

相机变换 → 投影变换 → 画在屏幕上

相机摆在原点      物体从三维  
View transform      宽向投影到  
                          二维照片.

正交  
透视.  
Projection transform.

MVP

Model transformation (placing objects)

View transformation (placing camera)

Projection transformation

- Orthographic

- Perspective.

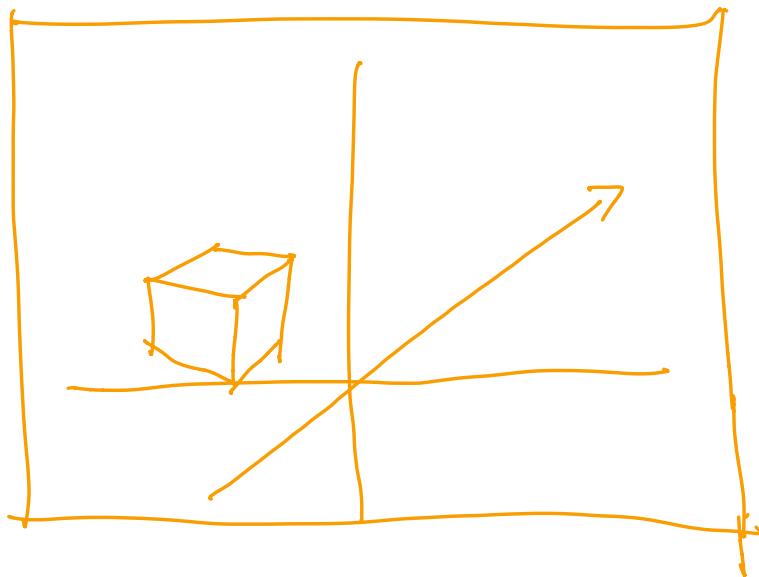
\* Canonical cube to Screen.

把物体画在屏幕上.

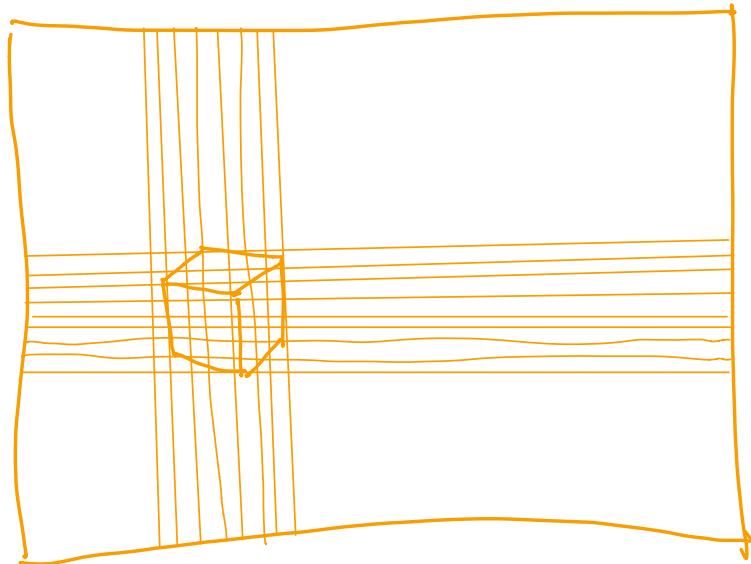
$x, y$  plane :  $[-1, 1]^2$  to  $[0, \text{width}] \times [0, \text{height}]$ .

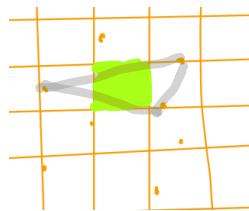
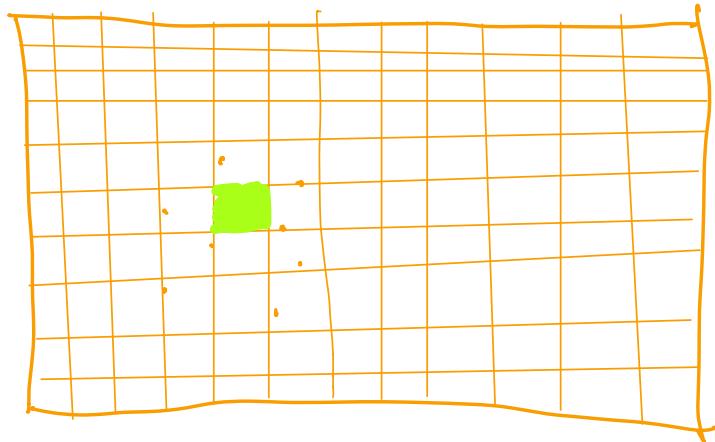
$$M_{\text{viewport}} = \begin{bmatrix} \frac{\text{width}}{2} & 0 & 0 & \frac{\text{width}}{2} \\ 0 & \frac{\text{height}}{2} & 0 & \frac{\text{height}}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

View space.



Screen space





在渲染屏幕前  
遍历屏幕像素.

在检测到在三角形  
内部的像素.

