太极图形课
光线追踪作业分享

乘风龙王·小威

2021年12月

[Taichi] version 0.8.5, llvm 10.0.0, commit 45c6ad48, win, python 3.9.9

作业链接：https://forum.taichi.graphics/t/topic/2063
Github链接：cflw/taichi_demo/光线追踪/照镜子.py

目录

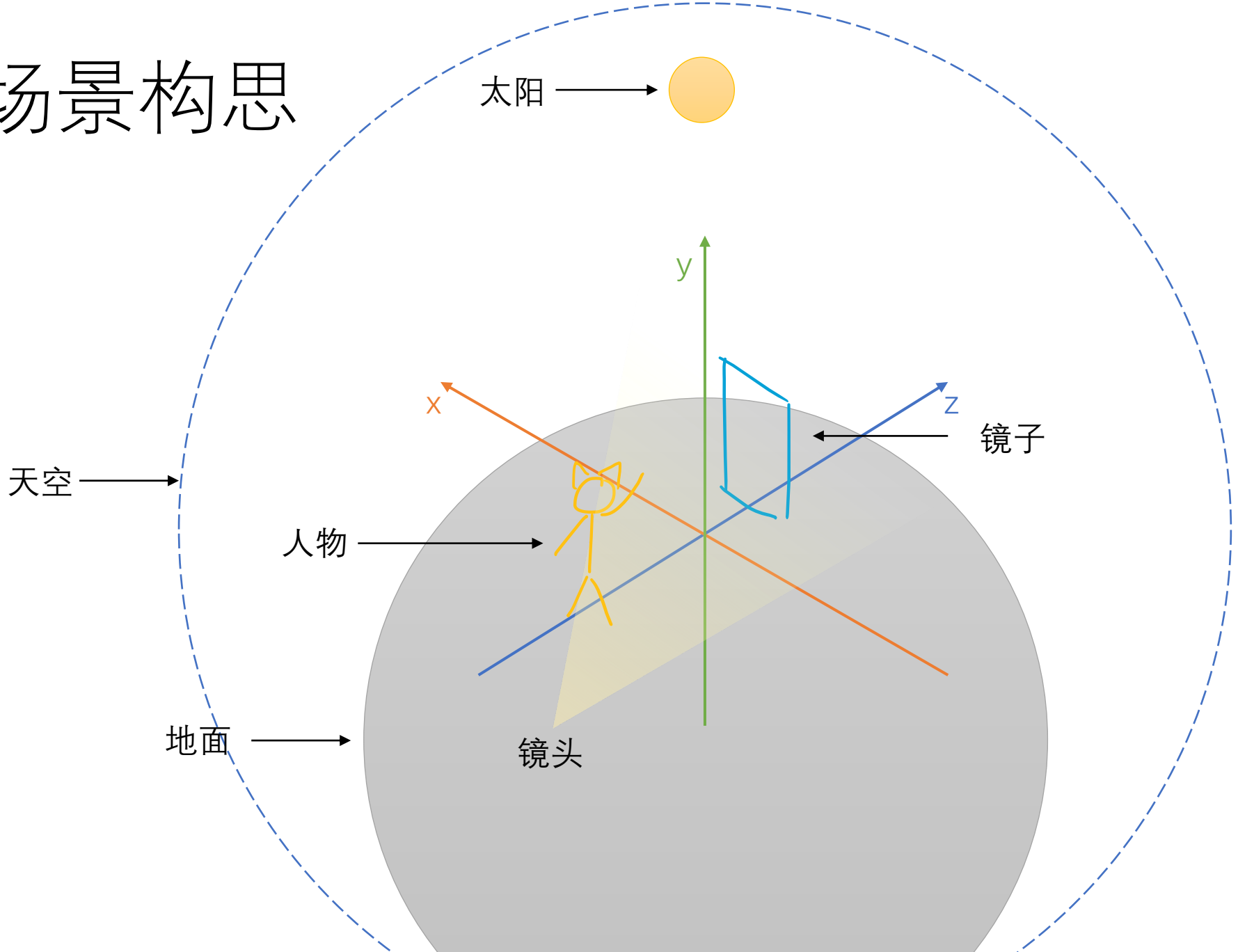
太极图形课：

- 光线追踪原理、算法
- 光线追踪代码
  - 场景
    - 物体列表
    - 碰撞判定
  - 相机
    - 发射光线
  - 渲染
    - 光线反射、折射、染色
  - 球体
    - 碰撞判定

