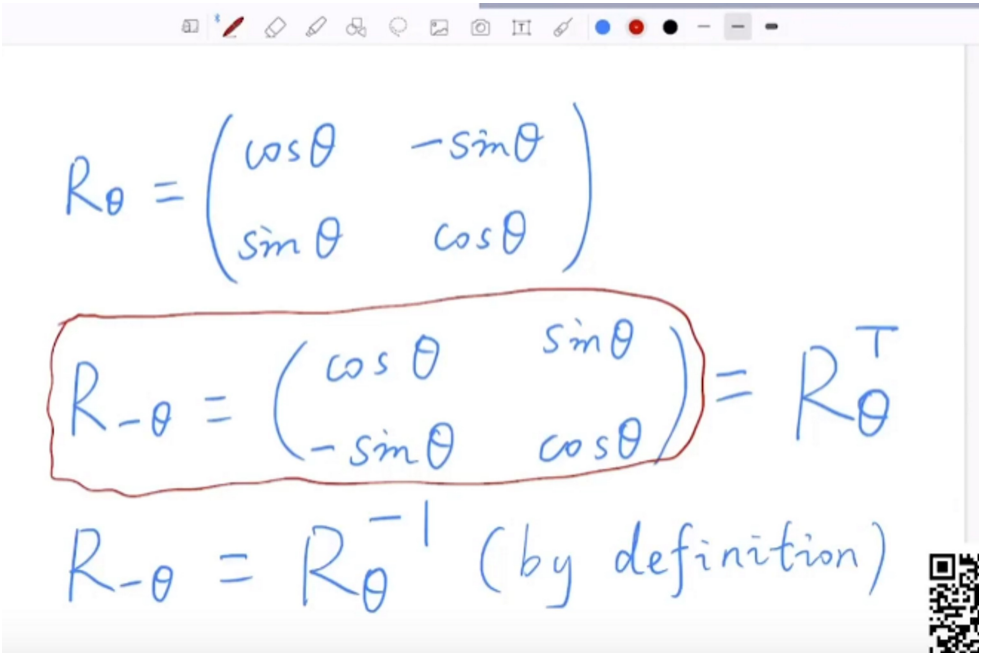
Lecture 04 Transformation Cont.



旋转θ角度和旋转-θ角度的矩阵

可以看到旋转负角度就是旋转正角度的转置矩阵

同时从定义上来说，也是互逆矩阵。

因此就是Rθ是正交矩阵!

Today

- 3D transformations
- Viewing (观测) transformation
  - View (视图) / Camera transformation
  - Projection (投影) transformation
    - Orthographic (正交) projection
    - Perspective (透视) projection

3D Transformations

Use 4×4 matrices for affine transformations

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & c & t_x \\ d & e & f & t_y \\ g & h & i & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

What's the order?

Linear Transform first or Translation first?

3D Transformations

Scale

$$\mathbf{S}(s_x, s_y, s_z) = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Translation

$$\mathbf{T}(t_x, t_y, t_z) = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

可以用齐次坐标表示3D空间中的变换

对于仿射变换，是先应用线性变换，再加上平移。

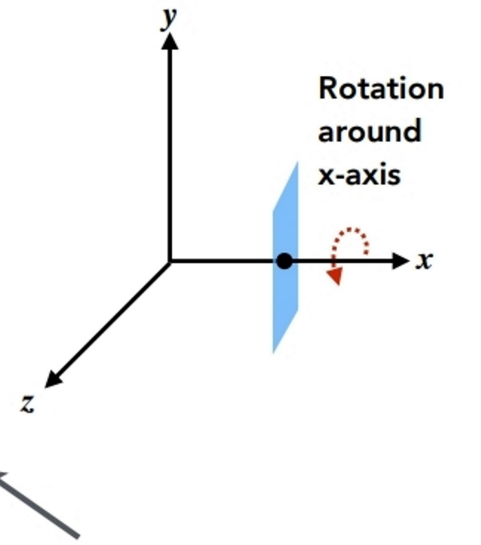
3D Transformations

Rotation around x-, y-, or z-axis

$$\mathbf{R}_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{R}_y(\alpha) = \begin{pmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{R}_z(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



