

---

# 计算机图形学 SSDO 大作业技术报告

小组成员：王思远（信科 2000013180），王小翔（信科 2000013146）

## 1. 概述

在常规的冯氏光照或布林-冯氏光照模型中，光照被分为环境光、漫反射光、镜面反射光三个光照分量。对于其中的漫反射光和镜面反射光，渲染过程中使用了基于物理的光线模拟，以此来达到近似真实世界中的漫反射与镜面反射的效果。在模拟环境光时，使用的是固定不变的光照常量。但是在真实世界中，光线会以任意方向进行散射，它的强度和颜色不是一成不变的。

在褶皱、缝隙、墙角以及孔洞等位置，散射光线较难进入，因此在现实中它们应该比周围的物体更暗。对于这些物体，在渲染过程中也应该使其变暗，计算机图形学中将其称为 Ambient Occlusion(环境光遮蔽)，简称为 AO。SSAO(Screen Space Ambient Occlusion，屏幕空间环境光遮蔽)即为屏幕空间中计算 AO 的一种高效的实现方式，首次使用 SSAO 的是 2007 年由游戏公司 Crytek 开发的电脑游戏《孤岛危机 1》。

天空的颜色不是一成不变的，它理应影响到物体看上去的颜色（例如日落时分，物体朝向晚霞的部分理应比背向晚霞的部分更亮）；物体看上去的颜色还会受到其附近物体反射的光线的影响（例如，在一个黄色的物体附近，一个蓝色的物体理应受到其反射光影响而显得有些发绿）。在 2011 年，Crytek 在其新作《孤岛危机 2》带来了 SSAO 的扩展处理方式，即 SSDO(Screen Space Directional Occlusion，屏幕空间定向遮蔽)，模拟了上面所说的两种光效，使画面的真实感又进一步提升。与 SSAO 相比，SSDO 只增加了一点点开销，近似表现了屏幕空间的直接和一次弹射光线，且能与其他方法结合模拟微表面结构。与 SSAO 一样，SSDO 作为屏幕空间算法，不依赖于场景的几何复杂度，效率较高。可以说，SSDO 就是 SSAO 增加了最新发展的技术，利用 SSAO 处理中已经获得的信息，计算两个能进一步提高真实性且没有过大开销的效果：来自天空盒的直接光照、来自周围

