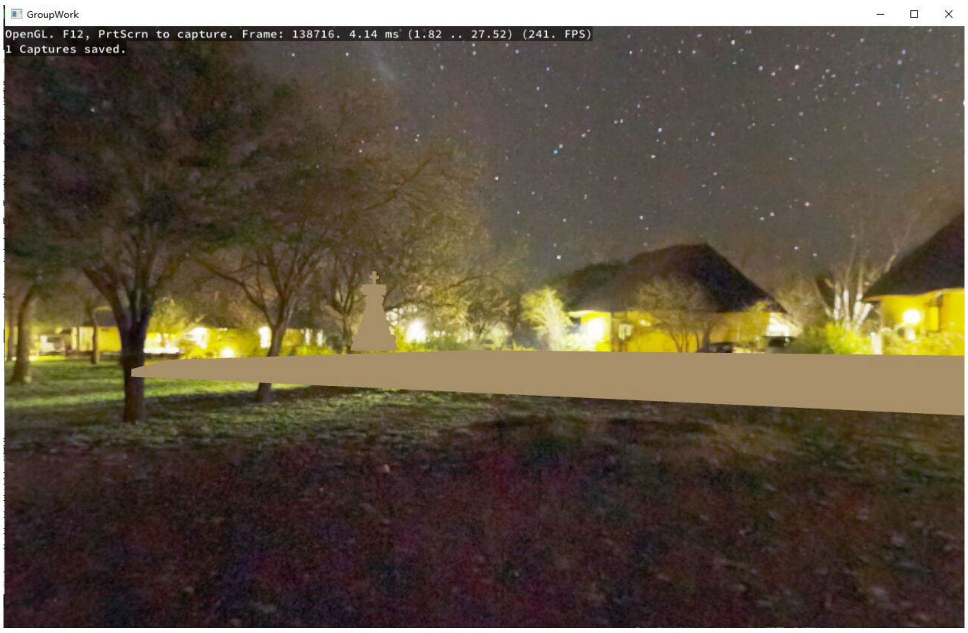


We blur our image for two times using the framebuffer objects. After the horizontal and vertical pass implemented in shader on image respectively, we hold two shading together. Lastly, we bind the two models we've rendered. And we adjusted the parameters of Gaussian blur in the second step based on the final effect of the binding.