用重心坐标  
插值出这个像素在三个三角形顶  
点内坐标的权重 alpha, beta, gamma

---

已知 Triangle Vertex       $V_1$      $V_2$      $V_3$

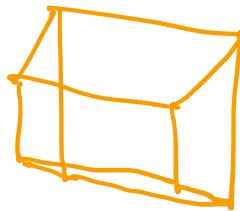
Pixel pos       $\rightarrow d \beta r$ .

$a \beta r$  插值板重

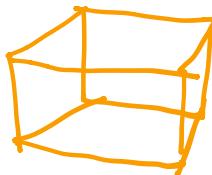
是屏幕空间得出的

的

因为  
透视  
投影变换  
把空间  
挤压了，  
在此之后  
的屏幕空间



$x \alpha \beta r$



|   |   |   |   |   |
|---|---|---|---|---|
| . | c | v | . | . |
| . | 0 | 0 | 0 | 0 |
| : | 0 | 0 | 0 | 0 |
| : | 0 | 0 | 0 | 0 |
| : | 0 | 0 | 0 | 0 |

插值输出的板重  $\alpha \beta r$   
是不对的。 stack overflow

有人回答解释了如何在  
Screen Space 插值求得 View Space  
正确未挤压的线性插值。

# Stackoverflow 讨论线性插值

<https://stackoverflow.com/questions/24441631/how-exactly-does-opengl-do-perspectively-correct-linear-interpolation>

## 转换成伪代码与语义

[https://www.comp.nus.edu.sg/~lowkl/publications/lowk\\_persp\\_interp\\_techrep.pdf](https://www.comp.nus.edu.sg/~lowkl/publications/lowk_persp_interp_techrep.pdf)

## 语义解释

[http://games-cn.org/forums/topic/zuoye3-interpolated\\_shadingcoords/](http://games-cn.org/forums/topic/zuoye3-interpolated_shadingcoords/)

正确的插值公式 ✓

屏幕上正确插值结果 =

视图空间正确的插值。

$$f = \frac{a * f_a / w_a + b * f_b / w_b + c * f_c / w_c}{a / w_a + b / w_b + c / w_c}$$

$a = \text{alpha}$  - 重心坐标,  $f_i$  插值属性,  $w_i$  顶点的前

$b = \text{beta}$  - 重心坐标,  $f_i$  插值属性,  $w_i$  顶点的后

$c = \text{gamma}$  - 重心坐标,  $f_i$  插值属性,  $w_i$  顶点的后

由以上式可插值得出近似正确的值.

下面是 shader 完整内容.

为了做 shader, 我们需要正确的  
物体表面坐标, 因此要在  
ScreenSpace 前插值获取的坐标.

$$a \cdot \text{ViewPos}[0] + b \cdot \text{ViewPos}[1] + c \cdot \text{ViewPos}[2]$$

$$\frac{V[0].w}{V[2].w}$$

Z

$$Z = 1.0 / (\alpha/\text{ViewPos}[2].w + \beta/\text{ViewPos}[1].w + \gamma/\text{ViewPos}[0].w)$$

以上计算为应用公式，将 View Space  
的 pos 在 Screen Space 坐标转换  
到的 View pos.

## phong shader

$$\text{Ambient } L_a = k_a I_a \quad k_a \text{ 固有色} \quad I_a \text{ 光强度}$$

$$\text{Diffuse } L_d = k_d (I/r^2) \max(0, n \cdot l)$$

$k_d$  固有色 光强度  
 $r^2$  光传播到点距离  
 $n \cdot l$   $n$  法线  $l$  光向量  
 $\max(0, n \cdot l)$  入射光与法向  
量夹角余弦  
而反比。

$$\text{Specular } L_s = k_s (I/r^2) \max(0, n \cdot h)^p$$

$k_s$  固有色 工光强度 + 光泽强度

$n \cdot h$   $n$  法线  $h$  半径向量

$\max(0, n \cdot h)$

高光角度限制.

光向量与摄像机  
向量的中间向量

用于和光线点乘  
求夹角，用来  
计算光反射向

量与视场向量  
夹角度，用于

限制在一定  
范围角度

判定高光.

texture Shader

V2f uv = get tex-coords

V3f texcolor = texture.getColor(u, v)

bump Shader

$$dp/du = c1 * [h(cu1) - h(cu)]$$

$$h(cu) = \text{texture\_color}(u, v) \cdot \text{norm}$$

$$h(cu1) = h(cu1/w, v) \quad w \rightarrow \text{texture.width}$$

$$= \text{texture\_color}(cu1/w, v) \cdot \text{norm}$$

$$dp/dv = \text{?} \uparrow$$

$$h(vu1) = h(u, v+1/h)$$

$$h \rightarrow \text{texture.height}$$

$\{ \frac{\partial f}{\partial u}, \frac{\partial f}{\partial v} \}$  normal

$$n = (-du, -dv, 1 \cdot f)$$

$$n = (TBN * n) \text{. Normalized}$$

$$TBN = [t, b, n]$$

$$= \left[ \frac{xy / \sqrt{x^2 + z^2}, \sqrt{x^2 + z^2}, 2y / \sqrt{x^2 + z^2}}{\text{normal } \times t \uparrow} \right]$$

normal

displacement shader

③ bump, 但将 viewpos  
进行修正

prince  $t = kn \cdot \text{changed } n \cdot$   
 $\text{texture}.color(u, v).norm$