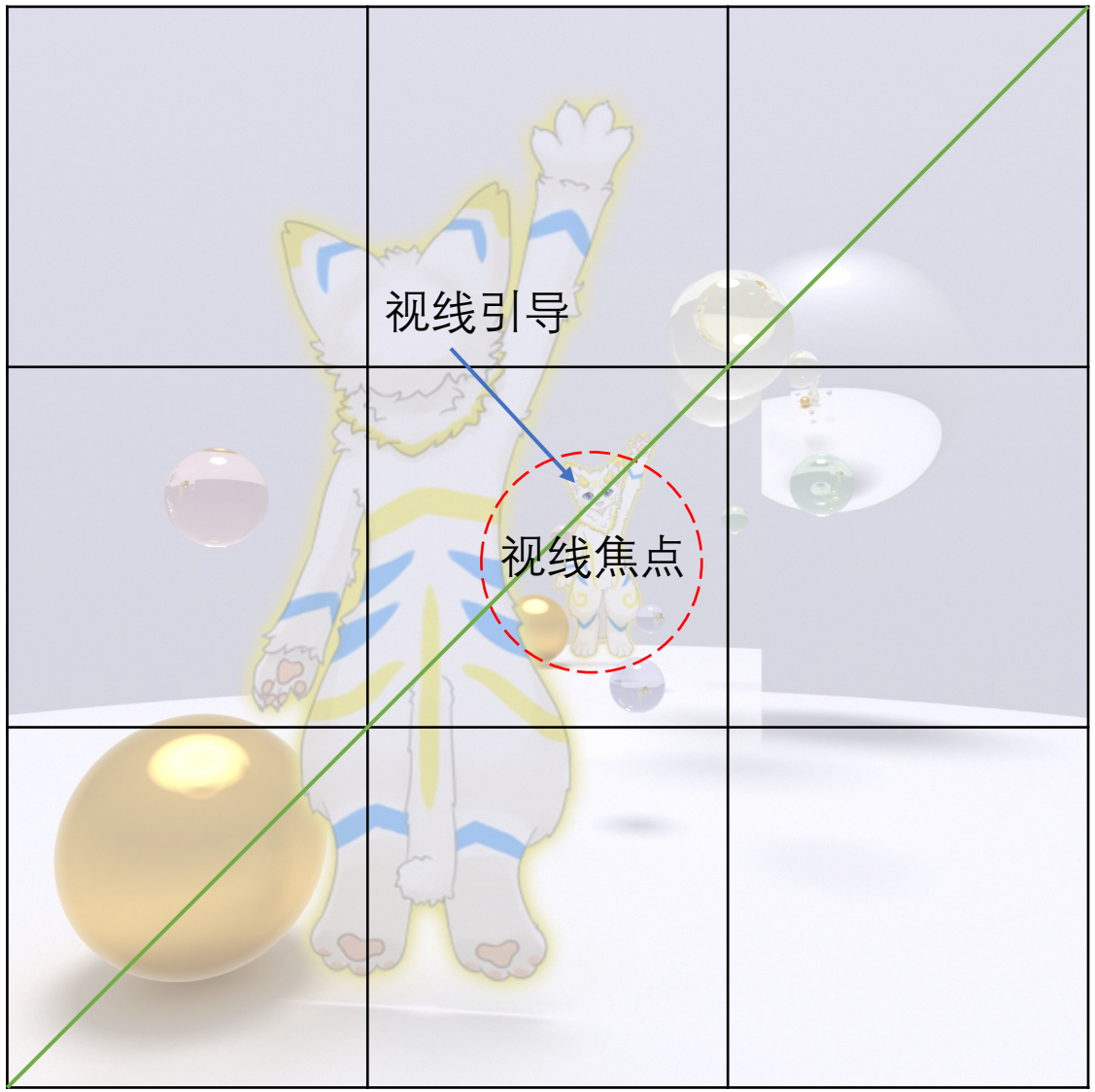
我：

- 美术
  - 场景构图
  - 角色双视图
- 光线追踪代码
  - 矩形
    - 碰撞判定
  - 图片
    - 载入图片
    - 纹理过滤
    - 碰撞判定

