Cross Browser Tracking Summary Part-2

一叶飘零 / 2018-12-29 10:54:00 / 浏览数 2599

前言

本篇文章主要讲解模型和参数选择部分知识，下一篇文章将讲述相关代码实现。

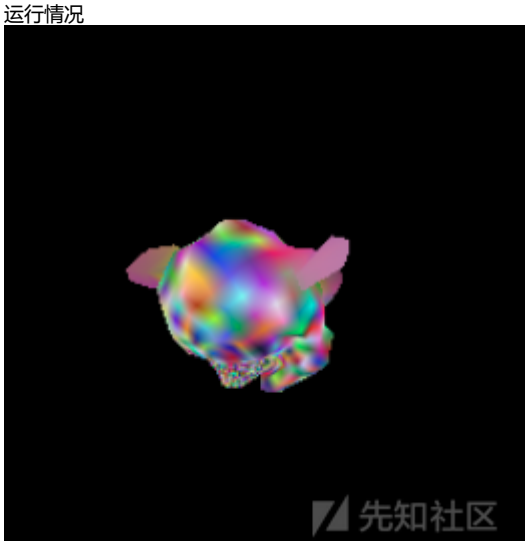
Task(a)-纹理

简单来说即测试片段着色器中的常规纹理特征。
具体来说，即将经典的Suzanne模型在随机生成纹理的画布上呈现。纹理大小为256×256的正方形，通过随机选择每个像素的颜色来创建，即我们在一个像素的三个基色（

之所以选择这个随机生成的纹理，是因为这个纹理比常规纹理具有更多的指纹特征。原因如下，当片段着色器将纹理映射到模型时，片段着色器需要在纹理中插入点，以便
相关模块为：

```
this.testList.push(new TextureTest(this.susanVertices, this.susanIndices, this.susanTexCoords, this.texture));
```

运行情况




```
toServer.js:188
Uint8Array(262144) [0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255,
  0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0
▼, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0,
  255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255,
  0, 0, 0, 255, …]
```

(后续文章将解释这个262144的数组)

Task(b)-渐变

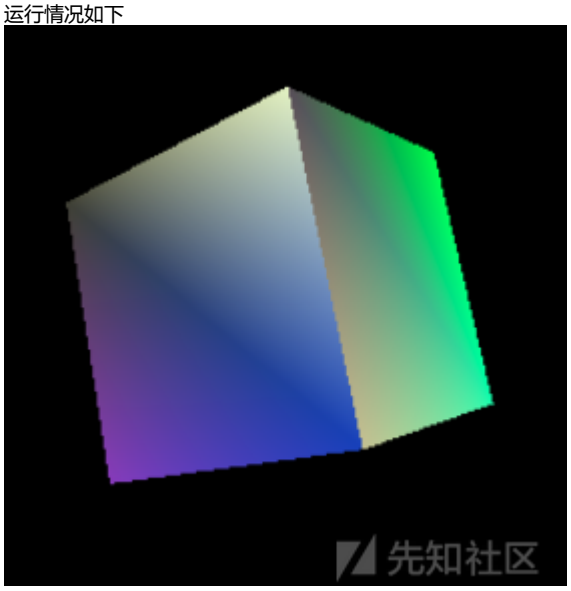
简单来说即测试片段着色器在画布上的渐变特征。
具体来说，不同的渐变颜色被画在一个正方体模型的六个面上,每个面上的4个顶点都作为颜色的出发点，例如下图

(b) Varyings

并且尽量选择这些渐变颜色来扩大每个面上的色彩之间的差异，例如某个面的一个顶点上蓝色比例非常大(0.9/1)，那么另一个顶点的蓝色比例就必须非常少(0.1/1)，相应的
相关模块为

```
this.testList.push(new CubeTest('normal'));
```

运行情况如下



toServer.js:188

```
Uint8Array(262144) [0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0,
  0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0,
▶ 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0,
  0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0,
  255, …]
```

