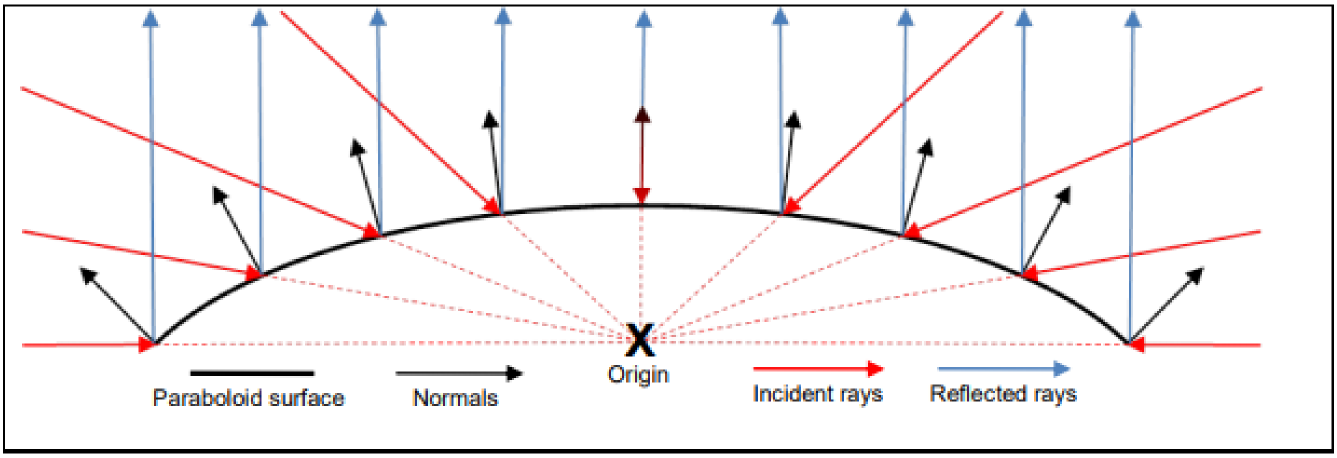


详解双抛物面环境映射


Jeffrey Z...
不正常代码研究中心—高级研究员

已关注

在计算环境反射的时候，最常见的方法就是采用 Cube Mapping。

当环境需要时候实时更新，Cube Mapping就会暴露非常明显的缺点：

立方体六个面，更新的时候需要针对六个方向分别渲染，总共绘制六次。很容易构成性能瓶颈。

还有一种常见方法， Sphere Mapping （又称作 Spherical Environment Mapping）

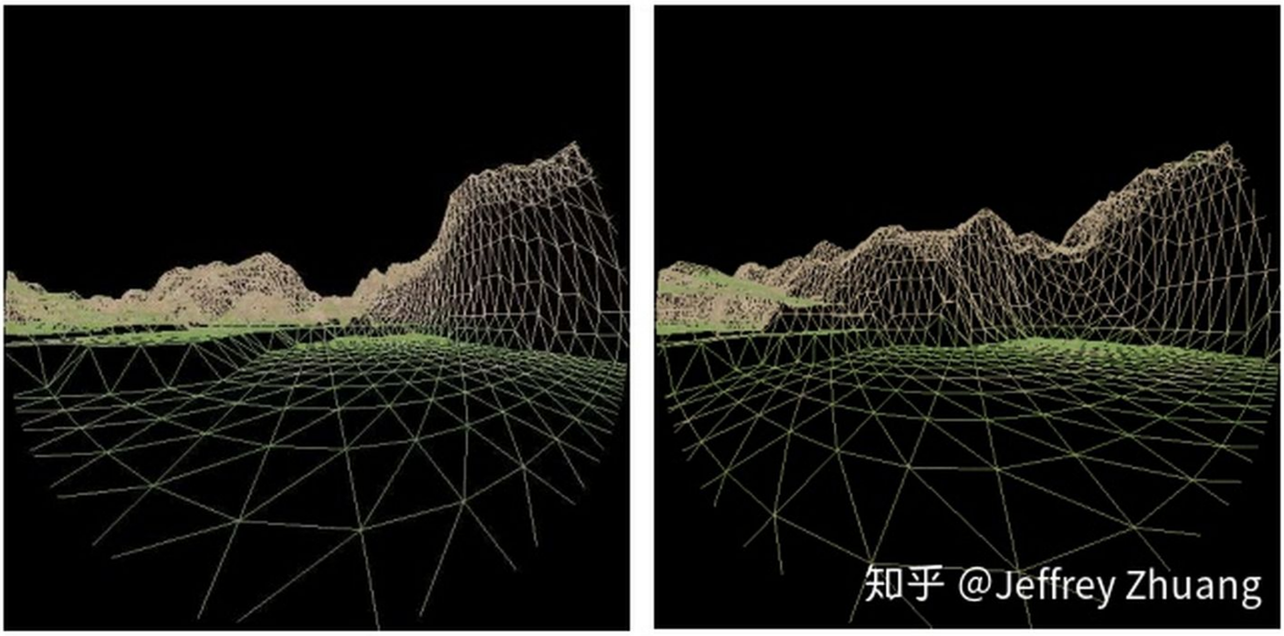
Sphere Mapping 只需要一张贴图即能表示360度全景。但是在贴图的边缘，表示的环境分辨率急剧降低。