---

物体的间接光照。

## 2. 示例场景

我们用于展示和对比效果而使用的示例场景包括一个飞船模型、围绕该模型随机生成的 8 个颜色随机的点光源和一个天空盒。

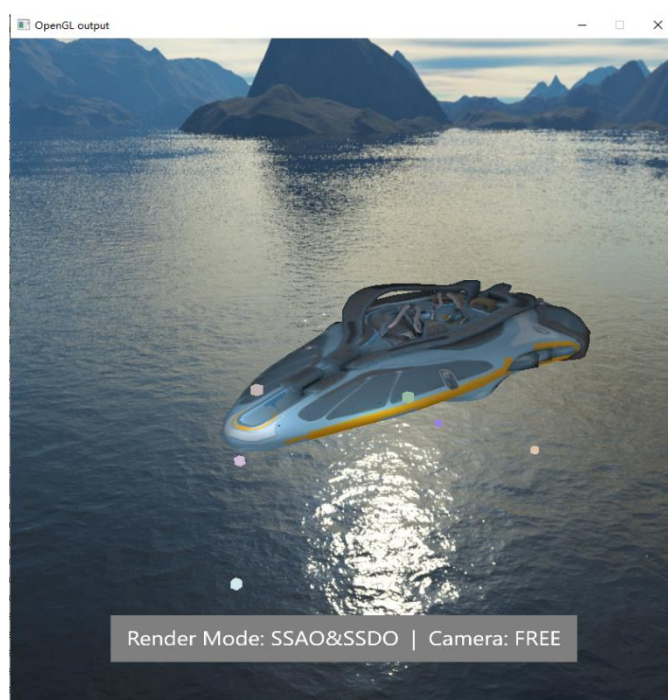


图 1 示例场景

## 3. 基本原理与效果对比

计算 SSAO 的基础是延迟着色法，也叫延迟渲染法。所谓延迟着色法，就是先渲染过程拆成两个处理阶段：在第一个几何处理阶段中，我们开启深度测试，然后先进行一次渲染，将几何信息和用于大计算量的渲染的信息存储于一个叫做 G 缓冲（G-Buffer）的帧缓冲中，因为开启了深度测试所以只保留了最上层未被遮挡的片元；在第二个光照处理阶段中，我们才进行大部分计算量非常大的渲染（像是光照）。与之前的常规渲染方法相比，延迟着色法不会对被遮挡住的物体计算光照和颜色（之前的常规渲染方法，会绘制被遮挡住的物体的颜色，但这是无意义且耗时的），因此在复杂的场景中产生了很大的优化。

SSAO 技术便基于延迟着色法。对于延迟着色法获取的 G 缓冲中通过深度测

---

试的每一个片段，SSAO 技术都会根据周边深度值计算一个遮蔽因子。这个遮蔽因子之后会被用来减少该片段的环境光照分量。遮蔽因子是通过采集片段周围半球型核心的多个深度样本，并和当前片段深度值对比而得到的。高于片段深度值样本的个数越多，该位置就更可能处于褶皱、缝隙、墙角等位置，遮蔽因子也就越大。

SSDO 是 SSAO 的延伸，但并不是 SSAO 的超集。SSDO 的提出者们认为，SSAO 技术采样了许多样本，但是只用来计算遮蔽因子，稍显浪费。对于高于片段深度值（被遮挡）的样本，将其对应表面视作虚拟点光源，可以用来计算附近物体反射的光线产生的效果；对于低于片段深度值（未被遮挡）的样本，可以将其视为天空盒对应点带来的虚拟点光源，计算天空盒光照产生的效果。

在图 2 的展示中，“Render Mode: OFF” 的展示为普通延迟着色的效果，“Render Mode: SSAO” 的展示为使用 SSAO 的效果，而“Render Mode: SSDO” 的展示为单纯使用 SSDO 的效果，“Render Mode: SSAO&SSDO” 为 SSAO 与 SSDO 结合使用的效果。

可以看到，开启 SSAO 后，模型中角落的地方会变得更加黑暗，从而产生更加真实的效果。开启 SSDO 后，由于光照还考虑了周围物体（间接光照），角落的地方会显得更亮，其他地方也会因为考虑了天空盒的直接光照而更亮一些。SSAO 与 SSDO 相结合，会得到比单纯的 SSAO 更好更真实的效果。更为详细完整的结果展示位于该报告的最后部分。



图 2 四种渲染模式下的效果对比

## 4. 设计与实现







### 4.1. 数据结构与整体流程

#### 4.1.1. 数据结构

##### ①渲染类

多个 Renderer，具体代码在\Complete\_project\src 下。将渲染封装成类。


---

|   |                |              |       |
|---|----------------|--------------|-------|
|  <code>renderer_both.h</code>      | 2022/7/1 16:06 | C/C++ Header | 18 KB |
|  <code>renderer_cube_quad.h</code> | 2022/7/1 1:07  | C/C++ Header | 6 KB  |
|  <code>renderer_image.h</code>     | 2022/7/1 15:38 | C/C++ Header | 5 KB  |
|  <code>renderer_off.h</code>       | 2022/7/1 16:28 | C/C++ Header | 9 KB  |
|  <code>renderer_ssao.h</code>      | 2022/7/1 15:58 | C/C++ Header | 14 KB |
|  <code>renderer_ssdo.h</code>      | 2022/7/1 16:03 | C/C++ Header | 14 KB |

