1 纹理映射

通过前面着色部分的介绍, 我们可以绘制出光滑均匀的表面. 然而, 现实中诸如木头, 锈铁这样的材质包含了丰富的细节, 表面的颜色和粗糙度等属性在表面上都不均匀. 可以想见, 想用高精度的几何表示表面的这些信息是不现实的, 更为实际的做法是将这些信息记录在二维的图像中, 然后将其附加到几何表面上. 记录表面颜色, 粗糙度等信息的图像就称为**纹理 (Texture)**, 或者称为**贴图**; 而将其映射到三维几何体上的过程就称为**纹理映射 (Texture Mapping)**.

1.1 纹理映射

定义 1.1 纹理坐标 纹理是二维图像, 我们可以通过直角坐标系中的点 (u, v) 访问其上的颜色, 一般称作**纹理坐标 (Texture Coordinate)**.

纹理映射的过程实际上就是将三维几何体上的点 (x, y, z) 映射到二维纹理图像上的点 (u, v) 的过程. 对于三角形网格模型, 我们只需要为每个顶点记录纹理坐标 (u, v), 然后通过重心插值的办法即可得到表面上所有点的 uv 坐标.

上面的过程在三维空间中是显然的. 然而, 在屏幕空间上, 我们应该如何确定每个像素对应的纹理坐标? 我们已经知道每个像素在屏幕空间中的位置, 也知道每个顶点在屏幕空间中的位置和纹理坐标. 一种简单的想法是直接在屏幕空间中进行重心插值, 然而这样得到的深度是不准确的.

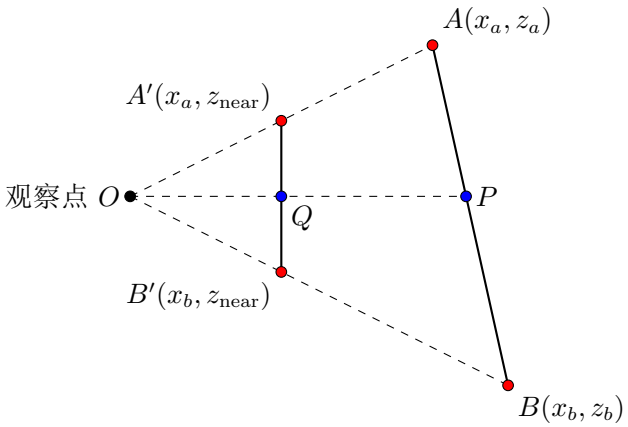

图 1: 投影变换与插值

如上图所示, 空间中的顶点 A 和 B 经过投影变换后映射到屏幕上的 A' 和 B'. 然而, 容易看出 A' 和 B' 的中点 Q 所对应的空间中的点 P 却不是 A 和 B 的中点.

看起来只有在投影变换前先进行插值才能解决这个问题, 不过这样做的开销会比较大. 实际上, 我们可以采用一种更为高效的做法, 即对屏幕空间的插值结果进行修正, 称作**透视校正插值 (Perspective-Correct**

Interpolation). 具体而言, 假定用屏幕上的坐标求得以 $\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3$ 为顶点的三角形内某一点 $\boldsymbol{q}$ 的重心坐标为

$$\boldsymbol{q} = \alpha \boldsymbol{p}_1 + \beta \boldsymbol{p}_2 + \gamma \boldsymbol{p}_3$$

那么该点经过校正的插值结果为

$$f(\boldsymbol{q}) = \frac{\frac{\alpha}{w_1} f(\boldsymbol{p}_1) + \frac{\beta}{w_2} f(\boldsymbol{p}_2) + \frac{\gamma}{w_3} f(\boldsymbol{p}_3)}{\frac{\alpha}{w_1} + \frac{\beta}{w_2} + \frac{\gamma}{w_3}}$$

其中 w_i 为顶点经过投影变换后齐次坐标未经归一化的第四个分量. 通过这种方式, 我们可以在屏幕空间高效地进行纹理坐标的插值.

1.2 纹理坐标插值

使用前面的经过校正线性插值获得的纹理坐标 (u, v) 是浮点数坐标, 而大部分纹理和正常的图像一样都是像素化存储的. 我们需要继续通过插值的方法获取该位置的颜色值.

回想 Lab 1 中神人一般的反走样, 我们可以通过双线性插值的办法获取纹理坐标的信息. 假定 (u, v) 位于纹理图像中四个像素 $(i, j), (i+1, j), (i, j+1), (i+1, j+1)$ 所围成的正方形内,(u, v) 在这一单位正方形中的坐标为 (s, t), 那么可以通过下面的公式计算出该位置的颜色:

$$\boldsymbol{c}_{uv} = (1-t)(1-s)\boldsymbol{c}_{ij} + (1-t)s\boldsymbol{c}_{i(j+1)} + (1-s)t\boldsymbol{c}_{(i+1)j} + st\boldsymbol{c}_{(i+1)(j+1)}$$

相比于寻找最近邻的像素, 这一方法能更好地避免锯齿, 得到较为平滑地结果.

1.3 纹理反走样

我们在 **Lecture 6 反走样**中已经介绍过使用 MIPMAP 进行反走样的办法, 这里就不再重复了.

1.4 纹理应用

除去用纹理记录颜色信息外, 我们还可以用纹理储存更多信息, 以实现更真实或更高效的渲染效果.