Task(b')-抗锯齿与渐变

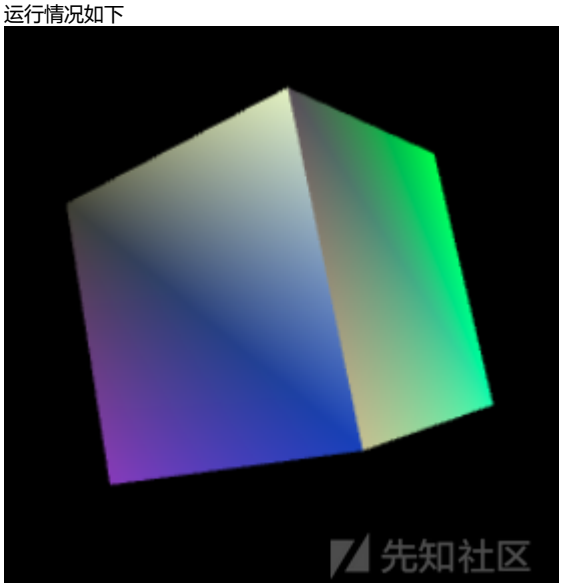
简单来说即测试抗锯齿特征，换句话来说，即测试浏览器如何让模型边缘更加光滑
这里使用与Task(B)相同的方式，并且加入抗锯齿。如果进一步测试，将会发现模型的每个边缘都被柔化

(b') Anti-aliasing

相关模块如下

```
this.testList.push(new CubeTest('aa'));
```

运行情况如下



toServer.js:188

```
Uint8Array(262144) [0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0,
  0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0,
▶ 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0,
  0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0,
  255, …]
```

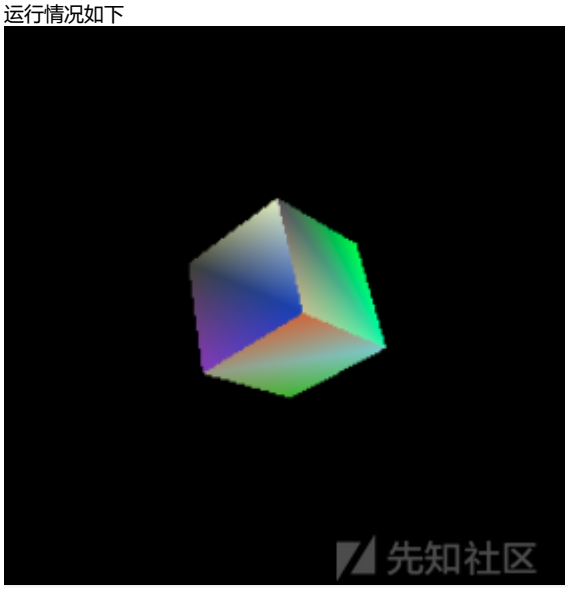
和前一张对面，边缘的确有显著的柔化

Task(c)-camera

简单来说即测试投影特性，换句话说，即一个输入片段着色器的投影矩阵
所有的配置信息都与任务一致，除了camera的被移动到了新的位置`[-1,-4,-10]`，得到的立方体比Task(b)更小，因为camera被移动的离立方体更远
相关模块如下

```
this.testList.push(new CameraTest());
```

运行情况如下



toServer.js:188

```
Uint8Array(262144) [0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0,
  0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0,
▶ 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0,
  0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0,
  255, …]
```

Task(d)-直线与曲线

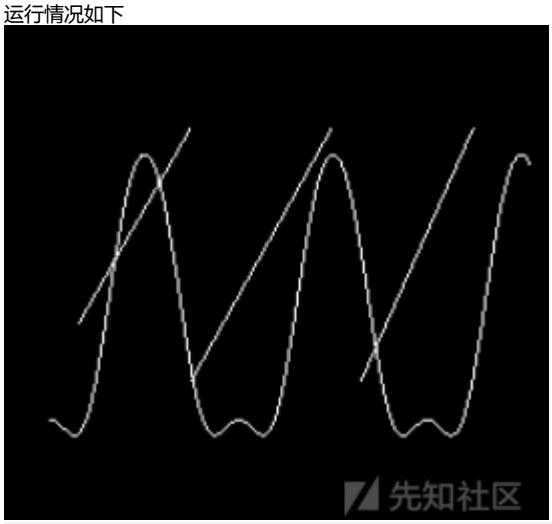
简单来说即测试直线和曲线。 在画布上绘制一条曲线和三条不同角度的直线。
具体来说，曲线遵循以下公式：

```
y = 256-100cos(2.0πx/ 100.0)+ 30cos(4.0πx/ 100.0)+ 6cos(6.0πx/ 100.0)
```