(1) `renderer_off.h` 为延迟渲染时，使用的渲染类。(2) `renderer_ssao.h` 为 SSAO 使用的渲染类。(3) `renderer_ssdo.h` 为 SSDO 使用的渲染类。(4) `renderer_both.h` 为 `renderer_ssao.h` 和 `renderer_ssdo.h` 的结合，即 SSAO 和 SSDO 同时使用的渲染类。(5) `renderer_cube_quad.h` 用于渲染立方体和矩形。(6) `renderer_image.h` 用于渲染提示信息(包括当前渲染模式和相机是否可动)。

## ②模型类

包括 `mesh` 和 `model` 两个类，用于读取模型。

|  |                 |              |       |
|--|-----------------|--------------|-------|
|  <code>utils_mesh.h</code>  | 2022/6/30 22:26 | C/C++ Header | 6 KB  |
|  <code>utils_model.h</code> | 2022/7/1 0:56   | C/C++ Header | 10 KB |

## ③相机类与点光源类

相机在小作业中已经实现，这里不再赘述；点光源类中包含位置和颜色两个属性，较为简单。

|   |                 |              |      |
|---|-----------------|--------------|------|
|  <code>utils_camera.h</code> | 2022/7/1 0:11   | C/C++ Header | 6 KB |
|  <code>utils_light.h</code>  | 2022/6/30 22:28 | C/C++ Header | 1 KB |

## ④着色器类

我们将着色器的读取、链接、使用、清除等封装成一个类，方便使用，简化代码。

|   |                 |              |      |
|---|-----------------|--------------|------|
|  <code>utils_shader_program.h</code> | 2022/6/30 22:11 | C/C++ Header | 6 KB |
|---|-----------------|--------------|------|

# 4.1.2. 整体流程

这里以 SSAO 技术与 SSDO 技术结合使用的渲染模式为例，简要介绍程序渲染图形的流程。

①几何处理阶段：我们先渲染场景一次，获取对象的各种几何信息（位置向量、法向量、颜色向量和镜面值），并储存在 G 缓冲中。这些储存在 G 缓冲中的几何信息将会在之后用来做遮蔽因子和光照的计算。

②SSAO：使用片段周围的多个采样点的信息计算 SSAO 遮蔽因子纹理。

③模糊 SSAO：通过模糊处理 SSAO 得到的结果，消除噪声。

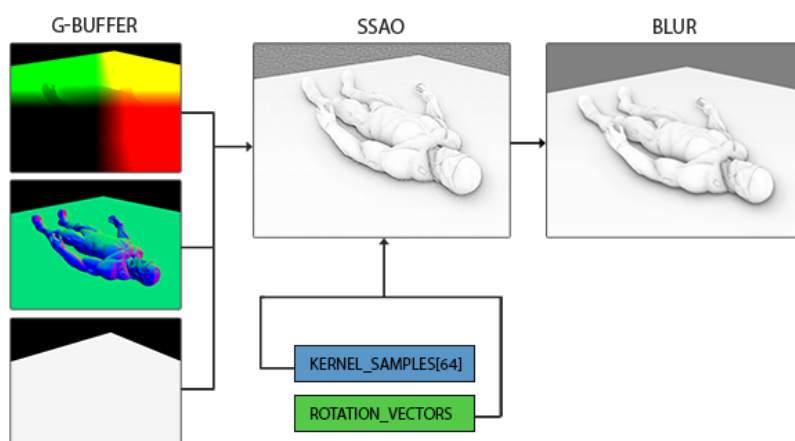


图 3 前三个阶段（SSAO）

④SSDO：类似于第②阶段，计算 SSDO 纹理。

⑤模糊 SSDO：类似于第③阶段，通过模糊处理 SSDO 得到的结果，消除噪声。

⑥光照处理阶段：利用储存的深度信息以及前面得到的几何信息、SSAO 纹理和 SSDO 纹理，计算混合光照。

⑦绘制灯（点光源）。

⑧绘制天空盒。

## 4.2. 具体设计和实现

### 4.2.1. 计算 SSAO

（1）首先，SSAO 需要获取几何信息来进行后续的计算和处理。对于每一个片段，我们将需要以下数据：

- ①逐片段位置向量
- ②逐片段的法线向量
- ③逐片段的反射颜色
- ④采样核心
- ⑤用来旋转采样核心的随机旋转矢量

前三点很好理解。需要采样核心，是因为我们需要沿着表面法线方向，生成大量的样本，且我们希望在半球形中生成样本，所以需要设计沿着法线的半球形采样核心。利用在观察空间中的逐片段法线纹理，我们可以将半球形采样核心对



