Screen Space Ambient Occlusion

屏幕空间环境光遮蔽

王龙 2017.3.23

目录

- 什么是AO？
- 什么是SSAO？
  - SSAO基本原理
  - 采样核（Sample Kernel）
  - AO算法
  - 模糊处理（Blur）
  - 混合结果
- Unity插件：Simple SSAO

什么是AO？

- AO，Ambient Occlusion， 环境光遮蔽
- 可以认为是一种全局光照模型
- 用来计算场景中物体表面任意一点暴露于环境光中的程度
- 使得渲染结果**更具层次感**

什么是AO？

AO = shadow created by ambient illumination

举个栗子

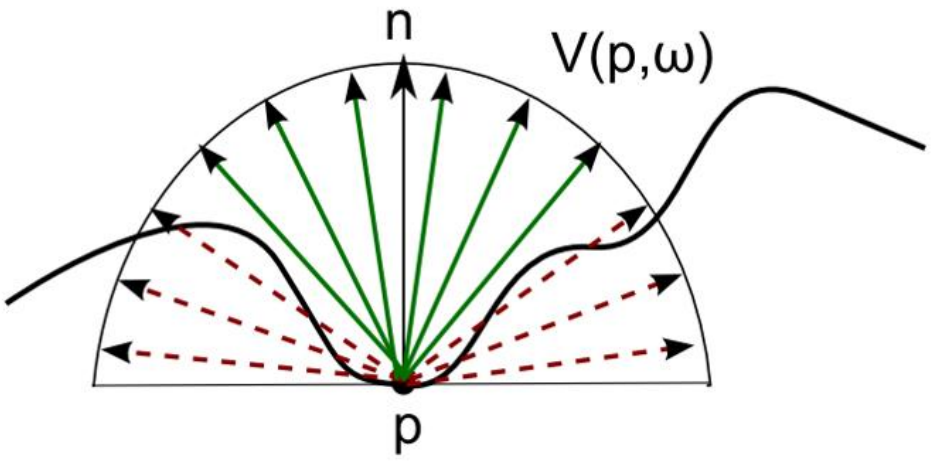
右图没有环境光产生的阴影，感觉很不真实，车像浮在空中。

举个栗子

左图只有直接光照产生的阴影，右图加上环境光产生的阴影，老头的皱纹更明显、更立体。

如何计算AO？

- $\hat{n}$：点P的法线
- Ω：切平面正方向的单位半球
- $\hat{\omega}$：点P到Ω表面的各个向量
- V：在$\hat{\omega}$方向点P的可见性函数
  - 如果点P被遮挡就是1，否则是0
  - 右图中，可以看到红色虚线被阻挡了，V=1；绿色实线没有被遮挡，V=0



$$A(P,\hat{n}) = \frac{1}{\pi}\int_{\Omega}(V(\hat{\omega},P))max(\hat{\omega}\cdot\hat{n},0)d\hat{\omega}$$

法线为$\hat{n}$的一点P，受到周边物体的遮挡值A的计算公式

如何计算AO？

- 一般离线渲染会采用Ray-Tracing(光线追踪)或是简化的Ray-Marching(所谓光线行进)算法, 模拟若干条射线以计算遮蔽百分比
- 不适合实时的图形渲染
- 存不存在一种近似的模拟？

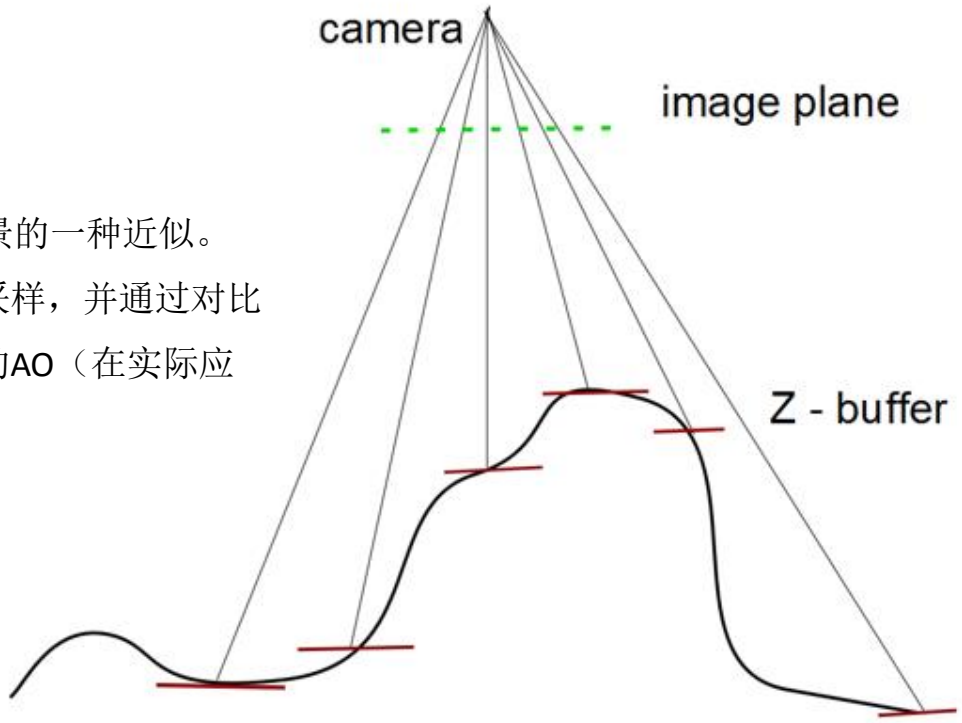
SSAO

- SSAO，Screen Space Ambient Occlusion，屏幕空间环境光遮蔽
- 实时高效的模拟环境光遮蔽的效果
- 由Vladimir Kajalin在Crytek工作时开发，并在2007年由Crytek开发Electronic Arts发行的电子游戏Crysis（孤岛危机）中第一次使用。