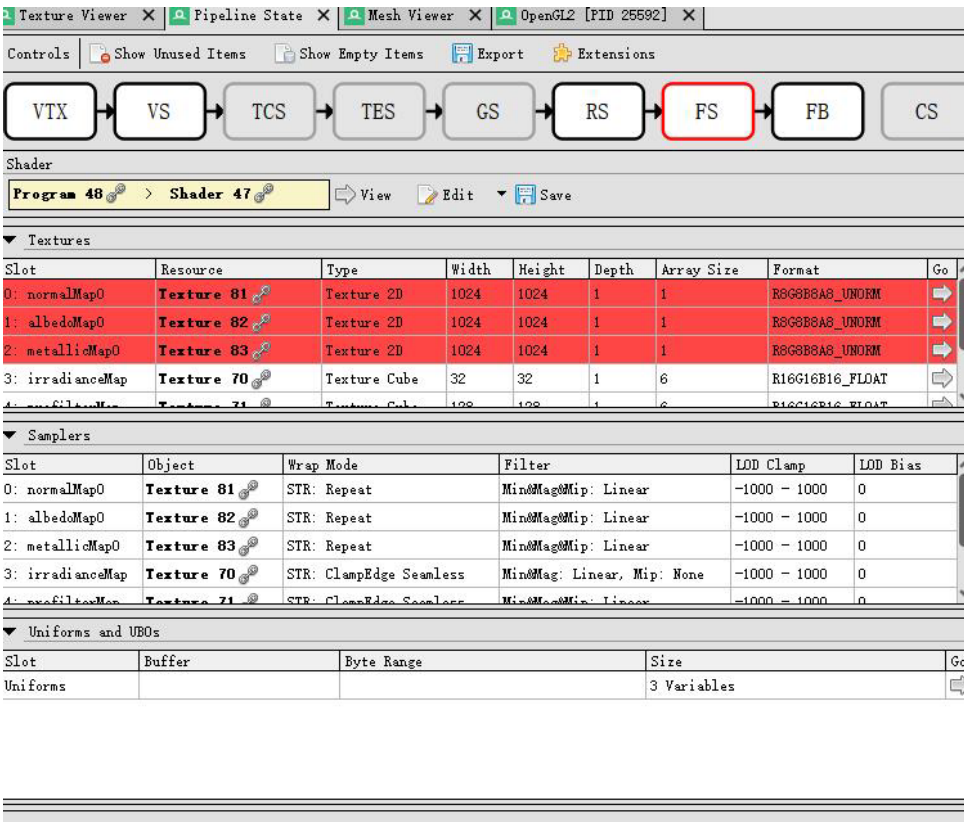
Details for debugging：

Sometimes it is difficult to find problems in openGL using common debugging methods. The method of setting breakpoints is used to view the main logic errors, but if there is a problem in the GLSL syntax, it is actually more difficult to find it out.

When writing the fragment shader of PBR, there was a problem that the texture could not be displayed normally. We used the third-party software RenderDoc to debug and found that there was a problem in the texture part of the rendering pipeline, which gave us great inspiration.



```
OpenGL. F12, PrtScrn to capture. Frame: 138716. 4.14 ms (1.82 .. 27.52) (241. FPS)
1 Captures saved.
```



Shader

Program 48 > Shader 47 View Edit Save

Textures

Slot	Resource	Type	Width	Height	Depth	Array Size	Format	Go
0: normalMap0	**Texture 81**	Texture 2D	1024	1024	1	1	R8G8B8A8_UNORM	➡
1: albedoMap0	**Texture 82**	Texture 2D	1024	1024	1	1	R8G8B8A8_UNORM	➡
2: metallicMap0	**Texture 83**	Texture 2D	1024	1024	1	1	R8G8B8A8_UNORM	➡
3: irradianceMap	**Texture 70**	Texture Cube	32	32	1	6	R16G16B16_FLOAT	➡
4: prefilterMap	**Texture 71**	Texture Cube	128	128	1	6	R16G16B16_FLOAT	➡

Samplers

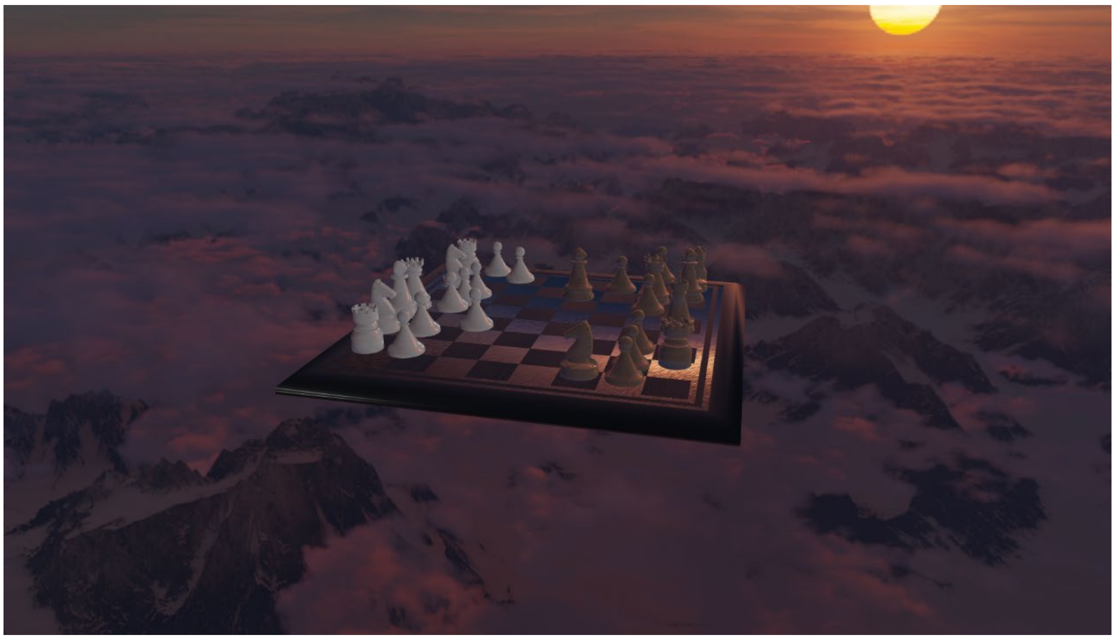
Slot	Object	Wrap Mode	Filter	LOD Clamp	LOD Bias
0: normalMap0	**Texture 81**	STR: Repeat	Min&Mag&Mip: Linear	-1000 - 1000	0
1: albedoMap0	**Texture 82**	STR: Repeat	Min&Mag&Mip: Linear	-1000 - 1000	0
2: metallicMap0	**Texture 83**	STR: Repeat	Min&Mag&Mip: Linear	-1000 - 1000	0
3: irradianceMap	**Texture 70**	STR: ClampEdge Seamless	Min&Mag: Linear, Mip: None	-1000 - 1000	0
4: prefilterMap	**Texture 71**	STR: ClampEdge Seamless	Min&Mag&Mip: Linear	-1000 - 1000	0