准该片段的观察空间表面法线。为了引入一些随机性，采样核心会根据一个随机的旋转矢量稍微偏转一点。对于之后每一个采样得到的样本我们会利用线性深度纹理来比较结果。

(2) 我们先在切线空间 (Tangent Space) 内生成采样核心，法向量将指向正 z 方向。

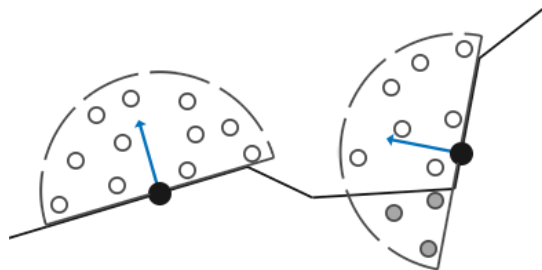


图 4 采样核心

我们在切线空间中以-1.0 到 1.0 为范围变换 x 和 y 方向，并以 0.0 和 1.0 为范围变换样本的 z 方向(如果以-1.0 到 1.0 为范围，取样核心就变成球型了)。由于采样核心将会与表面法线对齐，因此得到的样本将会在垂直表面的半球里。

现在，得到的所有的样本都是随机地平均分布在采样核心里的，但是我们想将更多的注意放在靠近真正片段（位于采样核心的原点）上，也就是让得到的样本稍微靠近采样核心的原点。我们可以用一个加速插值函数实现它：

```
#define MY_LERP(a, b, f) ((a) + (f) * ((b) - (a)))
```

```
// scale samples s.t. they're more aligned to center of kernel  
scale = MY_LERP(0.1f, 1.0f, scale * scale);  
sample *= scale;  
ssaoKernel.push_back(sample);
```

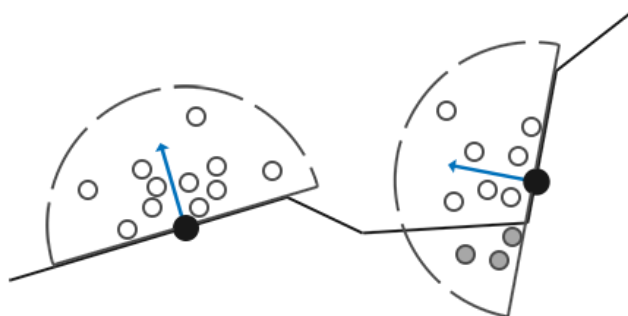


图 5 大部分样本靠近原点的采样核心

(3) 之后，通过引入一些随机性到采样核心上，我们可以大大减少获得较好效果所需的样本数量。具体做法是创建一个小的随机旋转向量纹理(噪声纹理，这里我们设其大小为 4x4)平铺在屏幕上，最后施加于采样核心上，即让每个采样核心都随机旋转一个角度。

注意，我们是将一个小的随机旋转向量平铺在屏幕上，然后施加于采样核心，因此对应于同一个随机旋转值的采样核心(例如坐标 1,1 和坐标 1,5)，它们的随机旋转是相同的。

(4) 利用前面得到的采样核心和几何信息，在 2D 的铺屏四边形上运行 SSAO 着色器，它对于每一个生成的片段计算遮蔽因子(为了在最终的光照着色器中使用)，并存储在一个帧缓冲对象中。具体过程如下：

①SSAO 着色器读取 G 缓冲纹理(包括线性深度)、噪声纹理和法向半球核心样本作为输入参数；

②构建 TBN 矩阵将向量从切线空间变换到观察空间；

③变换样本到屏幕空间，从而我们就可以直接取样样本的线性深度值；

④检查样本的当前深度值是否大于存储的深度值，如果是，则遮蔽因子加一；

⑤最后将遮蔽因子根据采样核心的大小标准化，最终结果以下图为例。



图 6 SSAO 纹理(未进行模糊处理)