SSAO的原理

基本原理

- 将深度缓冲（depth buffer）看做是对场景的一种近似。
- 在**片元着色器**中，在每个像素周围随机采样，并通过对比当前像素和采样像素的**深度**来计算最终的AO（在实际应用中可能会使用稍微复杂一点的算法）。

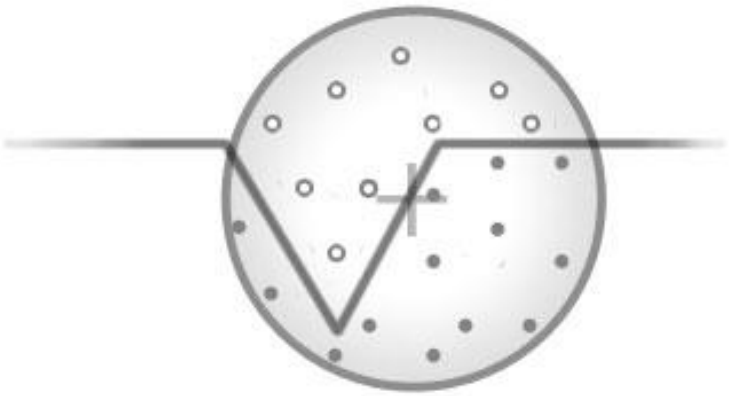


SSAO的原理

- 两个核心问题：
  - 在什么位置采样？——Sample Kernel
  - 如何判断当前像素是否被采样点遮蔽？——AO算法

Sample Kernel——Sphere

Crysis中的Sample Kernel是以顶点为圆心的球体

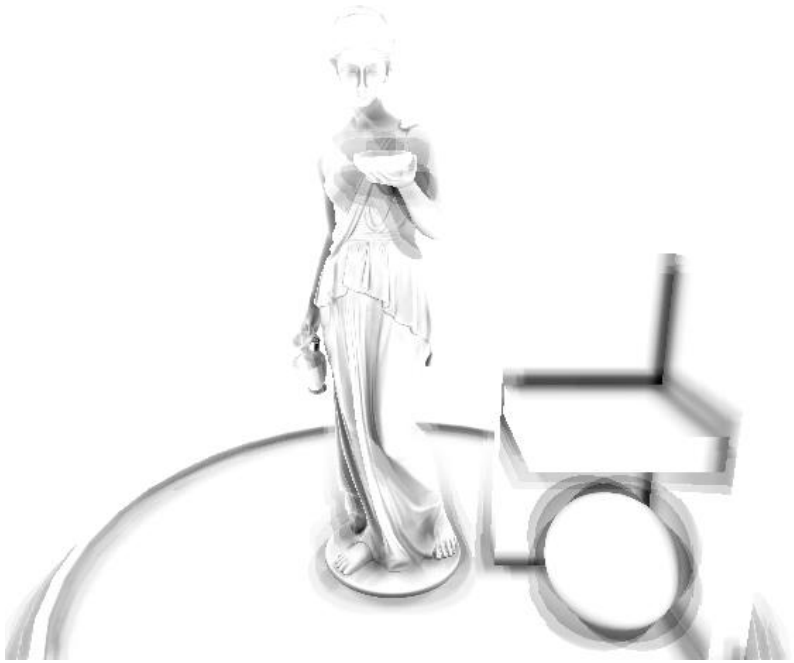


```
// define kernel
const half step = 1.f - 1.f/8.f;
half n = 0;
const half fScale = 0.025f;
const half3 arrKernel[8] =
{
        normalize(half3( 1, 1, 1))*fScale*(n+=step),
        normalize(half3(-1,-1,-1))*fScale*(n+=step),
        normalize(half3(-1,-1, 1))*fScale*(n+=step),
        normalize(half3(-1, 1,-1))*fScale*(n+=step),
        normalize(half3(-1, 1 ,1))*fScale*(n+=step),
        normalize(half3( 1,-1,-1))*fScale*(n+=step),
        normalize(half3( 1,-1, 1))*fScale*(n+=step),
        normalize(half3( 1, 1,-1))*fScale*(n+=step),
};
```

Crytek Engine Shader Source File: AmbientOcclusion.cfx

Sample Kernel——Sphere

low sample 'banding'