3D空间的旋转

可以看到：绕哪个轴旋转，哪个坐标就不变

不过只有Ry矩阵稍微不同，其他两个都是右上角sin是负的，只有他是左下角sin是负的

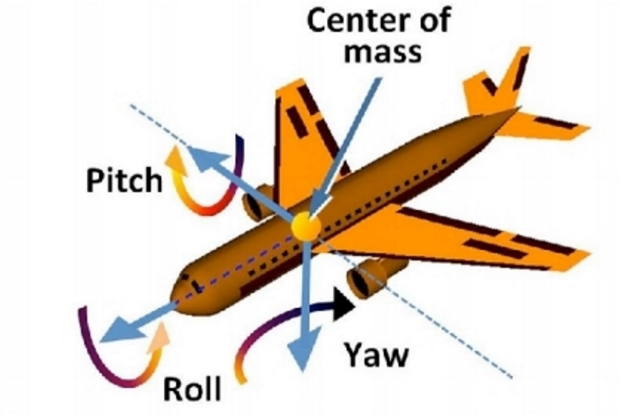
因为x叉乘y得到z，z叉乘y得到x，但是z叉乘x得到y（而不是x叉乘z），所以是反的

3D Rotations

Compose any 3D rotation from $\mathbf{R}_x, \mathbf{R}_y, \mathbf{R}_z$?

$$\mathbf{R}_{xyz}(\alpha, \beta, \gamma) = \mathbf{R}_x(\alpha)\,\mathbf{R}_y(\beta)\,\mathbf{R}_z(\gamma)$$

- So-called *Euler angles*
- Often used in flight simulators: roll, pitch, yaw



一般旋转可以用欧拉角表示，公式如上图

Rodrigues' Rotation Formula

Rotation by angle α around axis $\mathbf{n}$

$$\mathbf{R}(\mathbf{n},\alpha) = \cos(\alpha)\,\mathbf{I} + (1-\cos(\alpha))\,\mathbf{n}\mathbf{n}^T + \sin(\alpha)\underbrace{\begin{pmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{pmatrix}}_{\mathbf{N}}$$

How to prove this magic formula?

Check out the supplementary material on the course website!

可以用罗德里格斯旋转公式把任意的旋转写成一个矩阵。

这个公式定义了一个旋转轴和旋转角度。默认这个轴是过圆心的。

那么如果想绕任意一个点旋转怎么办？可以先把该点移动到圆心，旋转，再移动回去。

关于公式中的大N矩阵：

向量和向量可以写叉乘，其中可以把第一个向量写成矩阵形式，再用矩阵乘以第二个向量。
这就是我们改写的那个矩阵。

视图/相机变换

View / Camera Transformation

- What is view transformation?
- Think about how to take a photo
  - Find a good place and arrange people (**model** transformation)
  - Find a good “angle” to put the camera (**view** transformation)
  - Cheese! (**projection** transformation)

什么是视图变换（view transformation）？

想想拍照三步骤：

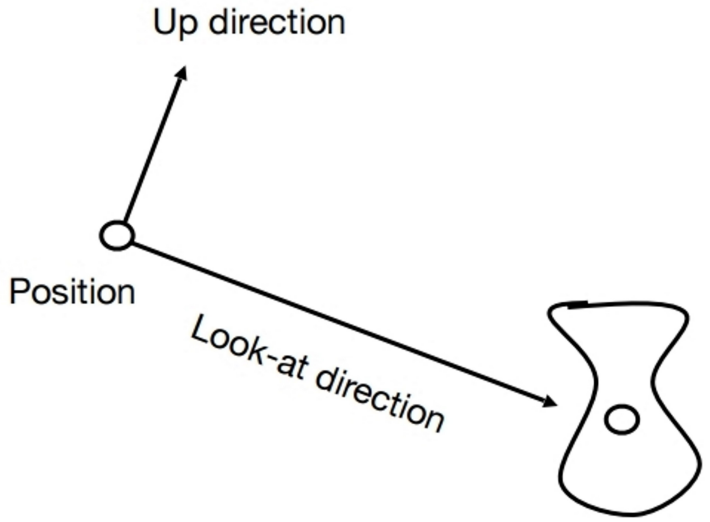
（1）确定拍摄地点和安排人物（model变换）

（2）确定相机的位置和角度（view变换）

（3）茄子！获得照片（projection变换）

View / Camera Transformation

- How to perform view transformation?
- Define the camera first
  - Position $\vec{e}$
  - Look-at / gaze direction $\hat{g}$
  - Up direction $\hat{t}$ (assuming perp. to look-at)