### 4.2.2. 对 SSAO 结果做模糊处理

从上面的结果可以看出，重复的噪声纹理在图中清晰可见（图中的许多小方块）。为了创建一个光滑的环境遮蔽结果，我们需要模糊环境遮蔽纹理。

由于平铺的随机向量纹理保持了一致的随机性，我们可以使用这一性质来创建一个简单的模糊着色器：

```
8 void main()
9 {
10     vec2 texelSize = 1.0 / vec2(textureSize(ssdoInput, 0));
11     vec3 result = vec3(0.0, 0.0, 0.0);
12     for (int x = -2; x < 2; ++x)
13     {
14         for (int y = -2; y < 2; ++y)
15         {
16             vec2 offset = vec2(float(x), float(y)) * texelSize;
17             result += texture(ssdoInput, TexCoords + offset).rgb;
18         }
19     }
20     FragColor = result / (4.0 * 4.0);
21 }
```

这里我们遍历了周围在-2.0 和 2.0 之间的 SSAO 纹理单元，采样与噪声纹理维度（4x4）相同数量的 SSAO 纹理，将所得的结果取平均值，获得一个简单但是有效的模糊效果：

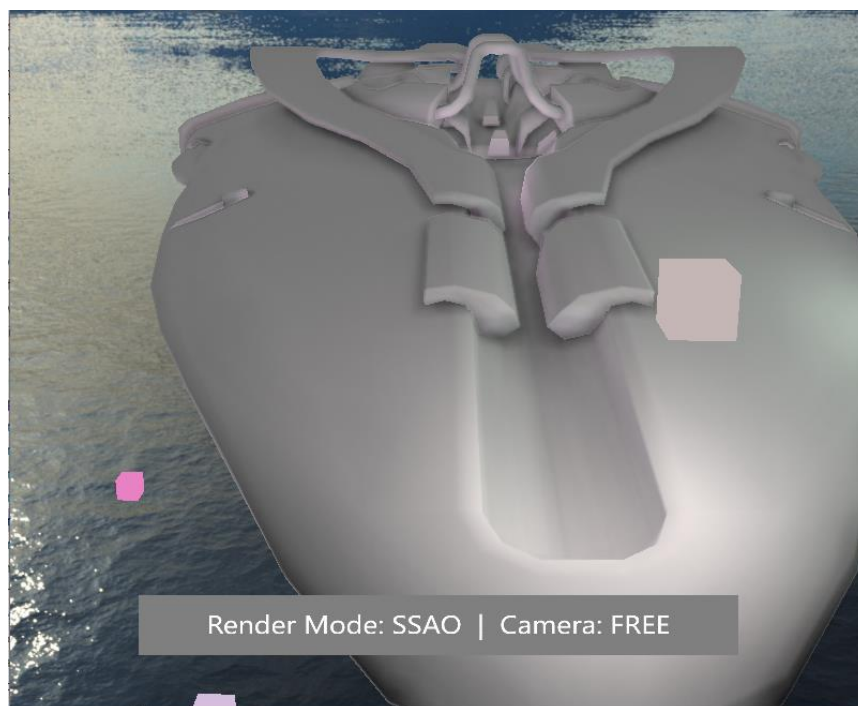


图 7 模糊处理后的 SSAO 结果

可以看到，与模糊处理前的结果相比，虽然清晰度有所损失，但是结果中的噪声被消除了，这就获得了一个包含逐片段环境遮蔽数据的纹理。

### 4.2.3. 计算 SSDO 和模糊处理

SSDO 的计算过程与 SSAO 很相似，区别在于：SSAO 计算的是遮蔽因子，给物体施加阴影，而 SSDO 计算来自天空盒的直接光照和来自周围物体的间接光照。

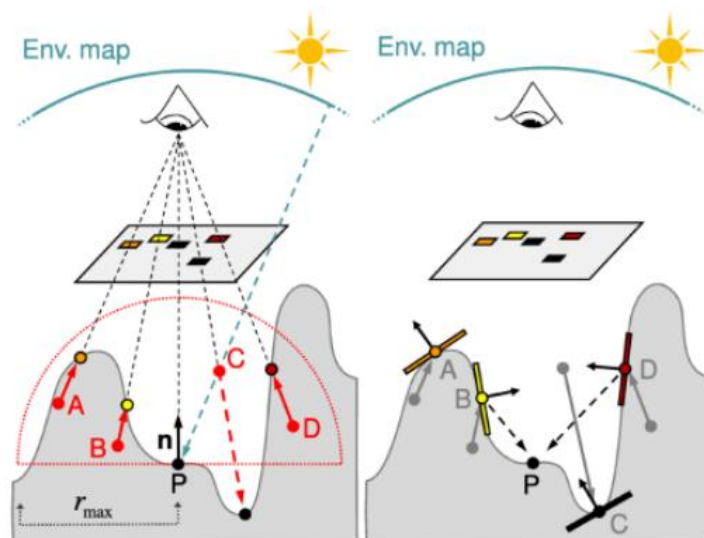


图 8 SSDO