场景构思

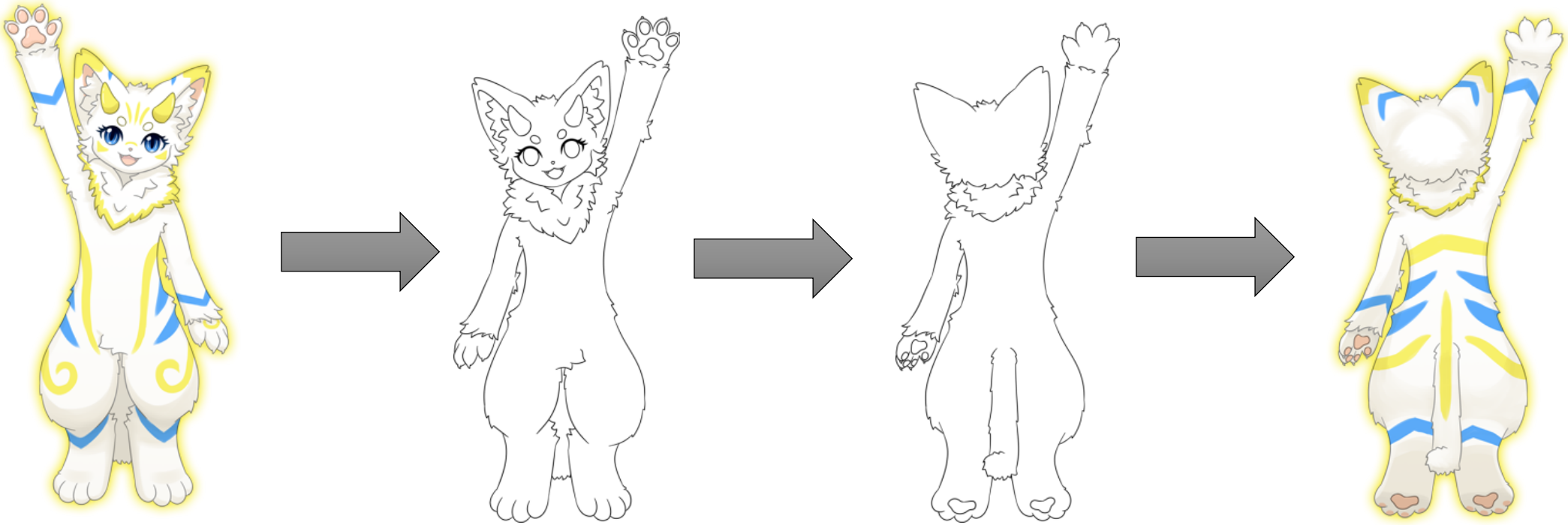


构图



双视图

因为是照镜子，要求姿势一样，外轮廓水平对称



先画正面

复制线稿图层，
水平翻转线稿

修改内部线条

上色

不宜采用带有强烈透视效果的图，这种图画对称双视图时容易出现透视错误。

矩形类

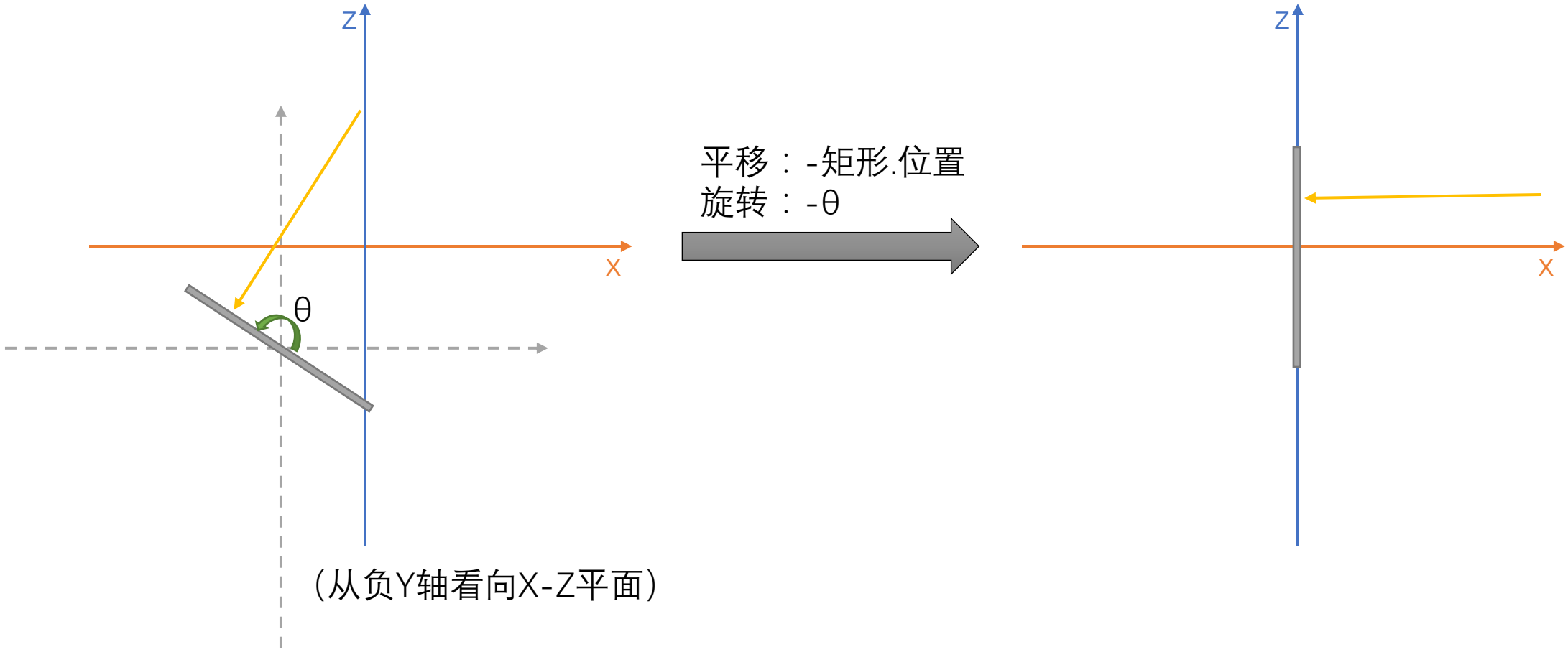
包含的变量：

- 位置：矩形中心点的位置
- 尺寸：矩形的尺寸
- 旋转：绕y轴旋转的角度
- 颜色：矩形的颜色
- 材质：矩形的材质

```
@ti.data_oriented
class C矩形:
    def __init__(self, a位置, a尺寸, a旋转, a颜色, a材质):
        self.m位置 = a位置
        self.m半尺寸 = a尺寸 * 0.5
        self.m旋转 = a旋转
        self.m颜色 = a颜色
        self.m材质 = a材质
    @ti.func
    def hit(self, a光线, a最小值, a最大值):
        … #碰撞判定
```