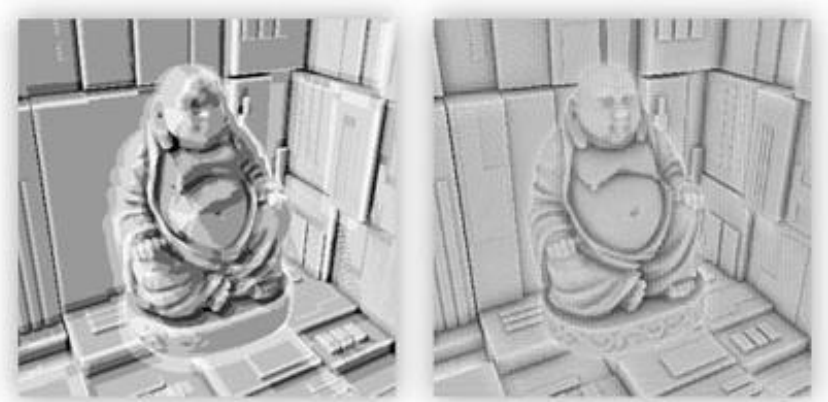


缺点一：

- 为了提升性能会尽力降低采样点的数量，但由于采样核固定，降低采样数时会出现“条带状（banding）” 的现象

Sample Kernel——Sphere

- 对每一个像素，采样时将采样核旋转随机的角度。
- 将旋转信息存储于一张贴图中（noise texture），平铺于整个屏幕，导致采样核的方向规律性的重复。
- 使贴图尽量小（比如4x4），使得这种规律重复变得高频，条纹转化成高频的**噪声（noise）**
- 噪声可后续通过模糊消除

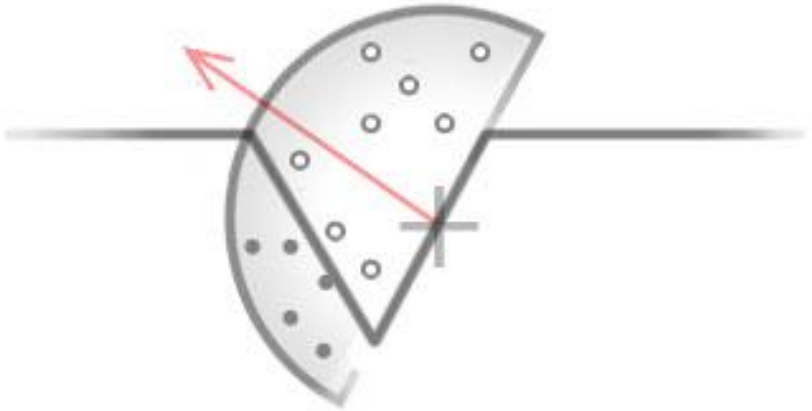


Sample Kernel——Sphere

缺点二：

- 平坦的墙面等物体看上去是灰的，因为有50%的采样点会落在墙内

Sample Kernel——Normal Oriented Hemisphere



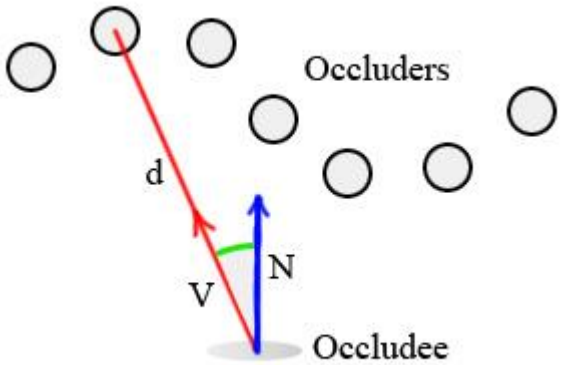
改进：

- 使用顶点法线方向的**半球**代替一整个球体

代价：

- 需要法线信息

AO算法



Occlusion = max(0.0, dot(N, V)) * (1.0 / (1.0 + d))

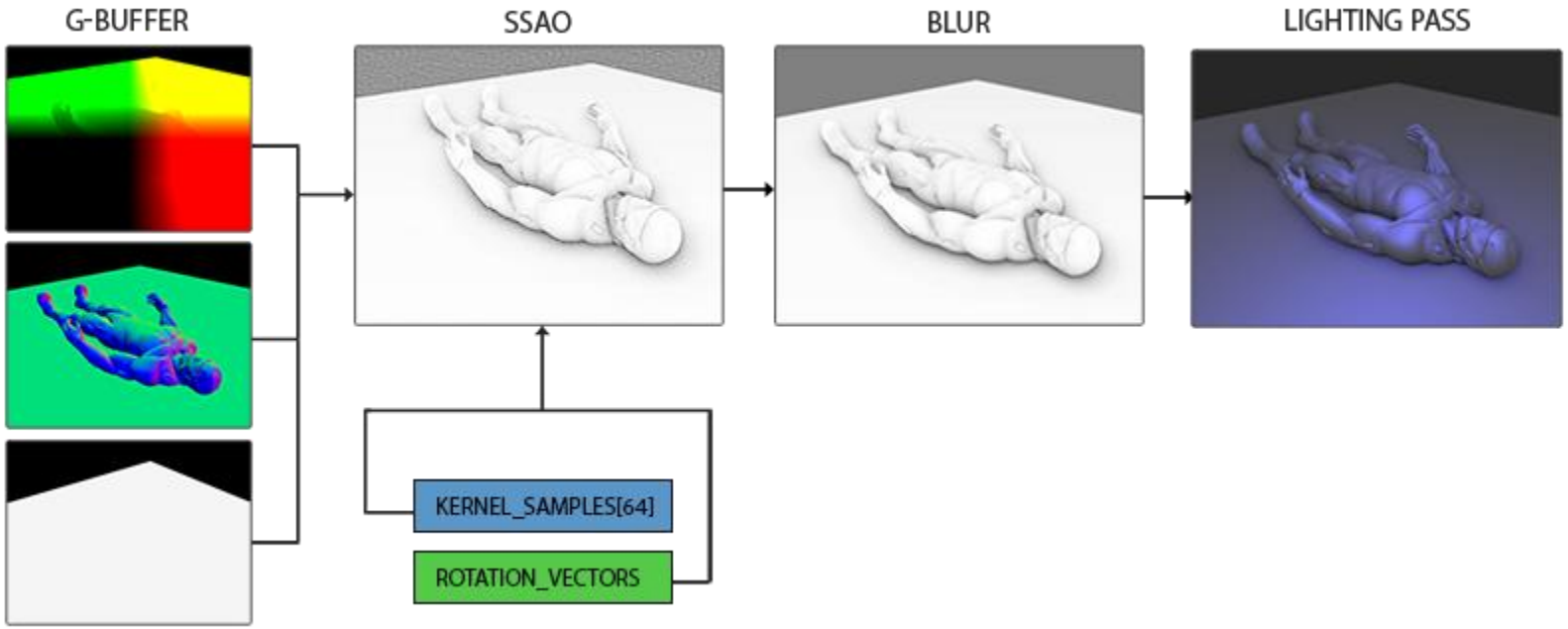
N：当前像素的法线方向（归一化）
V：场景中当前像素指向采样点的向量（归一化）
d：场景中采样点距离当前像素的距离