以上图为例，对 P 点进行半球形采样，样本点为 ABCD 四点，在 SSAO 计算中，ABD 三点被遮挡，会使得 P 的遮蔽因子变大，而 C 没有产生任何作用。而在 SSDO 计算中，ABD 三点被遮挡，所以其对应表面视作虚拟点光源（颜色为 ABD 三点分别对应的表面的颜色），计算它们对 P 点的光照（间接光照）；而 C 点未被遮挡，可以视作由天空盒带来的虚拟点光源（颜色为 PC 连线与天空盒相交点的颜色）。最后间接光照与直接光照相加，再根据采样核心的大小标准化，便得到 SSDO 的结果。

```
float rangeCheck = smoothstep(0.0, 1.0, radius / abs(fragPos.z - sampleDepth));
if (sampleDepth >= samplePos.z)
{
    indirectLight += rangeCheck * max(dot(sampleNormal, normalize(fragPos - samplePos1)), 0.0) * sampleColor;
}
else
{
    vec4 skyboxDirection = iview * vec4(samplePos - fragPos, 0.0);
    vec3 skyboxColor = texture(skybox, skyboxDirection.xyz).xyz;
    directLight += rangeCheck * skyboxColor * dot(normal, normalize(samplePos - fragPos));
}
```

和 SSAO 一样，我们需要对 SSDO 得到的结果进行模糊处理，处理方式也和 SSAO 的模糊处理的方式一样。

#### 4.2.4. SSAO 和 SSDO 作用于光照计算

前面得到了 SSAO 和 SSDO 的结果。接下来便是将其作用于最终的光照计算中。应用到光照方程中极其简单：我们要做的只是稍微修改下光照方程。

```
vec3 lighting = vec3(Diffuse * 0.5 * AmbientOcclusion + DirectionalOcclusion);
```

上式中，Diffuse 为漫反射光照，AmbientOcclusion 为 SSAO 遮蔽因子，DirectionalOcclusion 为 SSDO 计算的直接光照和间接光照。