本文要介绍的是另一种方法， 双抛物面环境映射

（Dual Paraboloid Environment Mapping）。

双抛物面环境映射

双抛物面环境将环境拆分为两个半球，分别用两张图表示。与 Cube Mapping 相比图形准确度降低，但是可以有效减少环境更新时的绘制次数。



简化后的数学原理

鉴于本人数学学渣，因此准备从最简单的情况开始逐步讲解、证明背后的数学原理。

首先要介绍一条神奇的抛物线

$$y = \frac{1}{2} - \frac{1}{2}x^2$$

如果这条抛物线可以反射光线，会发生什么呢？

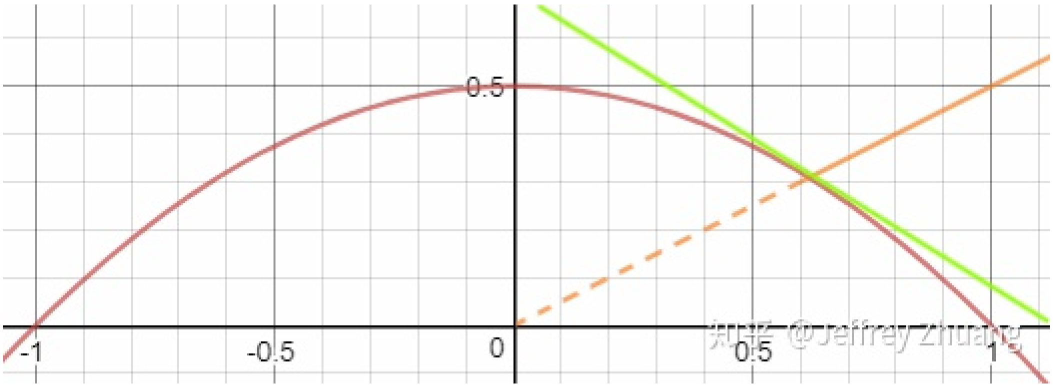
假设有一束光线（橙色）射向原点

那反射后的光线该如何表示呢?

要求反射光，首先要知道反射点上的法线。

那么如何求法线呢?

法线与切线垂直，因此**我们可以先求切线**（绿色）。



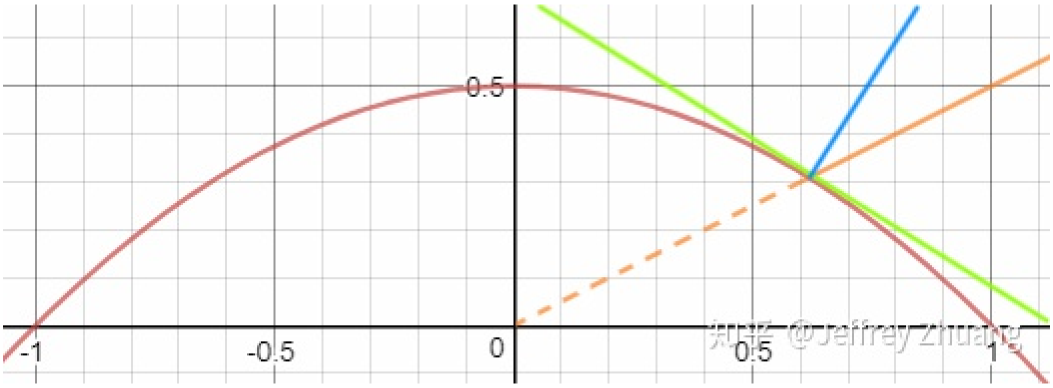
对 y 求导

$$\frac{dy}{dx} = -x \Rightarrow dy = -xdx$$

我可以得知切向向量 $[dx, -xdx]$ ，向量缩放并不影响表示的方向，xy分量分别乘 $\frac{1}{dx}$

因此**可得切向量** $[1, -x]$ 。

有了切线向量后，我们就可以求得法向量的表达式 $[x, 1]$



到此，我们先小结一下。

有入射光，法线，我们要求反射光。

入射光如何表示呢？反射点的坐标 $P(x)=[x,\frac{1}{2}-\frac{1}{2}x^2]$ ，一个点P的坐标，等同于原点到点P的向量。入射光射向原点，因此取反。

我们得到**入射向量** $[-x,-\frac{1}{2}+\frac{1}{2}x^2]$

同时我们还有**法向量** $[x,1]$

反射向量如何求呢?