- 正上方的遮挡物比旁边的遮挡物贡献更多的AO
- 距离越近，贡献的遮挡越大，距离越远，贡献的遮挡越小
- 对每个采样点按照上述公式计算其AO贡献，然后根据采样点的数量取平均，得到当前像素最终的AO

模糊处理

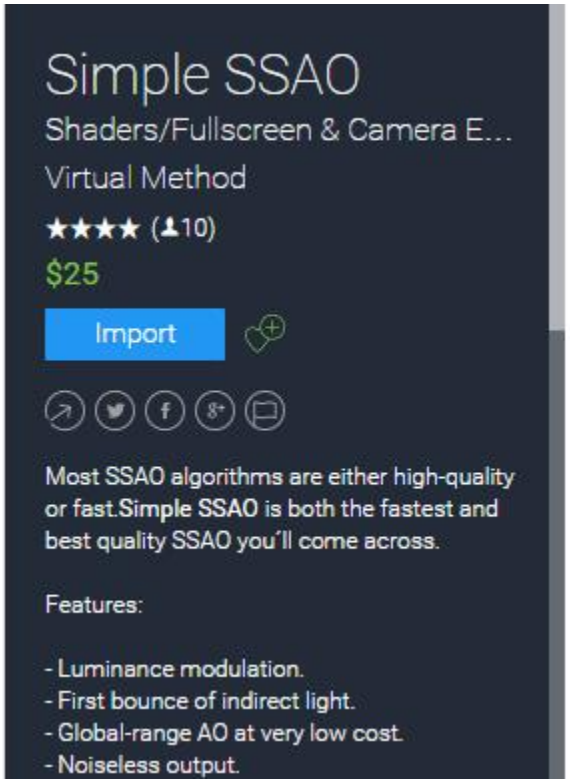
- 假如使用了4 x 4的noise texture来旋转采样核
- 对于每个像素，计算其周围4 x 4的范围内的像素值的平均值作为当前像素的最终值