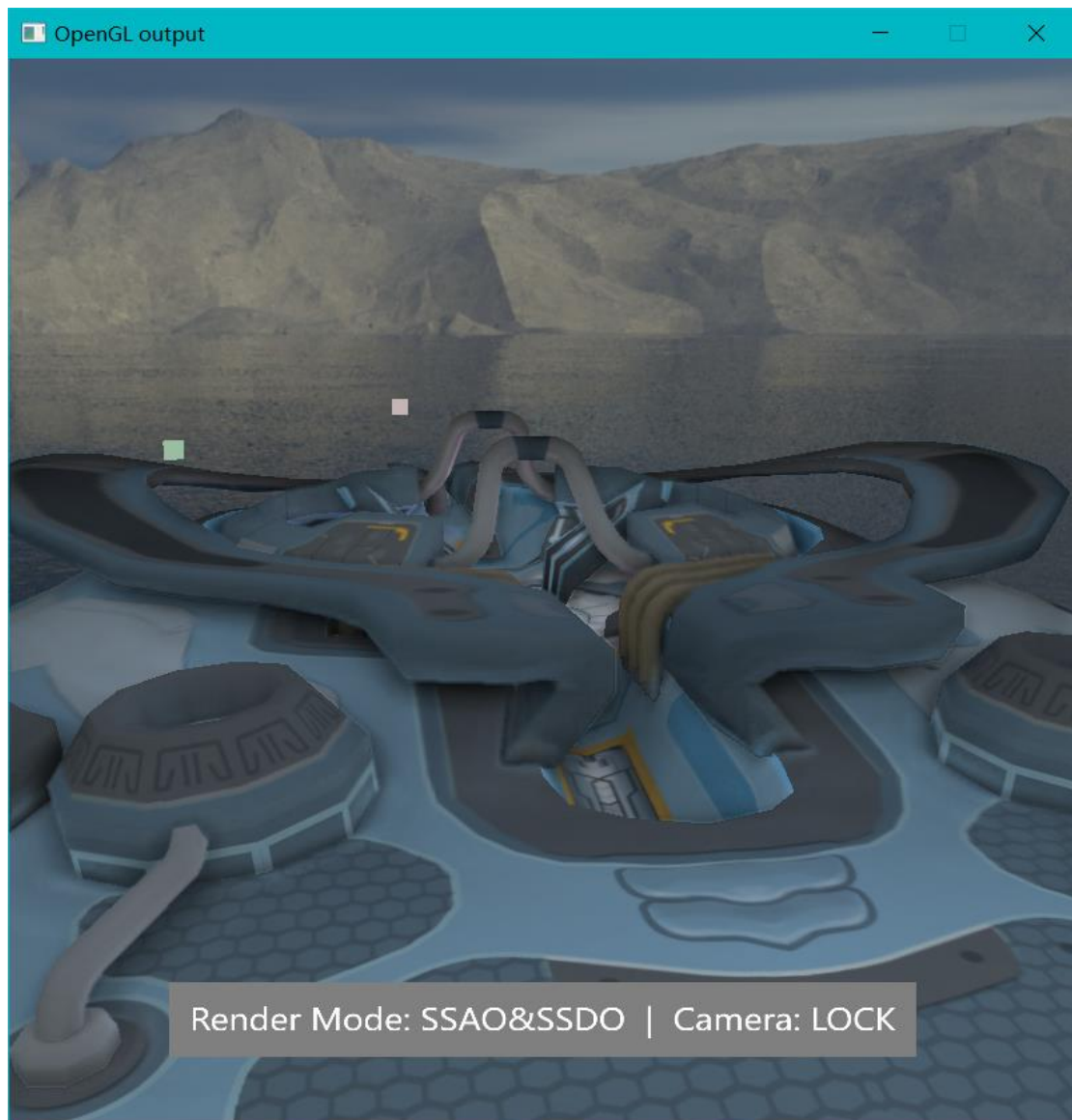


图 9 最终效果 (SSAO+SSDO)

可以看到，上图中缝隙中显出黑色阴影，即为 SSAO 得到的效果。上图黄色管子的下端部分明显可以看到蓝色墙壁对其的蓝色反光，蓝色墙壁在黄色管子照亮下也稍显绿色，这即为 SSDO 中的间接光照的效果。由于默认天空盒的颜色不

够明显，上图不能很明显地看出来 SSDO 的直接光照的效果。下面我们将天空盒的贴图进行替换后，就可以看出来直接光照的效果了（见 4.2.5 节）。

至此，我们成功地实现了 SSAO 和 SSDO，并将其应用于光照渲染中。

#### 4.2.5. 额外细节：天空盒和灯（点光源）

如前所述，SSDO 中包含来自天空盒的直接光照和来自周围物体的间接光照。因此，我们还绘制了天空盒，并将其应用于 SSDO 的计算中。天空盒贴图位于 data\skybox 文件夹中。在该天空盒下，SSDO 的直接光照效果并不明显。