$R=I-2(N\cdot I)N$

其中 I 入射光， N 法线， R 反射光

这里要求，入射光、法线都是单位向量。

因此先Normalize这两个向量。

入射光向量的模为

$$\sqrt{(-x)^2+(\frac{-1+x^2}{2})^2}$$

$$=\frac{x^2+1}{2}$$

Normalize 入射向量

$$\frac{[-x,-\frac{1}{2}+\frac{1}{2}x^2]}{\frac{x^2+1}{2}}$$

$$=[\frac{-2x}{x^2+1},\quad \frac{x^2-1}{x^2+1}]$$

法向量的模为

$$\sqrt{(x)^2+1^2}$$

$$=\sqrt{x^2+1}$$

Normalize 法向量

$$\frac{[x,1]}{\sqrt{x^2+1}} \Rightarrow [\frac{x}{\sqrt{x^2+1}},\frac{1}{\sqrt{x^2+1}}]$$

经过计算我们**获得了单位化后的** I **和** N

$$I=[\frac{-2x}{x^2+1},\quad \frac{x^2-1}{x^2+1}]$$

$$N=[\frac{x}{\sqrt{x^2+1}},\frac{1}{\sqrt{x^2+1}}]$$

反射公式 $R=I-2(N\cdot I)N$ 中

有一个 I 与 N 的点积，这里先来求这个点积

$$I \cdot N = \frac{(-2x)x}{(x^2+1)\sqrt{x^2+1}} + \frac{x^2-1}{(x^2+1)\sqrt{x^2+1}}$$

$$= -\frac{1}{\sqrt{x^2+1}}$$

$$2(N \cdot I)N = 2(-\frac{1}{\sqrt{x^2+1}})[\frac{x}{\sqrt{x^2+1}}, \frac{1}{\sqrt{x^2+1}}]$$

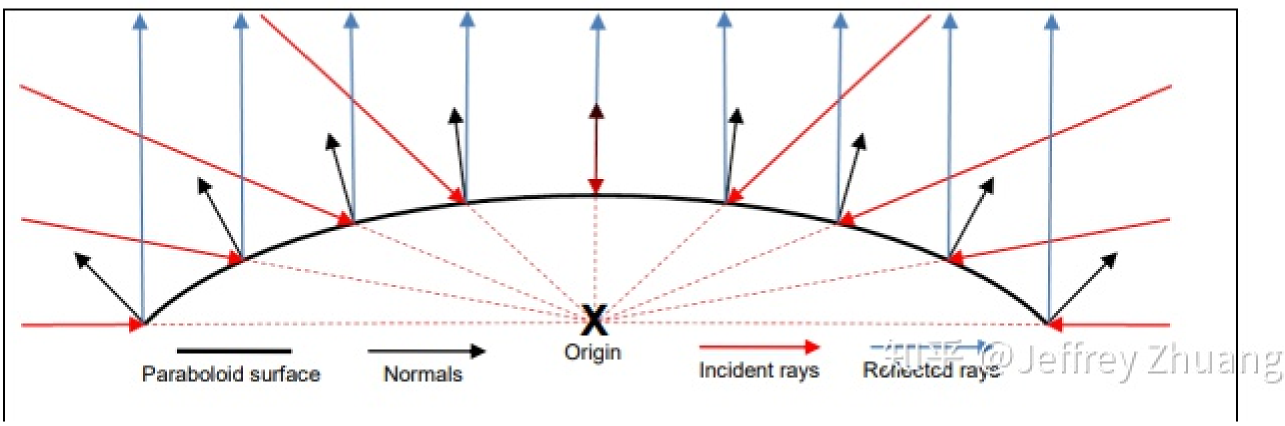
$$= [\frac{-2x}{x^2+1}, \frac{-2}{x^2+1}]$$

接下来是见证奇迹的时刻

$$R = I - 2(N \cdot I)N$$

$$= [\frac{-2x}{x^2+1}, \quad \frac{x^2-1}{x^2+1}] - [\frac{-2x}{x^2+1}, \frac{-2}{x^2+1}]$$

$$= [0, 1]$$



对于抛物线 $y = \frac{1}{2} - \frac{1}{2}x^2$ **，** $x > -1$ **且** $x < 1$**，由y>0的任意点指向原点的向量**