光线和矩形碰撞判定

1、平移、旋转参考系，计算出光线相对矩形的位置和方向



光线和矩形碰撞判定

1、

```
#平移
v相对位置 = a光线.m位置 - self.m位置

#旋转
c = ti.cos(-self.m旋转)
s = ti.sin(-self.m旋转)
v旋转矩阵 = ti.Matrix([[c, 0.0, s], [0.0, 1.0, 0.0], [-s, 0.0, c]])

v相对位置 = v旋转矩阵 @ v相对位置
v相对方向 = v旋转矩阵 @ a光线.m方向
```

绕y轴旋转矩阵，旋转方向和右手螺旋相反

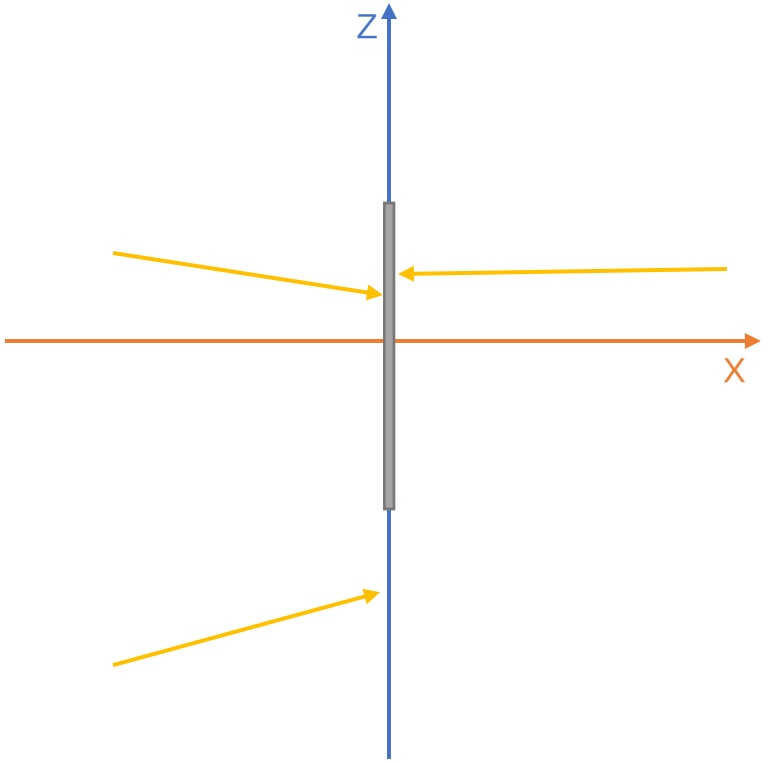
$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

光线和矩形碰撞判定

2、计算光线到Y-Z平面（x=0）时的t值

$$t = -\frac{位置_x}{方向_x}$$

```
t = -v相对位置.x / v相对方向.x
```



（从负Y轴看向X-Z平面）

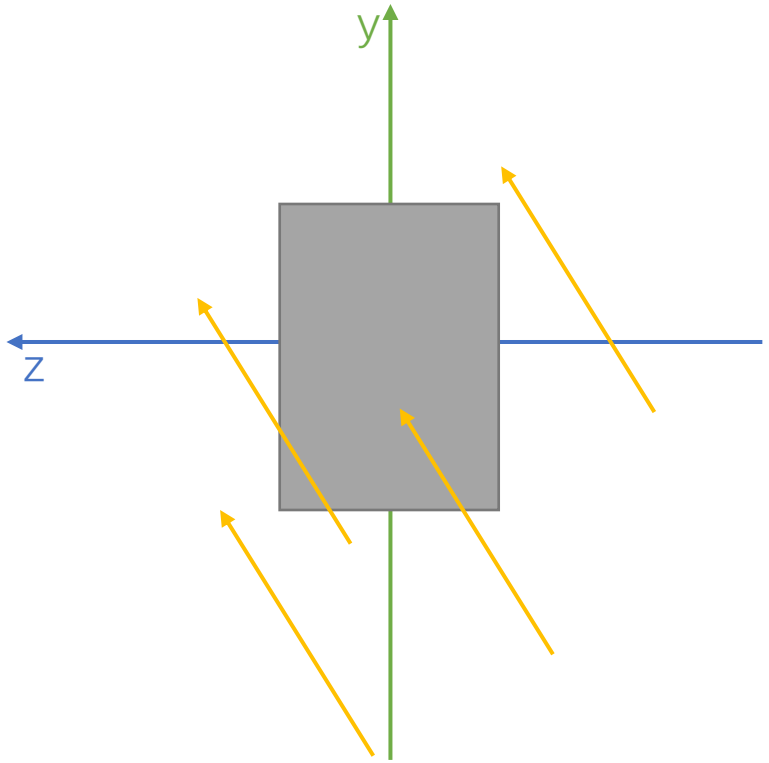
光线和矩形碰撞判定

3、计算光线到Y-Z平面（x=0）的交点坐标

交点 = 位置 + 方向 · t

当相对交点坐标在矩形范围内，相交

```
v相对交点 = v相对位置 + v相对方向 * t
if abs(v相对交点.z) < self.m半尺寸.x and abs(v相对交点.y) < self.m半尺寸.y:
    … #碰撞
else:
    pass #不相交
```