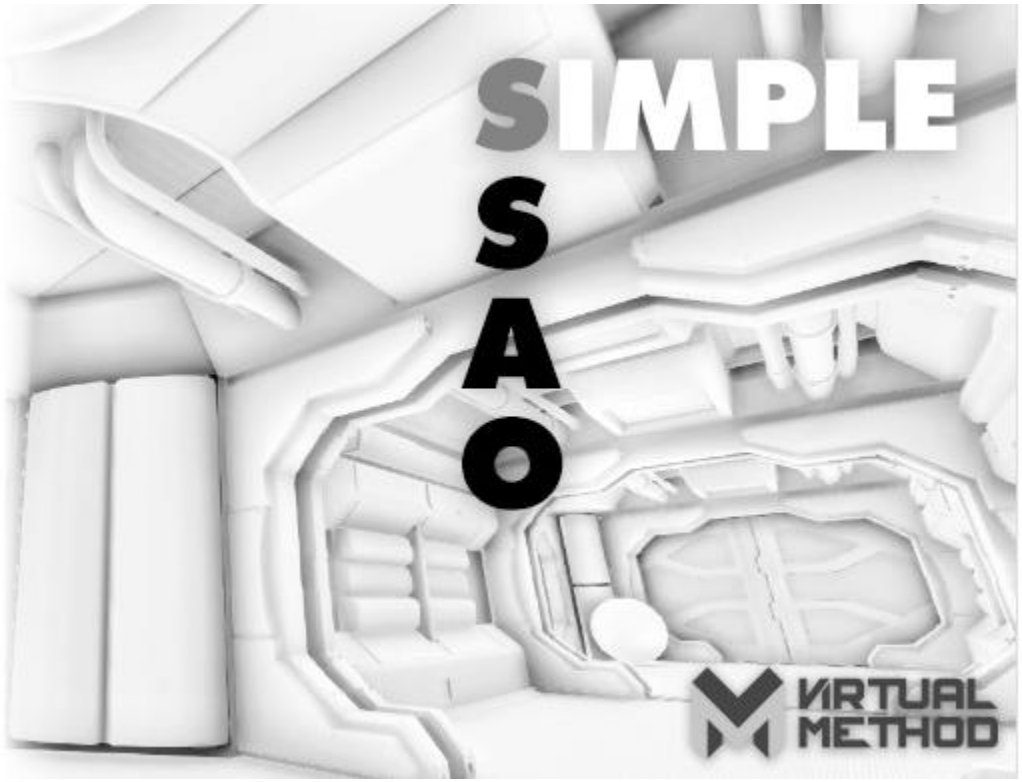
混合结果



将每个像素的AO计算结果，乘到像素原先颜色的RGB上即可。

Simple SSAO


Simple SSAO
Shaders/Fullscreen & Camera E...
Virtual Method
★★★★ (10)
$25
Import
Most SSAO algorithms are either high-quality or fast.Simple SSAO is both the fastest and best quality SSAO you'll come across.
Features:
- Luminance modulation.
- First bounce of indirect light.
- Global-range AO at very low cost.
- Noiseless output.


SIMPLE
SSAO
VIRTUAL METHOD


0 (绑 0)
156/240
活动
克雷孟特托蒂


0 (绑 0)
156/240
活动
克雷孟特托蒂

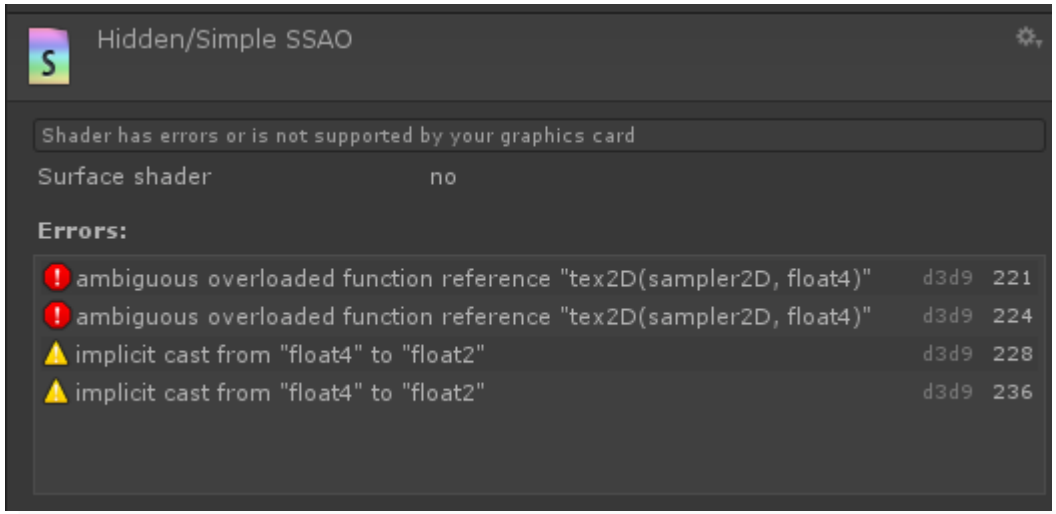

贵宾 0
0 (绑 0)
0 (绑 0)
156/240
1
战力 578
充值
首充大礼
开服庆典
15日活动
活动日常
埃兰商城
关卡
探险
战场
商店
即将到来
任务
组队招募
无妄之灾
通关-埃兰之灾
克雷孟特托蒂
克劳利伯爵
背包
技能
变身
符文
宠物
权杖
元素
公会
好友


贵宾 0
0 (绑 0)
0 (绑 0)
156/240
1
战力 578
充值
首充大礼
开服庆典
15日活动
活动日常
埃兰商城
关卡
探险
战场
商店
即将到来
任务
组队招募
无妄之灾
通关-埃兰之灾
克雷孟特托蒂
克劳利伯爵
背包
技能
变身
符文
宠物
权杖
元素
公会
好友

Simple SSAO

- 基于作者一篇“well known”的文章
- 需要Unity 5.5.1或更高（简单修改之后可支持Unity4）
- 在**屏幕空间**而不是摄像机空间内采样
- 模糊算法
- 可避免高光区域受SSAO影响（Luminance modulation）
- 支持一次光线反弹（Color Bleeding）
- 开放一堆参数微调

Simple SSAO——In Unity4



Aras

Graphics Plumber

Unity Technologies

Joined: Nov 7, 2005

Posts: 4,521

I think texture sampling needs to be:

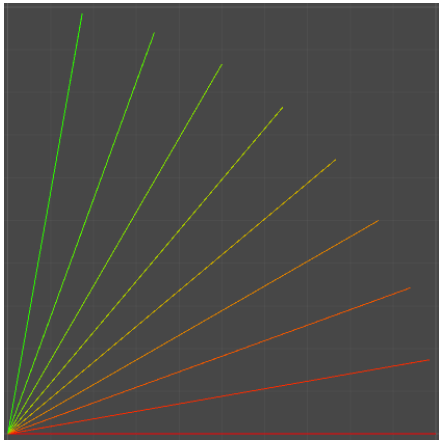
Code (csharp):

```
1. tex2D(_MainTex, i.uv.xy);
```

Since 'uv' is a float4, and sampling a 2D texture needs two coordinates, not four.