如果将天空盒贴图的顶部和底部贴图更换成纯色图片，便可以看到更为明显的结果。更换的图片位于 data\skybox\skybox\_redblue 文件夹中。



在更换后的天空盒下，飞船模型下侧会被天空盒底部照成红色，上侧会被天空盒顶部照成蓝色。可以明显地看出 SSDO 的直接光照效果。

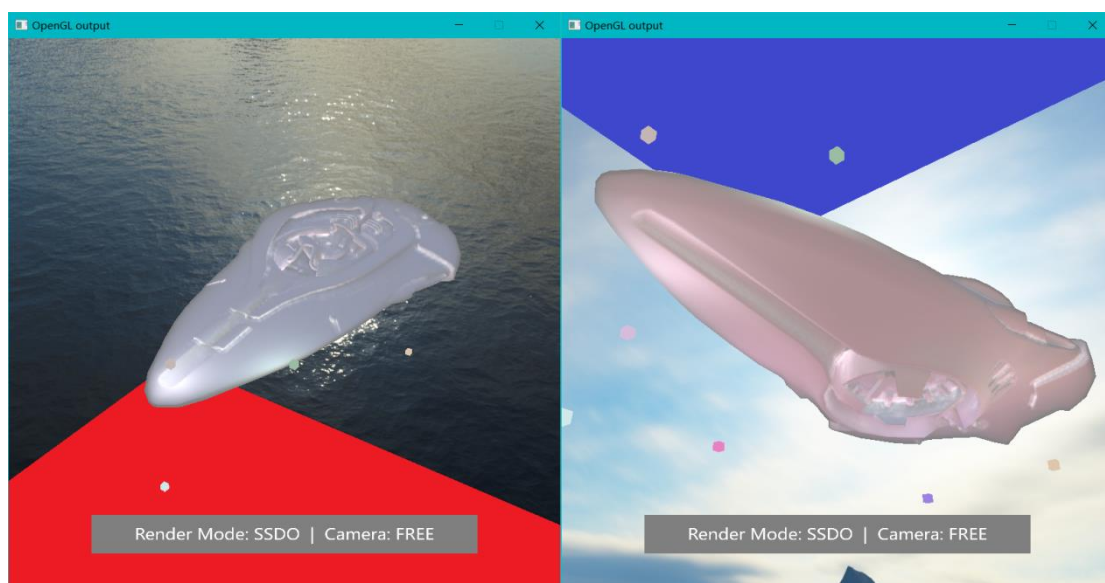


图 10 更换天空盒的效果



至于点光源，我们添加它们只是为了提供一些简单的光照，它们被绘制为围绕在飞船周围的有颜色的小立方体。

## 5. 结果展示与对比

以下结果展示中，左边的图片是有颜色（使用漫反射贴图和镜面反射贴图）的模型，右边的图是无颜色的白模（可以更明显地看出 SSAO 和 SSDO 技术的效果）；每一行对应一种渲染模式。



图 11 结果展示



续图 11 结果展示

对比无颜色的模型下，SSAO 和 OFF 的区别，可以看出来，SSAO 使得模型缝隙部分阴影更强，显得更有层次感。SSAO&SSDO 显得更明亮和更真实，因其加入了来自天空盒的直接光照和来自周围物体的间接光照。

## 6. 总结

我们复现了 SSAO 和 SSDO 的计算并将其应用于光照计算中，提升了画面的真实度，并在引入的飞船模型上取得了较好的效果。



---

## 参考资料

- [1] T Ritschel, T Grosch, HP Seidel. 2009. Approximating dynamic global illumination in image space. In Proceedings of the 2009 symposium on Interactive 3D graphics and games, 75-82.
- [2] LearnOpenGL, <https://learnopengl-cn.github.io/>