（从正X轴看向Y-Z平面）

光线和矩形碰撞处理

和球体类似

```
v相对位置 = …
v相对方向 = …
t = -v相对位置.x / v相对方向.x
v相对交点 = v相对位置 + v相对方向 * t
#准备返回变量
v碰撞 = False
v交点 = t向量3(0.0, 0.0, 0.0)
v交点法线 = t向量3(0.0, 0.0, 0.0)
v前面 = False
#判断碰撞
if abs(v相对交点.z) < self.m半尺寸.x and abs(v相对交点.y) < self.m半尺寸.y:
    v碰撞 = True
    v交点 = a光线.at(t)
    v交点法线 = t向量3(c, 0.0, s)
    if v相对位置.x >= 0: #判断前面
        v前面 = True
    else:
        v交点法线 = -v交点法线
return v碰撞, t, v交点, v交点法线, v前面, self.m颜色, self.m材质
```

图片类

包含的变量：

- 位置：矩形中心点的位置
- 尺寸：矩形的尺寸
- 旋转：也是绕y轴旋转
- 纹理：一张图片，从文件读取

```
@ti.data_oriented
class C图片:
    def __init__(self, a位置, a尺寸, a旋转, a路径):
        self.m位置 = a位置
        self.m半尺寸 = a尺寸 * 0.5
        self.m旋转 = a旋转
        … #载入图片
    @ti.func
    def hit(self, a光线, a最小值, a最大值):
        … #碰撞判定
```

载入图片

使用pillow库载入图片并放入太极场

1、载入文件

```
from PIL import Image
v文件 = Image.open(“图片.png”)
```

2、转换成numpy数组

```
import numpy as np
v数据 = np.array(v文件.getdata(), dtype=float)
```

数组形状=(宽*高，通道数)

3、颜色范围从[0,255]转换成[0,1]（太极的颜色范围）

```
v数据 /= 255.0
```

载入图片

4、填充数据

```
v填充 = np.zeros((v文件.width, v文件.height, 4), dtype = float)
for x in range(v文件.width):
    for y in range(v文件.height):
        v填充[x, y] = v数据[y * v文件.width + x]
```

5、放入太极场
定义一个场，名为“纹理”，数据来自刚才的填充数据

```
v纹理 = ti.Vector.field(4, dtype = float, shape = (v文件.width, v文件.height))
v纹理.from_numpy(v填充)
```

6、结束

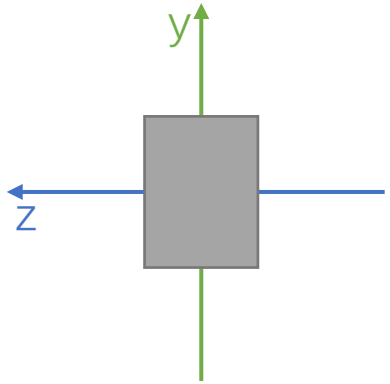
```
v文件.close()
```

载入图片代码

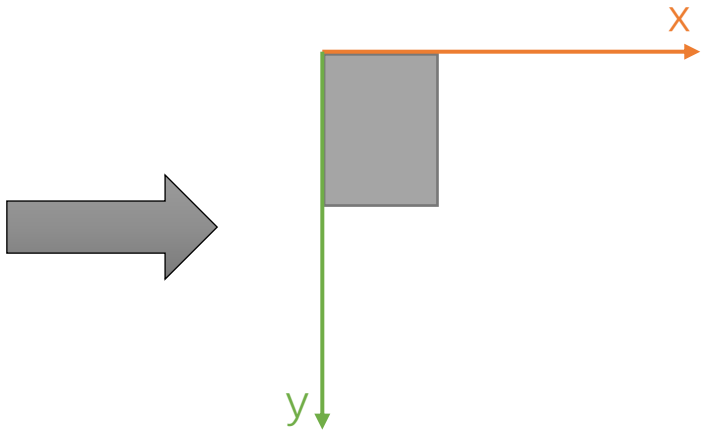
```
#打开文件
v文件 = Image.open(a路径)

#判断颜色通道，在这个照镜子场景中，只能使用RGBA
if v文件.mode != "RGBA":
    raise ValueError("只能载入带透明通道的图片")

#转换成numpy数组
v数据 = np.array(v文件.getdata(), dtype = float)
v数据 /= 255.0

#填充数据
v填充 = np.zeros((v文件.width, v文件.height, 4), dtype = float)
for x in range(v文件.width):
    for y in range(v文件.height):
        v填充[x, y] = v数据[y * v文件.width + x]

#填入太极场
self.m纹理 = ti.Vector.field(4, dtype = float, shape = (v文件.width, v文件.height))
self.m纹理.from_numpy(v填充)

#结束
v文件.close()
```