其中[0,0]为画布的左上角，x轴向右增加，y轴增加到底部。
三条直线的起点和终点是
`{[38.4,115.2]■[89.6,204.8]}`，`{[89.6,89.6]■[153.6,204.8]}`和`{[166.4,89.6]■[217.6,204.8]}`。
选择这些特定的线条和曲线，以便测试不同的渐变和形状。
相关模块如下

```
this.testList.push(new LineTest('normal'));
```

运行情况如下



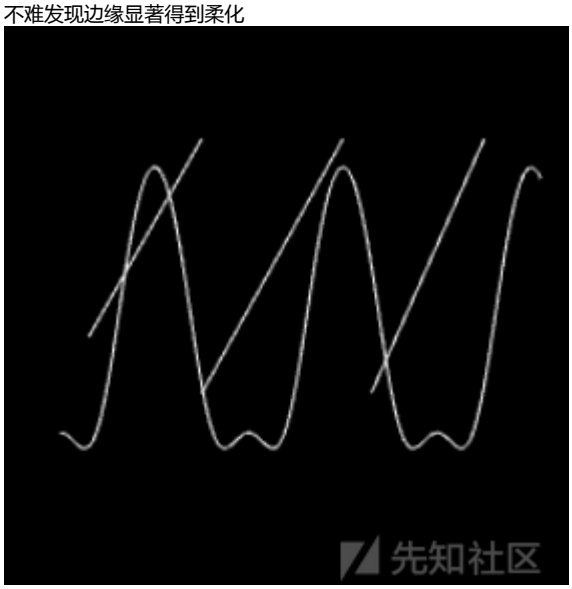

```
toServer.js:188
Uint8Array(262144) [0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0,
  0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0,
▶ 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0,
  0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0,
  255, …]
```

Task(d')-抗锯齿与直线与曲线

即在Task(d)中加入了抗锯齿
相关模块如下

```
this.testList.push(new LineTest('aa'));
```

不难发现边缘显著得到柔化



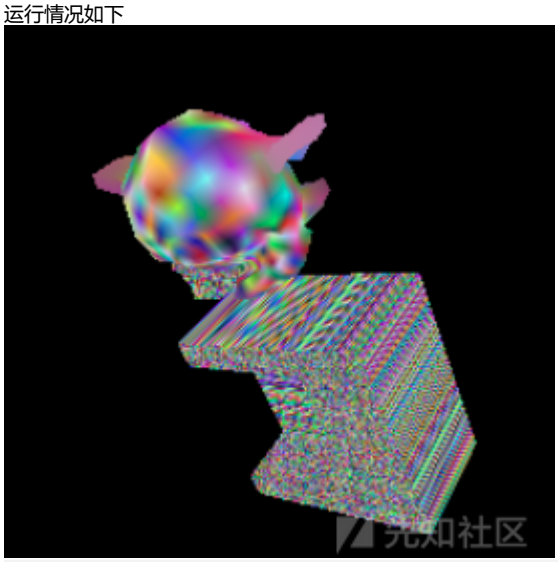

```
toServer.js:188
Uint8Array(262144) [0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0,
  0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0,
▶ 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0,
  0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0,
  255, …]
```

Task(e)-多重模型

简单来说即测试多个不同的模型在同一个画布上如何相互影响
除了Suzanne模型，这里引入了另一种模型：看起来像一个人倚靠在沙发上，称之为sofa模型，我们将两个模型平行放置，并同样对sofa模型用Task(a)中的随机生成的纹理
相关模块如下

```
this.testList.push(new TextureTest(this.combinedVertices, this.combinedIndices, this.combinedTexCoords, this.texture));
```

运行情况如下



toServer.js:188

```
Uint8Array(262144) [0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0,
  0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0,
▶ 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0,
  0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0,
  255, …]
```

Task(f)-光线

简单来说即测试漫射点光和Suzanne模型的相互作用。 漫射点光在照亮物体时会引起漫反射。
具体地说，该光是在RGB上具有相同值的白色，对于每种原色，光的功率为2，光源位于`[3.0,-4.0,-2.0]`。

在这个任务中选择一个白光源，因为纹理是各种颜色的，单色光可能会减少纹理上的一些微妙差异。
光线的强度需要精心设计。非常弱的光线不会照亮Suzanne模型，模型就会不可见；非常强的光会使一切变白，减少指纹特征。
在6台机器的小规模实验中，功率从0增加到255，我们发现当光功率为2时，这些机器之间的像素差异最大。并且光照位置可随机选择，不会影响特征指纹识别结果。
相关模块如下

```
this.testList.push(new SimpleLightTest(this.susanVertices, this.susanIndices, this.susanTexCoords, this.susanNormals, this.tex
```