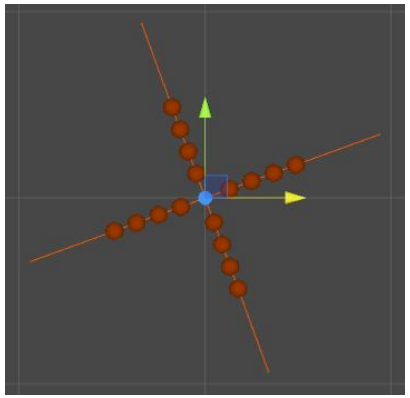
Aras, **Aug 19, 2008**

Simple SSAO——Sample



9个方向的采样轴（RG）
每个轴一个最小采样长度（B）

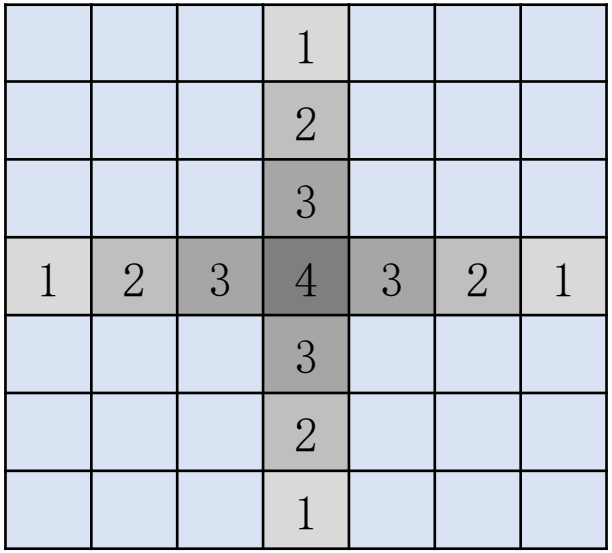
交替存储于3x3的贴图中
（AxisPattern.asset）



采样时每个采样轴扩展为
四个方向

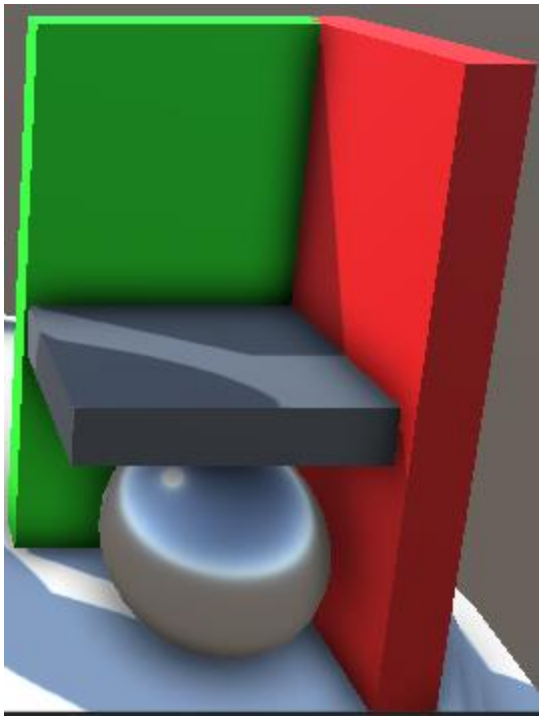
- 在屏幕空间（2D）采样，而不是在摄像机空间（3D）采样
- 根据每个像素的uv对AxisPattern采样，获取当前像素使用的采样轴
- 根据采样半径、当前像素深度对采样轴的长度进行缩放
- 在每个半轴上均匀采样

Simple SSAO——Blur

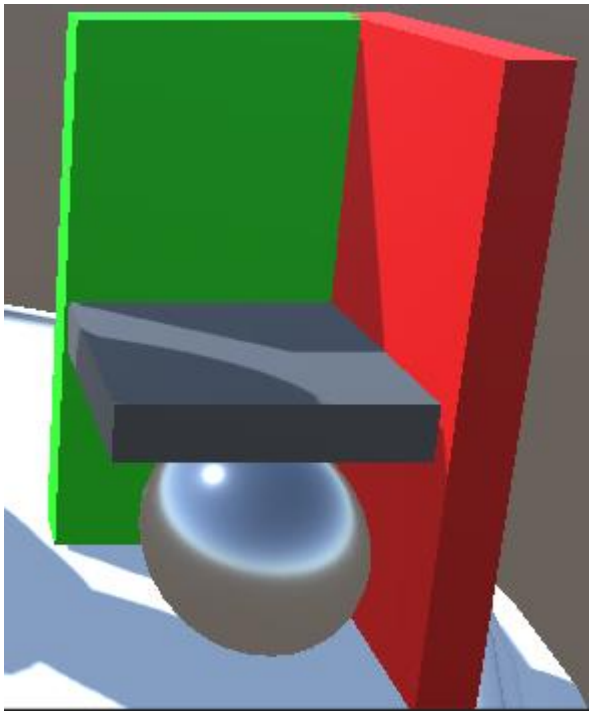


- 根据当前像素以及上下左右共13个像素进行加权平均
- 距离当前像素的距离是0、1、2、3的像素，权重依次是4、3、2、1
- 与当前像素的法线差值不超过指定阈值的才会被计算在内
- 与当前像素的深度差值不超过指定阈值的才会被计算在内