Uniforms and UBOs

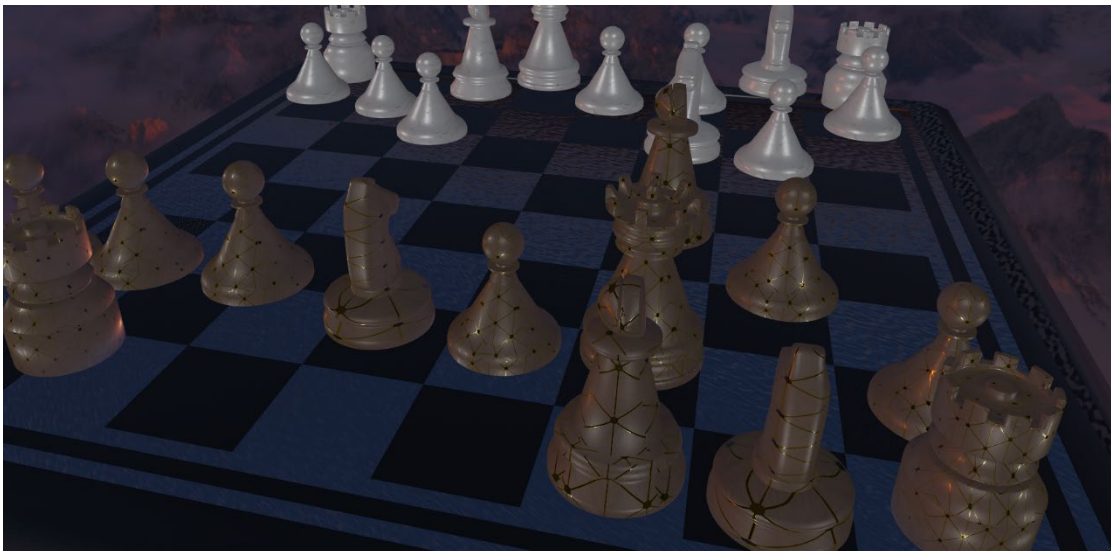
Slot	Buffer	Byte Range	Size	Go
Uniforms			3 Variables	➡

Results

This project is open source：[watermellye/chessCG: 计算机图形学 小组项目 (github.com)]
Demo video：[OpenGL 实现的国际象棋壁纸_哔哩哔哩_bilibili]
Please see “how to run” in repo.



棋子细节



吃子特效

Roles in group

- All of us contributed for model choosing and making, especially 赵艺博 contributed hardest work. What's more, she used Jason to analysis gltf model and load the model into the project.
- 荆宇泉 choose and wrote the skybox. And he with 孔庆晨,赵艺博 were responsible and HDR.
- 陈泓仰 made the frame of the project, realized moving, rocking and beating other chess pieces with well-organized camera controlling and moving.
- 赵颂霖 generated special effects of beating other chess pieces. First, he tried particle system. But the effect is not good enough, he bring about bloom effect.
- 孔艺菲 and 孔庆晨 brought out pbr technique and added Gamma Correction and IBL, using the results of HDR to achieve better effect.
- What's more, 孔艺菲 tried to add shadow, but the results looked odd as the chess pieces are rocking. So the shadow is not added in the final project.
- Last,孔庆晨,赵艺博,陈泓仰 made great effort in merging codes.

References

- LearnOpenGL CN
- OpenGL tutorial YouTube