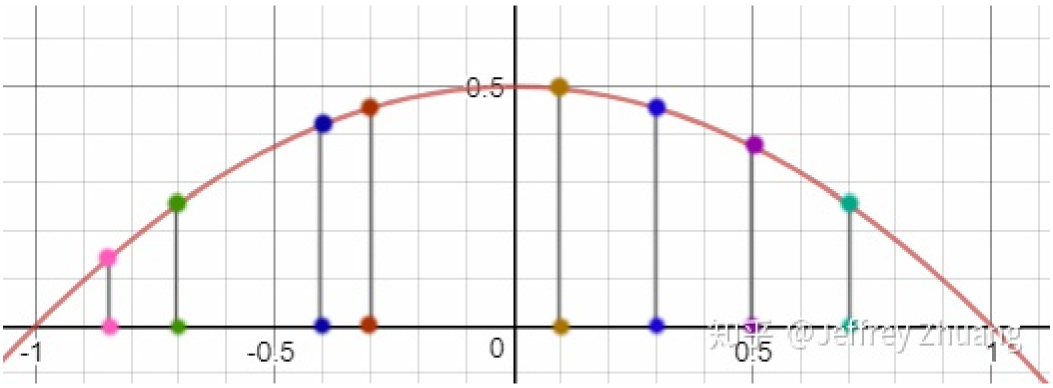
经过该抛物线反射后，在我们的二维坐标系里，反射向量恒指向y轴正方向。

二维情况下的数据的存取

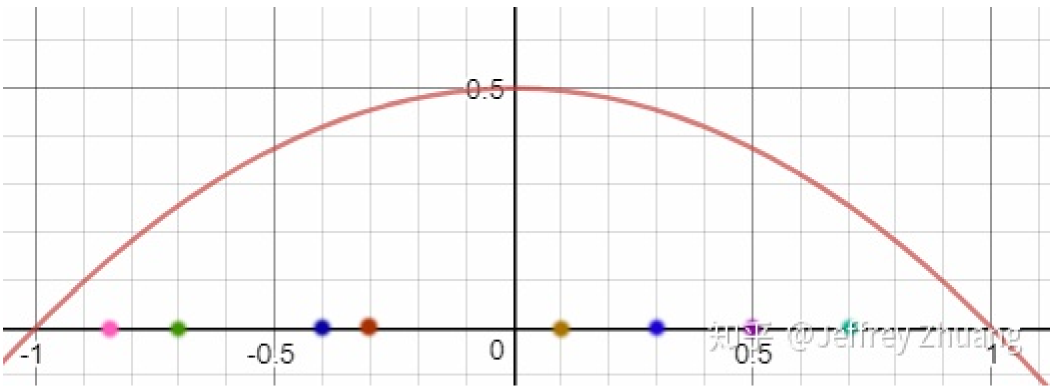
讲一下二维情况下，数据的存取、映射方式。

下图中列举了y > 0空间中某些方向上的颜色。

这些颜色投影到x轴上

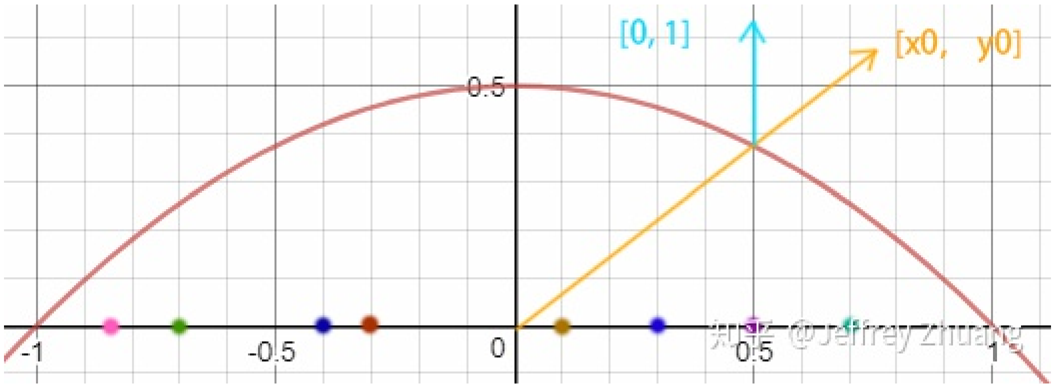


最终存储的形式如下图



如假，我们想知道方向 $[x_0, y_0]$ 上的环境颜色该如何获得呢?