1.4.1 凹凸贴图

如果物体的表面凹凸不平, 我们不得不采用更高精度的模型来表示这些细节, 从而增加开销. 然而, 如果这些凹凸结构只影响最后的渲染, 而不影响物体的其它效果 (比如物体的运动和碰撞等), 我们就可以使用精度较低的模型, 然后将这些表面的凹凸细节记录在纹理中. 这种技术称作**凹凸贴图 (Bump Mapping)**.

实现凹凸贴图的办法有很多. 一种办法是直接记录每个点的法向量 (这是三维空间中的向量, 因此恰好可以表示为 RGB 通道上的分量, 从而以图像的形式存储); 另一种办法是通过灰度图像记录高度信息, 然后通过差分计算法向量. 无论采用哪种办法, 最终我们都可以在片元着色器中使用凹凸贴图后的法向量进行光照计算,

从而得到更加真实的表面效果.

考虑模型的法线 $\boldsymbol{n}$. 我们希望通过凹凸贴图对法向量施加一个扰动 $\Delta\boldsymbol{n}$, 即

$$\boldsymbol{n}' = \boldsymbol{n} + \Delta\boldsymbol{n}$$

对于模型上任意一点, 将其位置 $\boldsymbol{p}$ 写做 UV 坐标系下的参数方程 $\boldsymbol{p} = \boldsymbol{p}(u, v)$. 该点处的切线和副切线为

$$\boldsymbol{t} = \frac{\partial \boldsymbol{p}}{\partial u}, \quad \boldsymbol{b} = \frac{\partial \boldsymbol{p}}{\partial v}$$

于是表面法线的方向即为

$$\boldsymbol{n} = \frac{\boldsymbol{t} \times \boldsymbol{b}}{||\boldsymbol{t} \times \boldsymbol{b}||}$$

现在假定每个点的高度由纹理函数 $h(u, v)$ 给出, 那么该点处真实的高度即为模型上的点沿模型法向量 $\boldsymbol{n}$ 方向移动 $h(u, v)$ 长度, 于是新的位置为

$$\boldsymbol{p}'(u, v) = \boldsymbol{p}(u, v) + h(u, v)\boldsymbol{n}$$

现在我们来计算新的法向量 $\boldsymbol{n}'$. 对 $\boldsymbol{p}'$ 求偏导数, 可得新的切线 $\boldsymbol{t}'$ 为

$$\boldsymbol{t}' = \frac{\partial \boldsymbol{p}'}{\partial u} = \frac{\partial \boldsymbol{p}}{\partial u} + \frac{\partial h}{\partial u}\boldsymbol{n} + \frac{\partial \boldsymbol{n}}{\partial u}h = \boldsymbol{t} + \frac{\partial h}{\partial u}\boldsymbol{n} + \frac{\partial \boldsymbol{n}}{\partial u}h$$

类似地可得新的副切线 $\boldsymbol{b}'$ 为

$$\boldsymbol{b}' = \boldsymbol{b} + \frac{\partial h}{\partial v}\boldsymbol{n} + \frac{\partial \boldsymbol{n}}{\partial v}h$$

一般而言, 法向量 $\boldsymbol{n}$ 的变化 (尤其是在精度不高地模型上) 是比较小的, 因此可以忽略上式中带有 $\dfrac{\partial \boldsymbol{n}}{\partial u, v}$ 的项. 于是新的法向量可以由下式给出 (这里没有经过单位化):

$$\begin{aligned}
\boldsymbol{n}' = \boldsymbol{t}' \times \boldsymbol{b}' &= \left(\boldsymbol{t} + \frac{\partial h}{\partial u}\boldsymbol{n}\right) \times \left(\boldsymbol{b} + \frac{\partial h}{\partial v}\boldsymbol{n}\right) \\
&= \boldsymbol{t} \times \boldsymbol{b} + \frac{\partial h}{\partial u}\boldsymbol{n} \times \boldsymbol{b} + \frac{\partial h}{\partial v}\boldsymbol{t} \times \boldsymbol{n} \\
&= \boldsymbol{n} - \frac{\partial h}{\partial u}\boldsymbol{t} - \frac{\partial h}{\partial v}\boldsymbol{b}
\end{aligned}$$

最后归一化即可得到新的法向量:

$$\boldsymbol{n}' = \text{normalize}\left(\boldsymbol{n} - \frac{\partial h}{\partial u}\boldsymbol{t} - \frac{\partial h}{\partial v}\boldsymbol{b}\right)$$

在 OpenGL 中, 我们可以通过 `dFdx` 和 `dFdy` 函数方便地计算出纹理坐标的偏导数, 从而按照上面的过程实现凹凸贴图.

1.4.2 立方体贴图与天空盒

简单的纹理通常是一张二维图形. 在本节我们将讨论将多个纹理组合起来映射到一张纹理上的贴图: **立方体贴图 (Cube Map)**.

立方体贴图包含六张等大的正方形纹理图片, 对应立方体的六个面. 这里略去立方体贴图的实现方式. 我们现在讨论其一个重要应用: **天空盒 (Skybox)**.

天空盒是一个包含了整个场景的立方体, 它包含周围环境的 6 个图像, 让玩家以为他处在一个比实际大得多的环境当中.