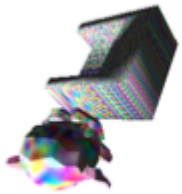
运行情况如下

toServer.js:188

```
Uint8Array(262144) [0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0,
  0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0,
▶ 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0,
  0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0,
  255, …]
```

Task(g)-光线与模型

简单来说即测试漫射点光和Suzanne模型与sofa模型的相互作用
因为当某个点的被光照时，一个模型可能会在另一个模型上产生阴影
这个任务中所有的光线配置与Task(f)一致，并且模型与Task(e)一致

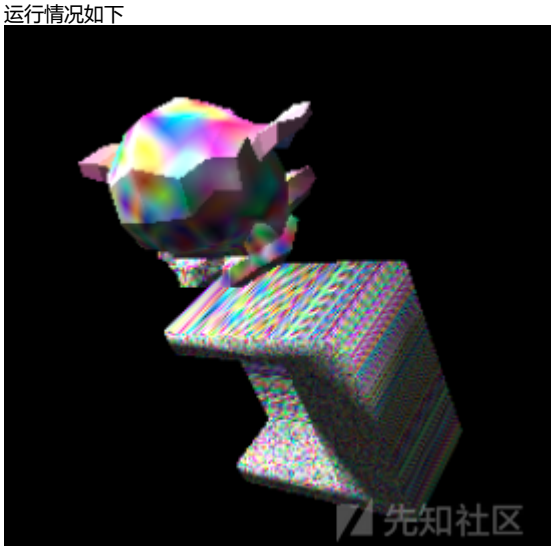


(g) Light&Models

相关模块如下

```
this.testList.push(new SimpleLightTest(this.combinedVertices, this.combinedIndices, this.combinedTexCoords, this.combinedNorma
```

运行情况如下



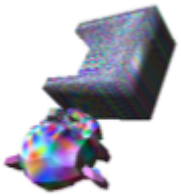
toServer.js:188

```
Uint8Array(262144) [0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0,
  0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0,
▶ 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0,
  0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0,
  255, …]
```

Task(h)-镜面光

简单来说即测试另一种色彩的漫射点光与镜面点光对2个模型的作用
与漫射点光相似，镜面点光将会导致照射对象的镜面反射，两束光源都位于`[0.8,-0.8,-0.8]`
漫射点光的RGB为`[0.75,0.75,1.0]`，镜面点光的RGB为`[0.8,0.8,0.8]`

