三维物体在二维平面上的投影的计算

2021 年 1 月 30 日

1 引言

假如说我们已经知道一个物体的一组顶点 (三维坐标)，应该怎样画出这个物体被看到的样子呢?

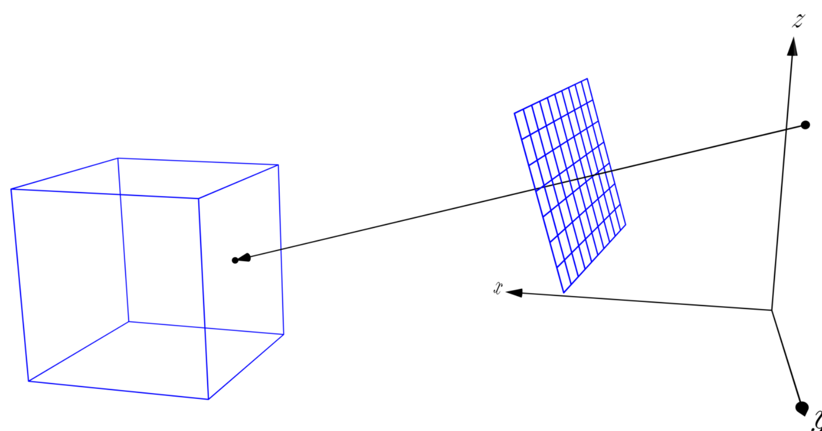

图 1-1: 渲染原理图

如图 1-1，想象在人眼 (或者摄像机) 和物体之间还有一个画布 (canvas)，也就是图中网格状的平面区域，一束光线 (图中连接两个黑点的黑色直线)，从物体表面上的某点射出，穿过画布，到达眼睛 (图右边的点)，那么当这束光线穿过画布的时候，我们想象它会在画布上留下一个「痕迹」，也就是说对于来自物体表面的光线，当它穿过画布时，我们就在画布相应的格子标上记号，如此一来，画布中被标上记号的格子总体上看起来就好像是物体在人眼中的图像. 而如果画布的格子分得越细，图像看起来也就越细腻.

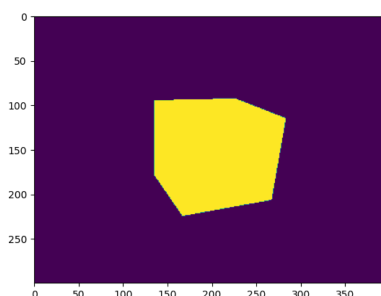

图 1-2: 效果

图 1-2是我们按照这个原理写的程序算出的一个立方体的二维图像.

2 算法

对于一个二维物体，如果我们只想知道他的形状，那么它的材质就不重要，所以说，对于图 1-1中的立方体，我们可以将它理解为现实生活中的一个空的纸箱子，我们只要把纸箱子的每一个面在画布上画出来就可以了，而无需关心箱子里面装着什么. 按照这样的思想，我们可以把纸箱子的每一个面都分成两个三角形，这样看来，这个「纸箱子」就好像是「三角形的纸板」拼接成的，尽管事实上是否真是如此并不重要. 所以说，在我们的算法中，这个立方体或者说这个「空的纸箱」就用一堆三角形来表示，也就是用一个 3 个 $n \times 3$ 的矩阵来表示，分别记它们为 `points_a`, `points_b`, 和 `points_c`，其中 `points_a` 的第 i 行就对应第 i 个三角形的第一个顶点，`points_b` 对应每个三角形的第二个顶点，`points_c` 对应第三个.

我们的最终目的是把画布上的格子 (对应输出图片上的像素) 一个个地描出来，所以还要确定画布上的各个格子的位置，以及摄影机的位置 (图 1-1中最右边那个点). 用 `camera` 表示摄影机的三维坐标，用 `lu`, `lb`, `rb`, 和 `ru` 分别表示画布左上角、左下角、右下角和右上角顶点坐标，设输出图片宽为 `width` 个像素，高为 `height` 个像素. 下面我们给出伪代码，作为思路:

```
1  import numpy as np
2
3  # 初始化画布各个格子的「状态」，同时它也是我们最终要输出的图片
4  image = np.zeros((height, width,), dtype=np.int)
5
6  # 计算出代表一个格子的长宽向量:
7  step_x = (ru - lu) / width
8  step_y = (lb - lu) / height
9
10 n_triangles = points_a.shape[0]
11 for i in range(height):
12     for j in range(width):
13         pixel = lu + i * step_y + j * step_x
14         view = pixel - camera
15
16         for k in range(n_triangles):
17             a, b, c = points_a[k, :], points_b[k, :], points_c[k, :]
18             ab, ac = b - a, c - a
19
20             z, x, y = solve_equation(
21                 'camera + z * view == a + x * ab + y * ac',
22                 'z x y'
23             )
24
25             # 无解说明射线与三角形平面平行，自然该射线不会穿过三角形
26             if no_solution((z, x, y,)):
27                 continue
28
29             # 判断射线延长后是否是在三角形区域内
30             if z > 0 and x >= 0 and y >= 0 and x + y <= 1:
31                 image[i, j] = 1
32                 break
```

以上这段代码的主要思想是：遍历每一条从 `camera` 出发，经过画布 `[i, j]` 格子的射线，对于该条射线，让它与每一个三角形比较，如果射线和三角形所在平面有交点 (也就是方程有解)，那么就判断，射线与三角形所在平面的交点是否是在三角形区域内，如果是，则说明该射线延长后能穿过三角形内部，从而对该像素置 1 并立即跳到下一个像素.

对于代码中出现的方程，或许有必要做一些解释：设摄像头的坐标是 O，设摄像头到像素点的向量是 $\boldsymbol{v}$，设三角形三个点分别是 A, B, 和 C，则方程

$$O + z\,\overrightarrow{\boldsymbol{v}} = A + x\,\overrightarrow{AB} + y\,\overrightarrow{AC} \tag{2.1}$$

准确地刻画了射线 $\{O + z\,\overrightarrow{\boldsymbol{v}} : z > 0\}$ 与三角形 △ABC 之间的位置关系.

首先看向方程2.1的等号的右边，A 是三角形上的一点，而 $\overrightarrow{AB}$ 与 $\overrightarrow{AC}$ 则分别是点 A 两条邻边的方向，换句话说，不管 x, y 是多少，向量 $A + x\,\overrightarrow{AB} + y\,\overrightarrow{AC}$ 都在三角形 △ABC 所在的平面上. 而等号的左端则比较好理解：它就是射线上的某一点. 由此可见，如果射线能够与 △ABC 所在的平面产生某种交集，那么方程2.1则必然有解.

现在假设已经求出了解 (z, x, y)，并且要求 z 是正的因为我们不考虑摄像头「后边」的情况. 通过将点 A 看做坐标轴上的原点 $(0, 0)$，将 $\overrightarrow{AB}$ 看做 x 轴上的单位向量，将 $\overrightarrow{AC}$ 看做 y 轴上的单位向量，易见：△ABC 内任何一点必须同时满足

$$\begin{cases} x + y \leq 1 \\ x \geq 0 \\ y \geq 0 \end{cases} \tag{2.2}$$

这样说看似很不严谨，但实际上总是能够做到的，通过对点 A, B, C 和 $O + z\boldsymbol{v}$ 同时施加某种几何变换，A 可以被移动到原点处，而 $\overrightarrow{AB}$ 和 $\overrightarrow{AC}$ 可以单位正交化，并且这样的几何变换过程不会改变 $O + z\boldsymbol{v}$ 已经在 △ABC 内部或者外部的事实.

读者可在 这里 找到完整的代码.