首先我们已知单位向量 $V_0=[x_0,y_0]$ ，如果光线从 $-V_0$ 射向原点，由上一节的推导，我们可知，反射向量恒为 $V_1=[0,1]$



法线可以由 V_0 、V_1 求得

$$N=V_0+V_1$$

$$=[x_0,y_0+1]$$

此处的 N 并不是单位向量，但是由上一节中的内容：

我们得到**入射向量** $[-x,-\frac{1}{2}+\frac{1}{2}x^2]$

同时我们还有**法向量** $[x,1]$

知乎 @Jeffrey Zhuang

可知，法向量具有 $[x,1]$ 的形式。于是我们将此处的 N 缩放到形如 $[x,1]$ 。

$$[\frac{x_0}{y_0+1},1]$$

x **为** $\frac{x_0}{y_0+1}$ **处存储的颜色值即** $V_0=[x_0,y_0]$ **方向上对应的颜色。**

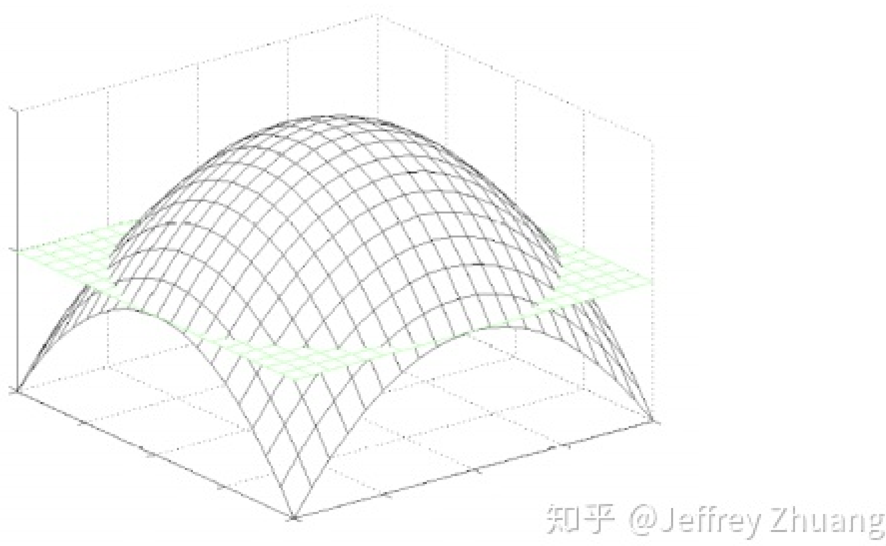
Tips: *此处补充说明下缩放到[x,1]的原因，已知抛物线的坐标满足* $P(x)=[x,\frac{1}{2}-\frac{1}{2}x^2]$ *，而P点对应的法线是[x,1]，由此反推，满足法线[x,1]的抛物线上P点坐标满足* $P(x)=[x,\frac{1}{2}-\frac{1}{2}x^2]$ *。再投影点P得到存取坐标。*

二维的情况讲完了，下面要开始进入三维世界，我们先约定一下坐标系，**这里的讲述采用左手坐标系**，Direct3D、Unity3D 使用的坐标系。

如果 X轴指向屏幕右侧， Y轴指向屏幕上方 则 Z轴指向屏幕内，同视线方向。

抛物面背后的数学原理

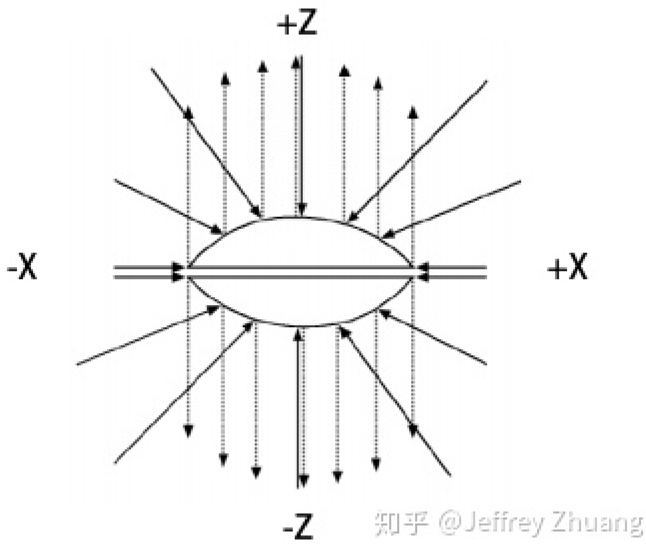
由二维世界进入三维世界，先前的抛物线就变成了抛物面。



$$z = f(x, y) = \frac{1}{2} - \frac{1}{2}(x^2 + y^2)\ ,\quad x^2 + y^2 < 1$$

双抛物面会将环境分成两个方向绘制，我们将环境拆分成 +Z轴方向和-Z轴方向。

从顶视图看如下：



我们以+Z轴为例讲解， -Z轴的原理相同。

先给结论：

对于抛物面 $f(x,y)=\frac{1}{2}-\frac{1}{2}(x^2+y^2)$ **，** $x^2+y^2<1$

由z>0的任意点指向原点的向量，经过该抛物面的反射后，反射方向恒为 $[0,0,1]$

抛物面映射的数据存取

原理与先前讲过的二维情况类似。这里会讲的比较跳跃。

先说如何生成贴图。

先说数据的存储，假设在+Z轴范围内，有一顶点 $P_0=(x_0,y_0,z_0)$ 。

首先将 V_0 看作是由原点开始，指向 P_0 的向量。

再有反射向量 $V_1=[0,0,1]$

$$N_0=V_0+V_1$$

$$=[x_0,y_0,z_0+1]$$

根据上一篇章

通过求 T_x 与 T_y 的叉积可得法向量，

即 $N_p=T_x\times T_y$

$$=[0\cdot(-y)-1\cdot(-x),-x\cdot 0-1\cdot(-y),1\cdot 1-0\cdot 0]$$

$$=[x,y,1]$$

知乎 @Jeffrey Zhuang

法线可以形如 $[x,y,1]$

于是将 N_0 表示为 $[\frac{x_0}{z_0+1},\frac{y_0}{z_0+1},1]$

$$x=\frac{x_0}{z_0+1}$$

$$y=\frac{y_0}{z_0+1}$$

x 、 y 的输出范围都为 $[-1,1]$ ，刚好与剪裁坐标系一致，因此不需要做额外调整。

环境的映射的贴图生成后，该如何读取呢?