这里有两件事值得注意。其一，我们选择了一个特殊的camera位置，其距离模型很近，并且有更大的影响，尤其是当sofa模型后部被镜面点光照射时。其二，虽然漫射点光

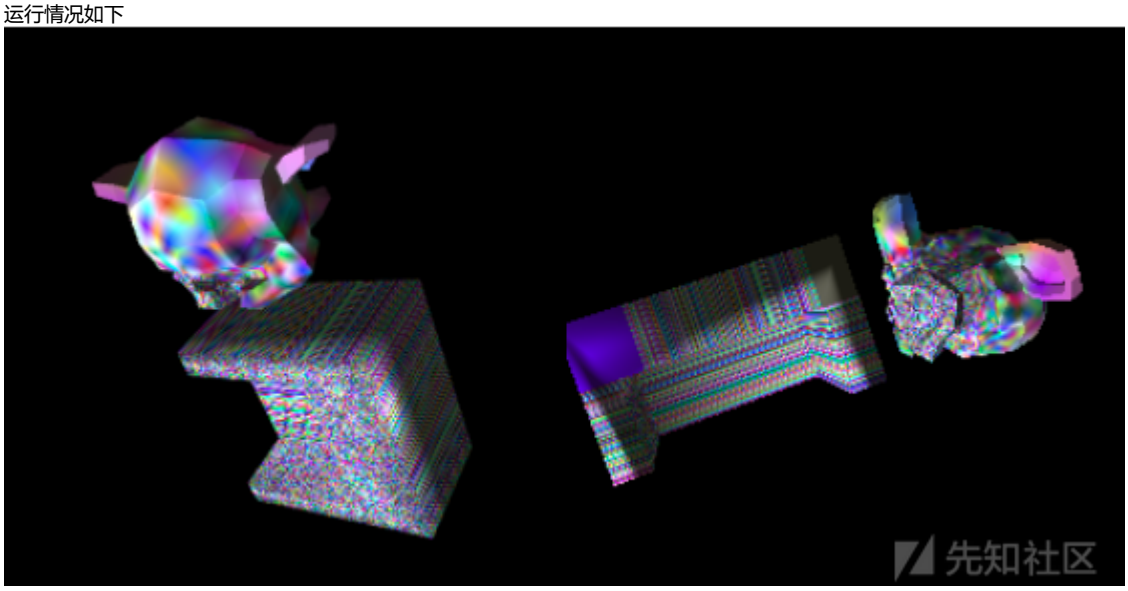


(h) Specular Light

相关代码如下，这里进行了多重测试，例如加入抗锯齿，再加入90度旋转，所以得到3组数据

```
this.testList.push(new MoreLightTest(this.combinedVertices, this.combinedIndices, this.combinedTexCoords, this.combinedNormals
```

运行情况如下




```
1                                                                    toServer.js:187
                                                                     toServer.js:188
  Uint8Array(262144) [0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0,
    0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0,
  ▶ 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0,
    0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0,
    255, …]
2                                                                    toServer.js:187
                                                                     toServer.js:188
  Uint8Array(262144) [0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0,
    0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0,
  ▶ 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0,
    0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0,
    255, …]
3                                                                    toServer.js:187
                                                                     toServer.js:188
  Uint8Array(262144) [0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0,
    0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0,
  ▶ 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0,
    0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0,
    255, …]
```

Task(i)-双纹理

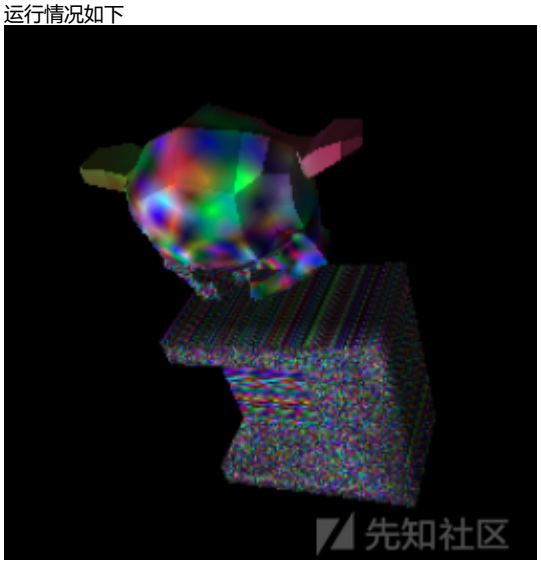
简单来说即测试对同一对象进行不同的纹理覆盖会有什么影响
我们生成另一种随机的纹理对Suzanne和sofa模型进行覆盖

(i) Two Textures

相关代码

```
this.testList.push(new TwoTexturesMoreLightTest(this.combinedVertices, this.combinedIndices, this.combinedTexCoords, this.comb
```

运行情况如下

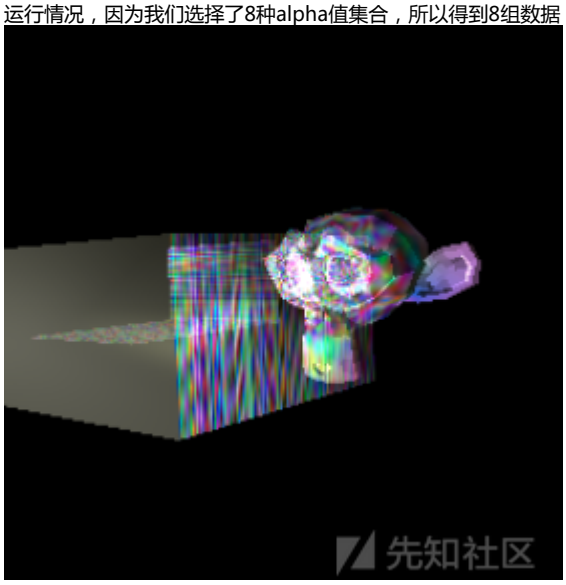


Task(j)-Alpha

该任务由8个子任务组成，即测试不同alpha值的影响
我们将Suzanne和sofa模型平行放置，并且从我们精心设置的alpha值集合{0.09,0.1,011,0.39,0.4,0.41,0.79,1}中选择alpha值，其中0表示完全透明，1表示完全
此时又有两点值得注意，其一，我们精心选择了3个变化不大的值，去反应不同alpha值带来的变化：{0.1,0.4,0.8}，并且值以0.01的速度变化，因为GUPs不能接受更小
相关代码如下

```
this.testList.push(new TransparentTest(this.combinedVertices, this.combinedIndices, this.combinedTexCoords, this.combinedNorma
```