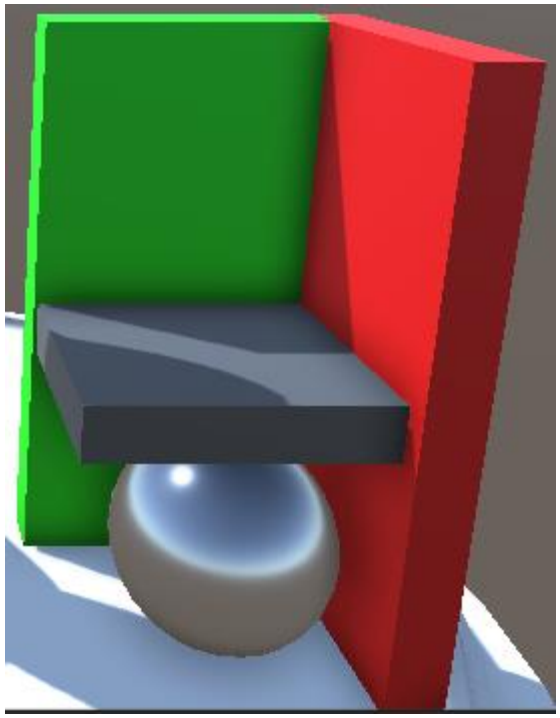
Simple SSAO——Luminance modulation



SSAO without Luminance modulation



No SSAO



SSAO with Luminance modulation

Simple SSAO——Luminance modulation

计算每个像素的亮度：

```
// 计算亮度（人眼对绿色最敏感，其次红色，最后蓝色）
half luminance = dot(tex2D (_ColorBuffer, i.uv).rgb, half3(0.299, 0.587, 0.114));
```

用像素亮度对AO进行线性插值，结果作为最后的AO：

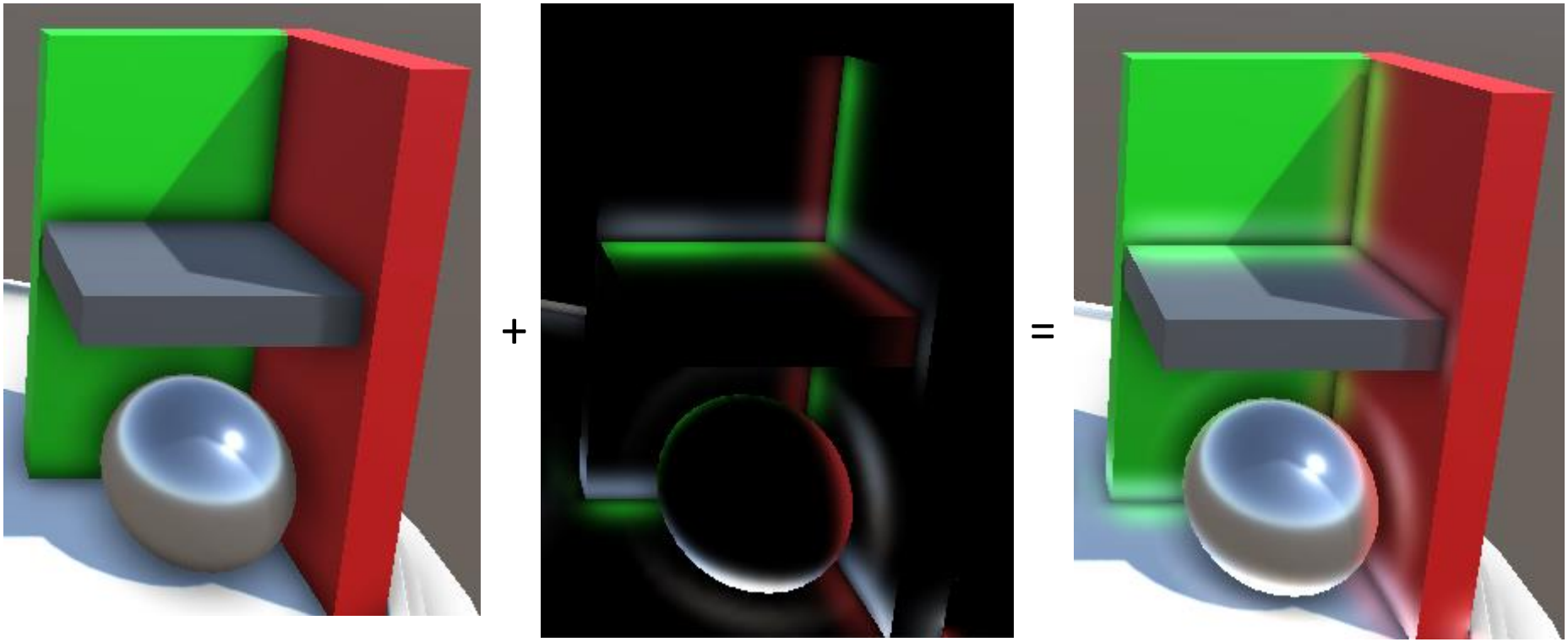
```
// _Params4.y = m_LuminanceModulation [0, 1]
gi.a = lerp(1 - gi.a, 1.0, luminance * _Params4.y);
```

Simple SSAO——Color Bleeding

- Wikipedia：
  - In computer graphics and 3D rendering, **color bleeding** is the phenomenon in which objects or surfaces are colored by reflection of colored light from nearby surfaces.