开始的步骤和贴图生成是一样的，但是读取的时候需要使用 $[0,1]$ 范围的UV坐标，所以要再做一次映射。

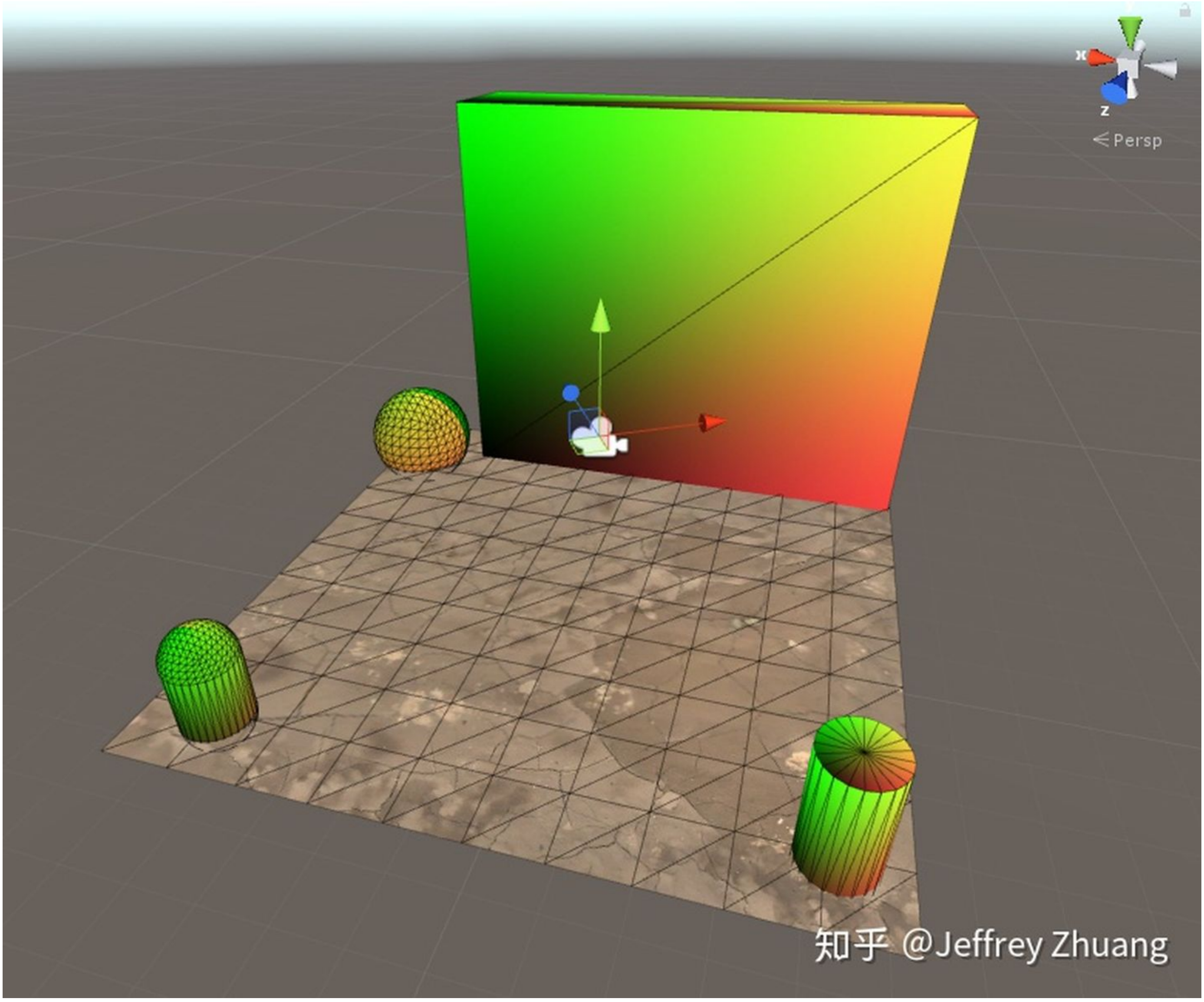
这里要注意下DX和OpenGL的UV坐标系不一致，DX的UV原点在贴图的左上角， $y_{DX}=1-y$

以OpenGL的UV坐标系为例，采样的 $u=\dfrac{x_0}{z_0+1}\cdot 0.5+0.5$ ，$v=\dfrac{y_0}{z_0+1}\cdot 0.5+0.5$