运行情况，因为我们选择了8种alpha值集合，所以得到8组数据



toServer.js:194

```
Uint8Array(262144) [0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255,
  0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0
▶ , 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0,
255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255,
0, 0, 0, 255, …]
```

2 toServer.js:193

toServer.js:194

```
Uint8Array(262144) [0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255,
  0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0
▶ , 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0,
255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255,
0, 0, 0, 255, …]
```

3 toServer.js:193

toServer.js:194

```
Uint8Array(262144) [0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255,
  0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0
▶ , 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0,
255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255,
0, 0, 0, 255, …]
```

4 toServer.js:193

toServer.js:194

```
Uint8Array(262144) [0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255,
  0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0
▶ , 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0,
255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255,
0, 0, 0, 255, …]
```

5 toServer.js:193

toServer.js:194

```
Uint8Array(262144) [0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255,
  0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0
▶ , 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0,
255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255,
0, 0, 0, 255, …]
```

6 toServer.js:193

toServer.js:194

```
Uint8Array(262144) [0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255,
  0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0
▶ , 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0,
255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255,
0, 0, 0, 255, …]
```

7 toServer.js:193

toServer.js:194

```
Uint8Array(262144) [0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255,
  0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0
▶ , 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0,
255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255,
0, 0, 0, 255, …]
```

8 toServer.js:193

Task(k)-复合光

测试复合光特性，例如反射光，移动光，追踪光
特别的，我们生成了5000个金属环模型，并以随机的角度将他们放置在地面上，同时堆叠在一起。为了稳定性，我们选择了一个随机数种子用来生成随机数，这样测试即可
注意，这里我们使用单色光源，因为模型不是彩色的，这样做可以为我们提供更多的相互影响的细节
相关代码

```
this.testList.push(new LightingTest());
```

运行情况，在第二组里加入了抗锯齿，所以得到了两组数据


```
1                                                                    toServer.js:193
                                                                     toServer.js:194
  Uint8Array(262144) [1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0,
▶ 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1
  , 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1,
  1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, …]
THREE.WebGLRenderer 78                                               three.js:24761
2                                                                    toServer.js:193
                                                                     toServer.js:194
  Uint8Array(262144) [35, 27, 29, 255, 35, 35, 34, 255, 29, 35, 33, 255, 20, 30, 29, 255, 13,
  31, 26, 255, 22, 19, 24, 255, 24, 17, 26, 255, 24, 23, 24, 255, 27, 25, 20, 255, 31, 20, 23,
▶  255, 26, 26, 30, 255, 34, 28, 28, 255, 32, 30, 30, 255, 32, 31, 33, 255, 31, 31, 33, 255, 2
  4, 31, 31, 255, 28, 33, 33, 255, 23, 31, 32, 255, 21, 26, 29, 255, 21, 22, 27, 255, 21, 20,
  30, 255, 24, 18, 36, 255, 24, 22, 27, 255, 26, 20, 28, 255, 25, 20, 30, 255, …]
```