  - **Color Bleeding** - The transfer of color between nearby objects or, caused by the colored reflection of indirect light. This is a visible effect that appears when a scene is rendered with Radiosity or full global illumination, or can otherwise be simulated by adding colored lights to a 3D scene.

Simple SSAO——Color Bleeding



Simple SSAO——Color Bleeding

将每个采样点的颜色，按照一定比重，加到当前像素的颜色上：

- 采样点的法线越朝向当前像素，比重越大。
- 采样点越在当前像素的正上方，比重越大。
- 采样点距离当前像素越近，比重越大。

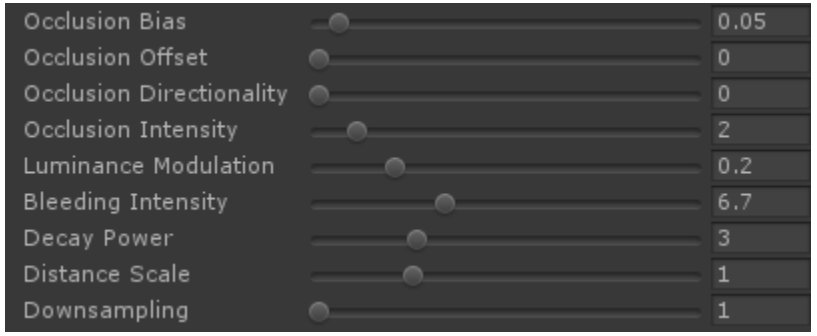
```
float3 v = sampleP - fragP;
float d = length(v);
v /= d;

float fvDot = dot(fragN,v);
float fsDot = dot(fragN,-sampleN);
float falloff = 1.0 + pow(d*_Params.w,_Params2.y);
float areaRatio = sampleP.z/fragP.z;
```

```
float nvDot = dot(sampleN,-v);
float3 gi = bleedColor * areaRatio * max(0,fvDot) * max(0,nvDot) /falloff*_Params3.x;
return half4(gi,ao);
```

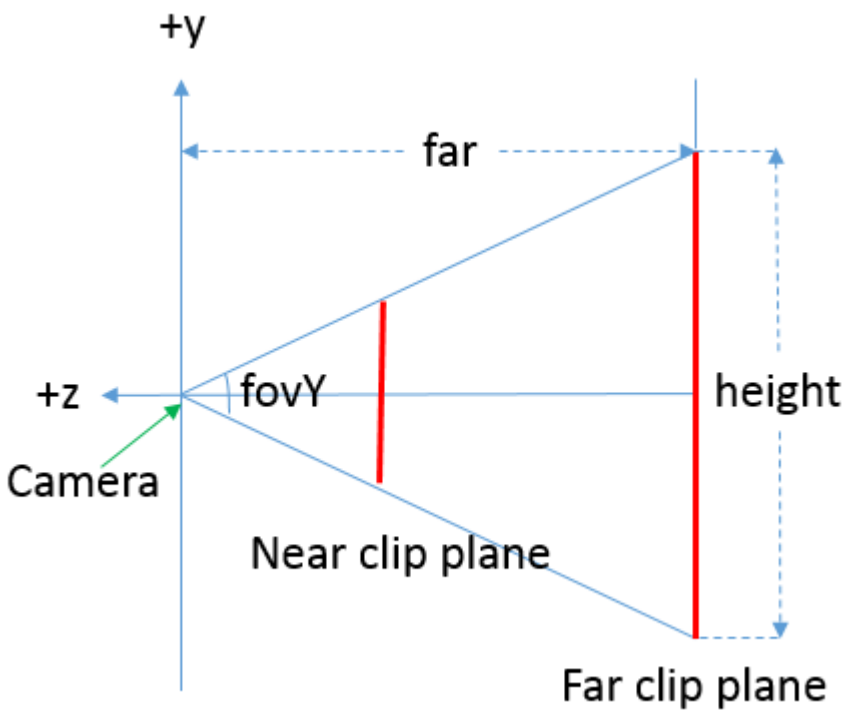
Simple SSAO——highly tweakable

- 采样:
  - 采样半径
  - 采样数量
  - 降采样（提升效率）
  - ……
- AO计算:
  - 遮挡强度
  - 遮挡偏移
  - 遮挡方向
  - 遮挡强度
  - 调制光照
  - 距离衰减
  - ……



Simple SSAO

根据uv、depth计算像素在摄像机空间的坐标：



参考

- https://en.wikipedia.org/wiki/Screen_space_ambient_occlusion
- https://www.gamedev.net/resources/_/technical/graphics-programming-and-theory/a-simple-and-practical-approach-to-ssao-r2753
- http://john-chapman-graphics.blogspot.com/2013/01/ssao-tutorial.html
- https://learnopengl.com/#!Advanced-Lighting/SSAO
- https://zhuanlan.zhihu.com/p/25038820
- http://developer.download.nvidia.com/presentations/2008/GDC/GDC08_Ambient_Occlusion.pdf

Thanks!