纹理采样

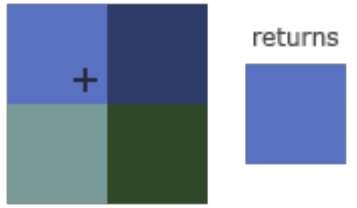
根据纹理坐标获取指定位置像素颜色



光线相交于矩形的y-z平面



纹理坐标



根据纹理坐标获取颜色

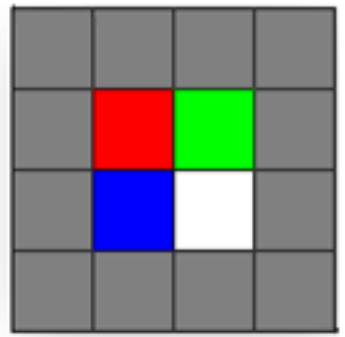
在光线与矩形碰撞判定的基础上，加上根据交点位置计算纹理坐标的代码

```
v相对交点 = …
v纹理尺寸x = float(self.m纹理.shape[0])
v纹理尺寸y = float(self.m纹理.shape[1])
v纹理坐标x = v纹理尺寸x - (v相对交点.z / self.m半尺寸.x * 0.5 + 0.5) * v纹理尺寸x - 1
v纹理坐标y = v纹理尺寸y - (v相对交点.y / self.m半尺寸.y * 0.5 + 0.5) * v纹理尺寸y - 1
v颜色 = self.m纹理[int(v纹理坐标x), int(v纹理坐标y)] #纹理采样
```