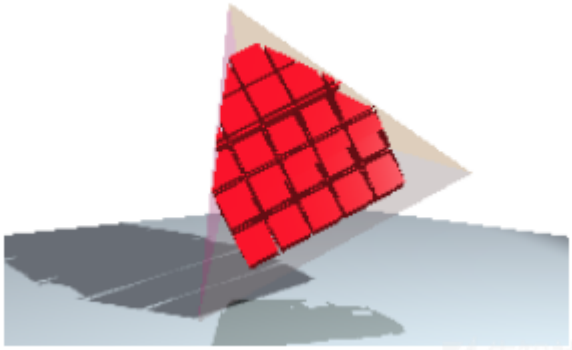
更多渲染任务

Task(l)-切面

相关模块为：

```
this.testList.push(new ClippingTest());
```

运行情况



toServer.js:188

```
Uint8Array(262144) [194, 206, 216, 255, 194, 206, 216, 255, 194, 207, 216, 255, 194, 207, 216, 255, 194, 207, 216, 255, 195, 207
, 216, 255, 195, 207, 216, 255, 195, 207, 217, 255, 195, 207, 217, 255, 195, 208, 217, 255, 195, 208, 217, 255, 195, 208, 217, 2
55, 196, 208, 217, 255, 196, 208, 217, 255, 196, 208, 218, 255, 196, 208, 218, 255, 196, 209, 218, 255, 196, 209, 218, 255, 196,
 209, 218, 255, 196, 209, 218, 255, 197, 209, 218, 255, 197, 209, 219, 255, 197, 209, 219, 255, 197, 210, 219, 255, 197, 210, 21
9, 255, …]
▶ [0 … 9999]
▶ [10000 … 19999]
▶ [20000 … 29999]
▶ [30000 … 39999]
▶ [40000 … 49999]
▶ [50000 … 59999]
▶ [60000 … 69999]
▶ [70000 … 79999]
▶ [80000 … 89999]
▶ [90000 … 99999]
▶ [100000 … 109999]
▶ [110000 … 119999]
▶ [120000 … 129999]
▶ [130000 … 139999]
▶ [140000 … 149999]
▶ [150000 … 159999]
▶ [160000 … 169999]
▶ [170000 … 179999]
▶ [180000 … 189999]
▶ [190000 … 199999]
▶ [200000 … 209999]
▶ [210000 … 219999]
▶ [220000 … 229999]
▶ [230000 … 239999]
▶ [240000 … 249999]
▶ [250000 … 259999]
▶ [260000 … 262143]
```


Task(m)-立方体纹理和菲涅尔效应

相关代码如下

```
this.testList.push(new BubbleTest());
```

运行情况

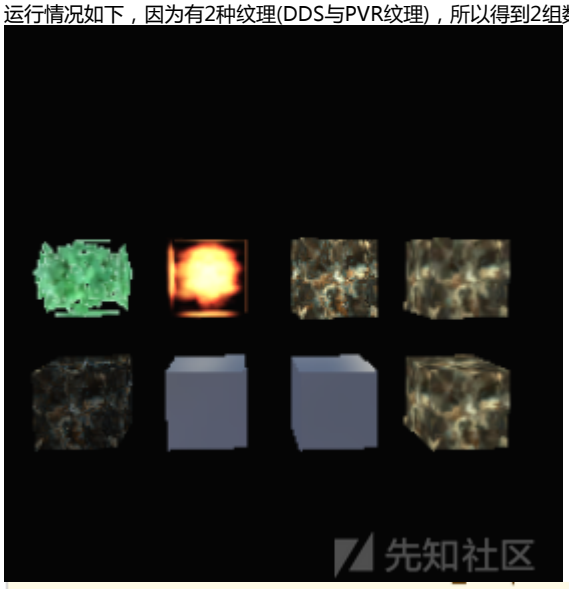

```
1                                                                    toServer.js:193
                                                                     toServer.js:194
Uint8Array(262144) [3, 15, 1, 255, 0, 12, 0, 255, 1, 12, 0, 255, 1, 11, 0, 255, 1, 10, 1, 25
5, 2, 13, 1, 255, 33, 44, 19, 255, 32, 41, 31, 255, 18, 29, 25, 255, 14, 34, 18, 255, 12, 27
▶ , 8, 255, 16, 25, 15, 255, 14, 24, 17, 255, 14, 28, 20, 255, 14, 26, 15, 255, 15, 27, 15, 25
5, 21, 34, 24, 255, 16, 29, 14, 255, 7, 18, 5, 255, 2, 11, 1, 255, 1, 12, 1, 255, 1, 13, 1,
255, 4, 17, 4, 255, 6, 17, 4, 255, 5, 16, 3, 255, …]
```