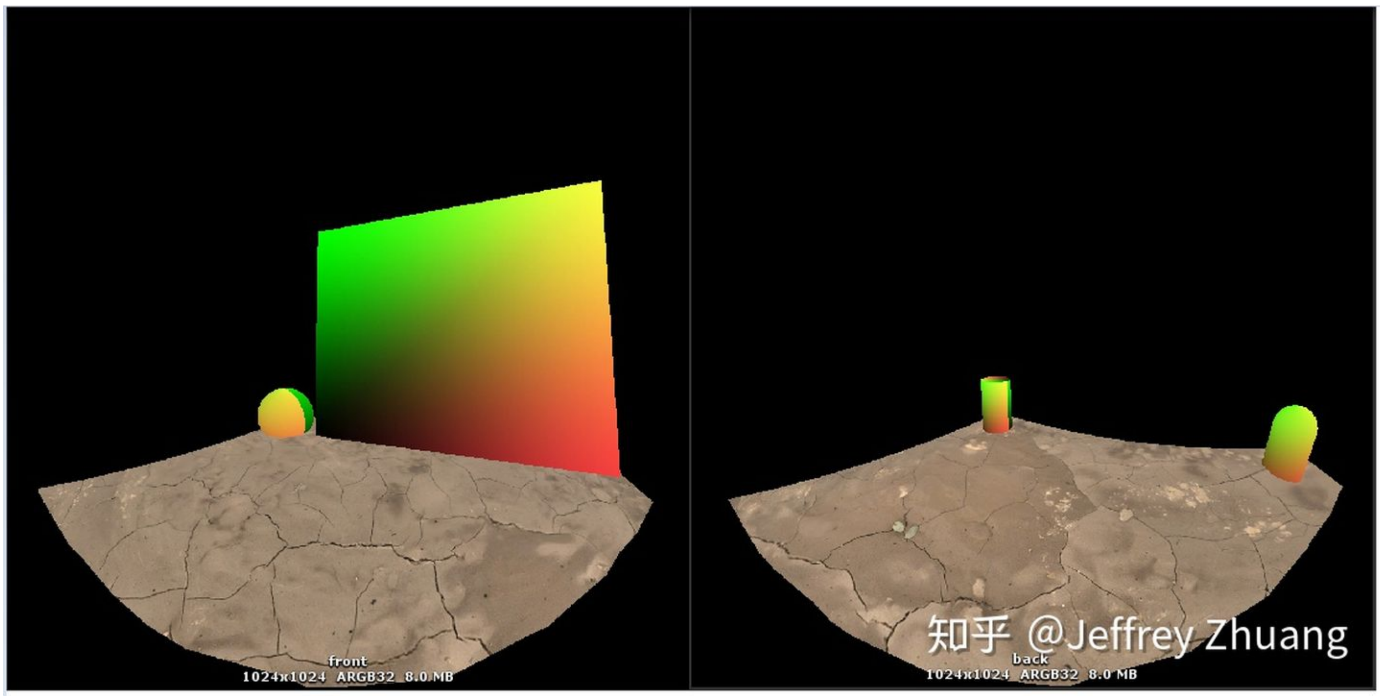
基本原理到此已经讲完了，接下来讲一下 Dual Paraboloid Environment Mapping 在 **Unity3D中实现的一些想法和坑点**。