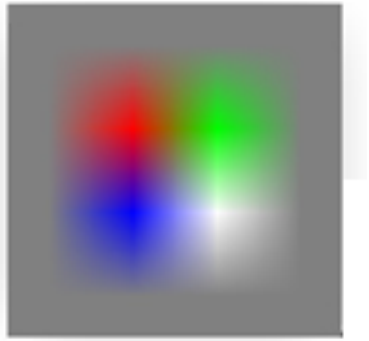
纹理过滤

当纹理放大后，如果只采集最近的一个像素，最后渲染出来的图是一堆马赛克。

为了得到平滑的图案，需要对纹理进行过滤。



最近点**采样**



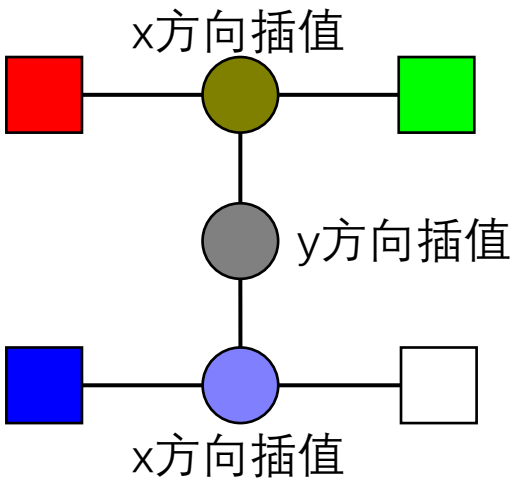
双线性**过滤**

双线性纹理过滤

对纹理坐标四周的像素进行采样，插值计算得到最终的颜色。

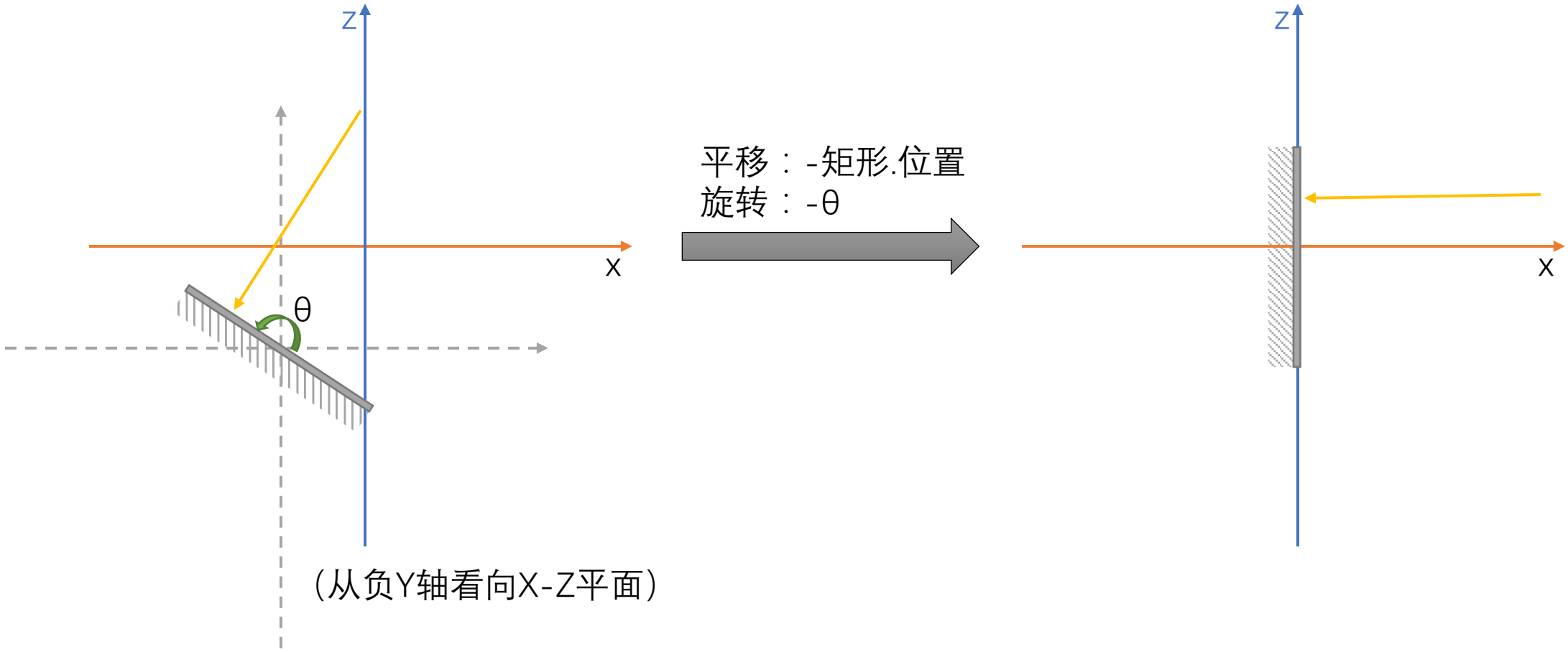
```
v纹理坐标x = …
v纹理坐标y = …
v纹理坐标x0 = int(ti.floor(v纹理坐标x))
v纹理坐标x1 = int(ti.ceil(v纹理坐标x))
v纹理坐标y0 = int(ti.floor(v纹理坐标y))
v纹理坐标y1 = int(ti.ceil(v纹理坐标y))
v纹理坐标x_ = v纹理坐标x - v纹理坐标x0
v颜色0 = lerp(self.m纹理[v纹理坐标x0, v纹理坐标y0], self.m纹理[v纹理坐标x1, v纹理坐标y0], v纹理坐标x_)
v颜色1 = lerp(self.m纹理[v纹理坐标x0, v纹理坐标y1], self.m纹理[v纹理坐标x1, v纹理坐标y1], v纹理坐标x_)
v颜色2 = lerp(v颜色0, v颜色1, v纹理坐标y - v纹理坐标y0)  #双线性纹理过滤
```

```
@ti.func
def lerp(a, b, t): #插值
    return a + (b - a) * t
```