Task(n/o)-DDS与PVR纹理

相关代码

```
this.testList.push(new CompressedTextureTest());
```

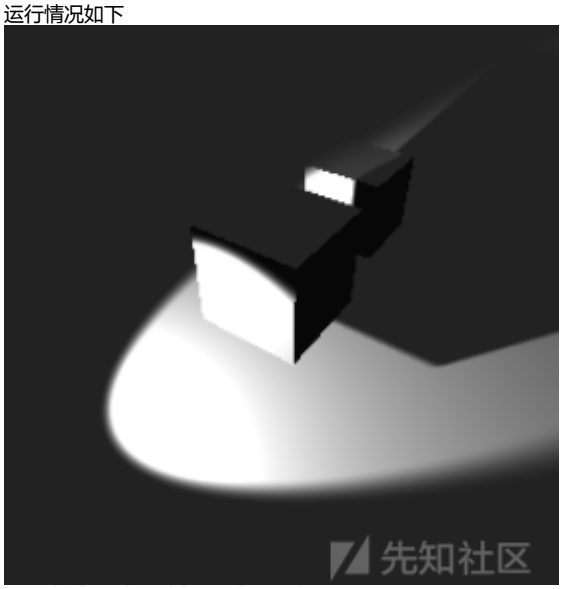
运行情况如下，因为有2种纹理(DDS与PVR纹理)，所以得到2组数据




```
1                                                                    toServer.js:193
                                                                     toServer.js:194
  Uint8Array(262144) [5, 5, 5, 255, 5, 5, 5, 255, 5, 5, 5, 255, 5, 5, 5, 255, 5, 5, 5, 255, 5,
    5, 5, 255, 5, 5, 5, 255, 5, 5, 5, 255, 5, 5, 5, 255, 5, 5, 5, 255, 5, 5, 5, 255, 5, 5, 5,
  ▶ 255, 5, 5, 5, 255, 5, 5, 5, 255, 5, 5, 5, 255, 5, 5, 5, 255, 5, 5, 5, 255, 5, 5, 5, 255, 5,
    5, 5, 255, 5, 5, 5, 255, 5, 5, 5, 255, 5, 5, 5, 255, 5, 5, 5, 255, 5, 5, 5, 255, 5, 5, 5,
    255, …]
2                                                                    toServer.js:193
                                                                     toServer.js:194
  Uint8Array(262144) [5, 5, 5, 255, 5, 5, 5, 255, 5, 5, 5, 255, 5, 5, 5, 255, 5, 5, 5, 255, 5,
    5, 5, 255, 5, 5, 5, 255, 5, 5, 5, 255, 5, 5, 5, 255, 5, 5, 5, 255, 5, 5, 5, 255, 5, 5, 5,
  ▶ 255, 5, 5, 5, 255, 5, 5, 5, 255, 5, 5, 5, 255, 5, 5, 5, 255, 5, 5, 5, 255, 5, 5, 5, 255, 5,
    5, 5, 255, 5, 5, 5, 255, 5, 5, 5, 255, 5, 5, 5, 255, 5, 5, 5, 255, 5, 5, 5, 255, 5, 5, 5,
    255, …]
```

Task(p)-浮点纹理

运行情况如下




```
1                                                                    toServer.js:193
                                                                     toServer.js:194
  Uint8Array(262144) [34, 34, 34, 255, 34, 34, 34, 255, 34, 34, 34, 255, 34, 34, 34, 255, 34,
    34, 34, 255, 34, 34, 34, 255, 34, 34, 34, 255, 34, 34, 34, 255, 34, 34, 34, 255, 34, 34,
  ▶  255, 34, 34, 34, 255, 34, 34, 34, 255, 34, 34, 34, 255, 34, 34, 34, 255, 34, 34, 34, 255, 3
    4, 34, 34, 255, 34, 34, 34, 255, 34, 34, 34, 255, 34, 34, 34, 255, 34, 34, 34, 255, 34, 34,
    34, 255, 34, 34, 34, 255, 34, 34, 34, 255, 34, 34, 34, 255, 34, 34, 34, 255, …]
```

Task(r)-字符集

为了获取浏览器支持的字符集列表，例如拉丁语，中文或是阿拉伯语。因为目前没有浏览器提供接口去获取其支持的字符集，于是我们想到了一个侧信道的方式去探测，方法
每一种字符集都会被浏览器进行渲染，如果该字符集被支持，那么浏览器就会渲染成功。否则，如果字符集不被支持，那么就会渲染出方块。如下图：

我们容易知道在这个被测试的浏览器中，我们容易知道，Javanese，Sudanese，Lontara和Thaana不被支持

后记

可以看到这些任务深入研究了图片渲染引擎的特征，js没办法直接获取到显卡的设置和驱动，但是通过这种方法，当不同的显卡渲染同一张图片时，因设置不同，渲染出来的