双抛物面环境反射的U3D中的实现

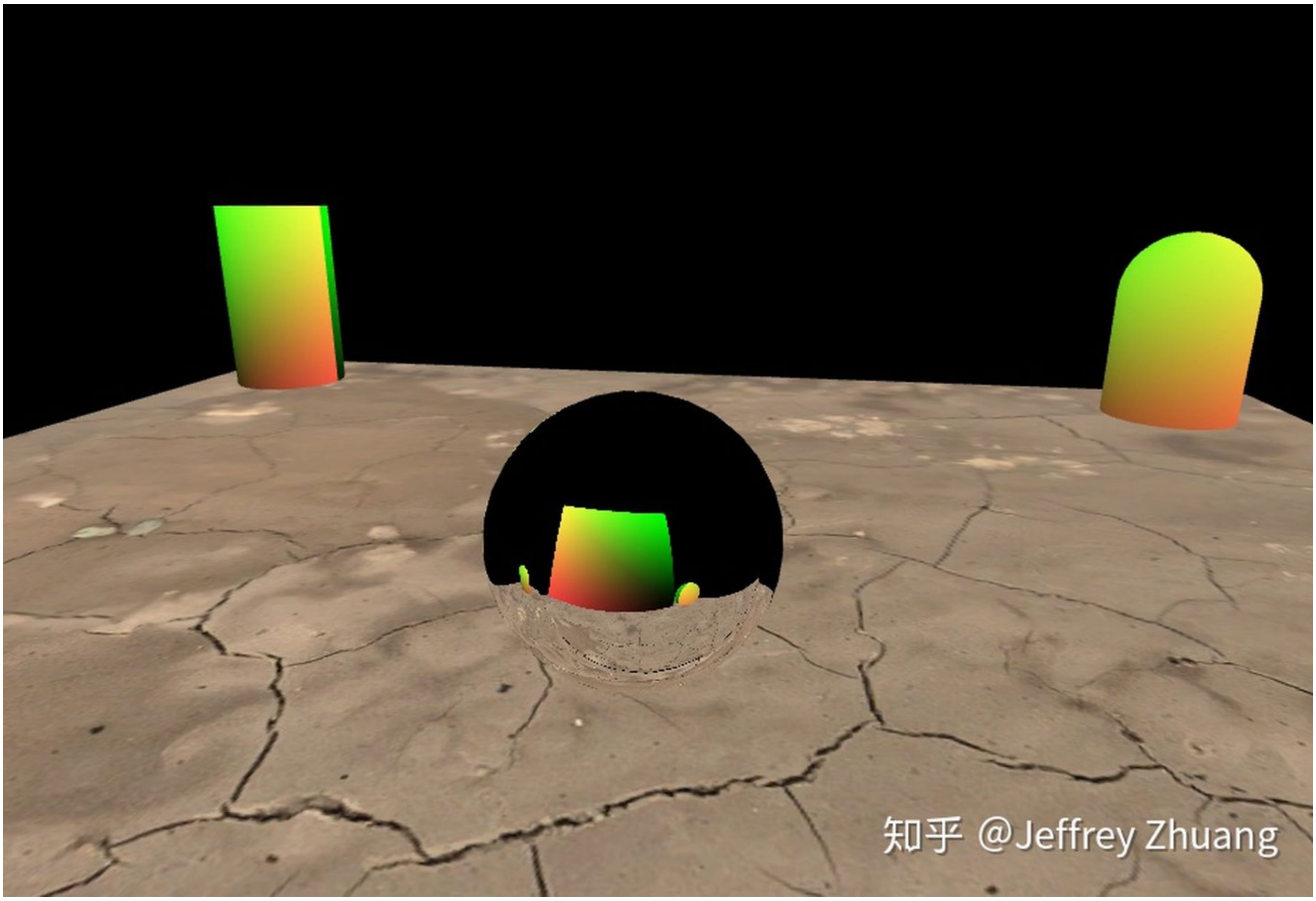
如下的场景



烘焙出来的两张双抛物面环境图如下



拿一个Sphere附上双抛物环境反射的Shader后结果如图



双抛物面环境反射的原理上非常简单，但要在实际项目中使用并不容易。

首先，**生成环境图的时候，用于渲染的Shader要在顶点着色器中要做针对抛物面计算剪裁坐标**。不同物体的着色方式不同，通过Replacement Shader的机制不可行。并且引擎没有一种能替换UnityObjectToClipPos实现的机制。

因此，要在实际项目中使用，会有很大的限制。

如果你的项目中，所有Shader有通用的顶点着色器Shader；或者所有Shader都调用统一的接口做Local到Clip Space的变换，那么也可以很容易的处理, 你可以通过multi_compile来切换普通的变换还是双抛物面变换。如果不同的Shader

都有自己的Vertex Shader那维护起来并不轻松，如果还有混用引擎内置Shader或者插件，那么基本就可以放弃了。

此外，**Unity的视锥剔除还会干扰环境图的生成（视锥剔除的范围和最终的剪裁坐标系不一致），比较简单的方案是改成正交摄像机，近裁面从0，正交的Size调大。**

Unity的ModelView变换后的坐标系是右手坐标系，在做抛物面映射的时候需要翻转Z。

Unity的Projection矩阵平台差异很大，还有Reversed Z捣乱，正确的Clip Space Z自己计算会非常繁琐。另外，在DirectX API下，渲染到屏幕和渲染到纹理的情况，Y值要考虑是否翻转的问题。

综上所述，下面给出一些代码参考片段。

扩展知识

- 采用双抛物面环境映射还可以用来生成点光源的Shadowmap
- 可以通过坐标变换把两张图生成到一张长方形的图上
- 可以输出到Cubemap的两个面，缺点是比较浪费空间
- 可以将正方形按一个对角线拆分，放对角线两侧（某热门手游的做法）
- 抛物线的光学性质：在焦点上的点光源发出的光线，经抛物线反射后平行于抛物线的对称轴。典型应用如手电筒。

参考

DualParaboloidMappingInTheVertexShader

Dual-Paraboloid Reflections

Cubic Environment Mapping (Direct3D 9)

Spherical Environment Mapping (Direct3D 9)

Dual Paraboloid Environment Mapping

dual-paraboloid-reflections

Dual-Paraboloid Shadow Maps
dual-paraboloid-variance-shadow-mapping
View-independent_environment_maps

dual-paraboloid_shadow_mapping

dual-paraboloid-shadow-mapping

编辑于 2019-09-11

PaintDream 2018-08-02

抛物线焦点发出的光线反射后为平行光，文章中那个抛物线的焦点正好是原点

抛物线焦点发出的光线反射后为平行光，文章中那个抛物线的焦点正好是原点。

1

yaobrother 2018-08-25

Cube Mapping 可以一次绘制立方体的6个面，参考DirectX SDK中的CudeMapGS

赞