光线和图片的碰撞判定

和矩形相同的地方：把光线转换到相对图片的坐标系

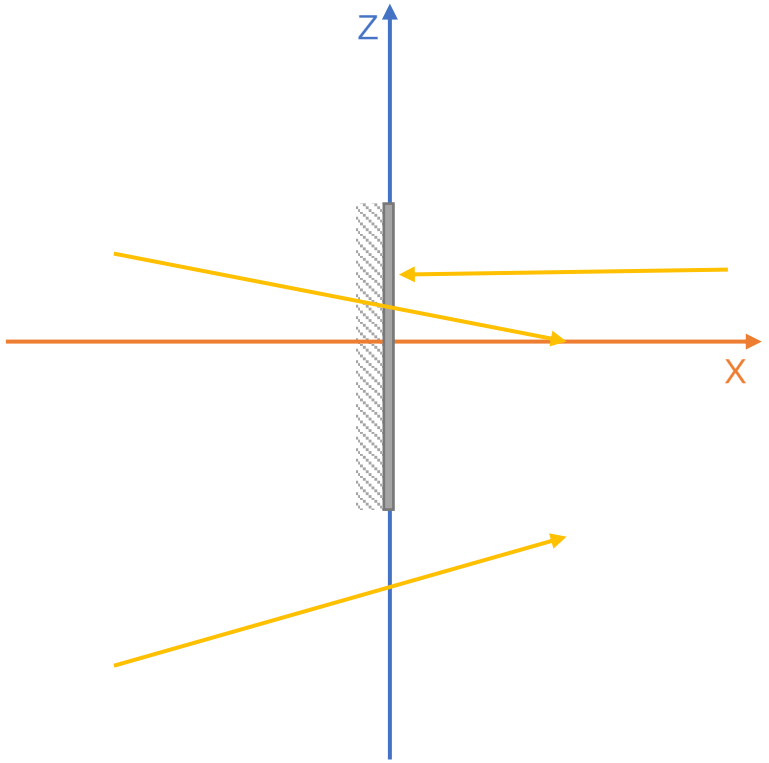


光线和图片的碰撞判定

和矩形不同的地方：

当光线位于图片背面时，不判断是否相交，直接穿过

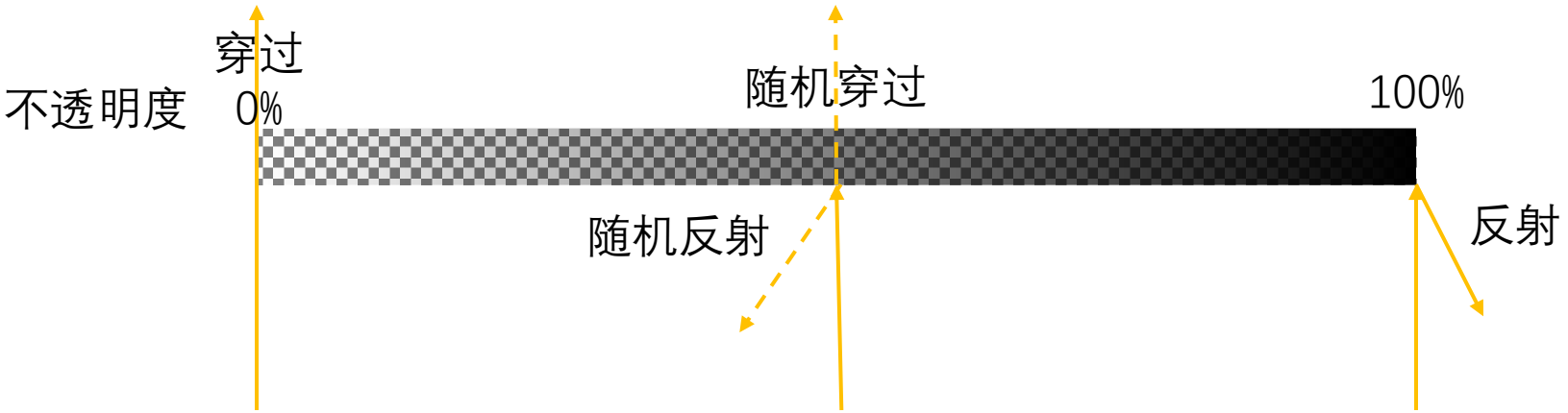
```
if v相对位置.x <= 0.0:
    pass #穿过
else:
    … #碰撞
```



光线和图片的碰撞判定

当交点位于图片的透明像素，根据不透明度决定是否穿过

```
if ti.random() <= v颜色.w:
    … #碰撞
else:
    pass #穿过
```



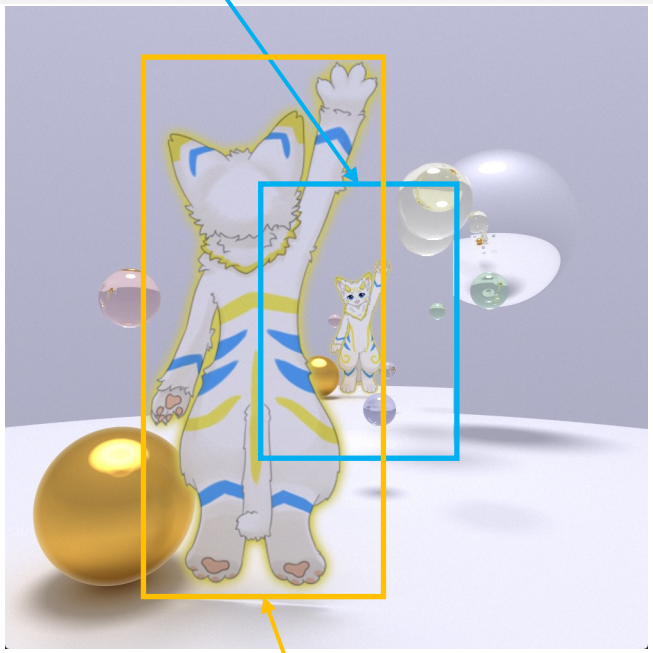
光线和图片的碰撞处理

只有当前面的碰撞条件都成立，才算真正的碰撞。

- 交点&法线：和矩形一样处理
- 正面：总是为真（背面不计算，直接穿过）
- 颜色：等于纹理颜色
- 材质：总是漫反射

最终布置

```
v场景 = C场景()
v场景.add(C球体(t向量3(0.0, 50.0, 0.0), 10, t向量3(10, 10, 10), E材质.e光源)) #光
v场景.add(C球体(t向量3(0.0, 0.0, 0.0), 100, t向量3(.9, .9, .92), E材质.e漫反射))          #天空
v场景.add(C矩形(t向量3(0.0, 1.0, 1.0), t向量2(2, 3), 半PI, t向量3(.99, .99, .99), E材质.e镜面反射))  #镜子
v场景.add(C球体(t向量3(0.0, -50.0, -1.0), 50, t向量3(0.8, 0.8, 0.8), E材质.e漫反射))      #地面
```



图片正反面位置重叠

```
v场景.add(C图片(t向量3(0.0, 1.0, -2.0), t向量2(c目标图片宽度, c目标图片高度), -半PI, "光线追踪/正面.png"))
v场景.add(C图片(t向量3(0.0, 1.0, -2.0), t向量2(c目标图片宽度, c目标图片高度), 半PI, "光线追踪/背面.png"))
```

编写物体类的思路

```
@ti.data_oriented
class C物体:
    def __init__(self, …):
        … #初始化物体属性
    @ti.func
    def hit(self, a光线, a最小值, a最大值):
        … #计算交点
        if …: #是否发生碰撞
            … #获取碰撞信息、物体信息
        return … #返回相关变量
```

个人主页

Github @cflw https://github.com/cflw

知乎 @乘风龙王 https://www.zhihu.com/people/cflw

礼物

谢谢！