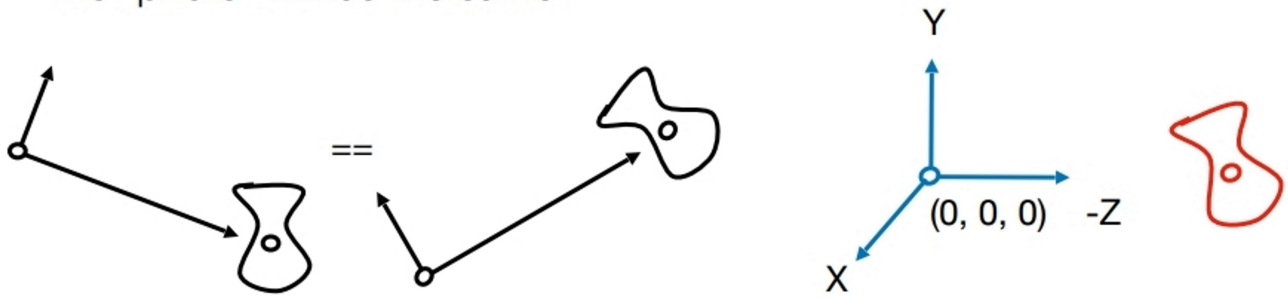


如何定义相机？三个元素：

位置；往哪儿看；向上方向

View / Camera Transformation

- Key observation
  - If the camera and all objects move together, the "photo" will be the same

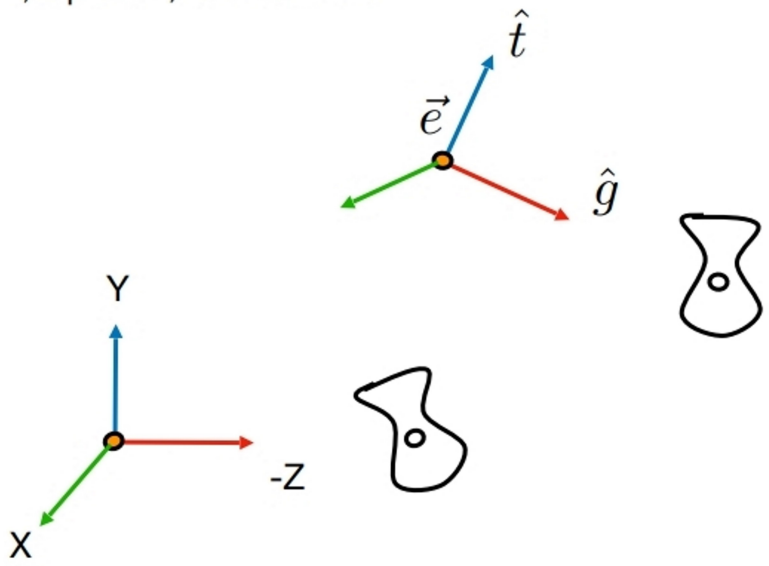


- How about that we always transform the camera to
  - The origin, up at Y, look at -Z
  - And transform the objects along with the camera

虽然现实世界中是相机移动，变换看到的景物
但在图形学中我们看作是相机不动，永远在原点

View / Camera Transformation

- Transform the camera by M_{view}
  - So it's located at the origin, up at Y, look at -Z
- M_{view} in math?
  - Translates e to origin
  - Rotates g to -Z
  - Rotates t to Y
  - Rotates (g x t) To X
  - Difficult to write!



经过变换，把相机的位置移动到原点，同时保持看到的景物不变

View / Camera Transformation

- M_{view} in math?
  - Let's write $M_{view} = R_{view}T_{view}$
  - Translate e to origin

$$T_{view} = \begin{bmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  - Rotate g to -Z, t to Y, (g x t) To X
  - Consider its inverse rotation: X to (g x t), Y to t, Z to -g

$$R_{view}^{-1} = \begin{bmatrix} x_{\hat{g}\times\hat{t}} & x_t & x_{-g} & 0 \\ y_{\hat{g}\times\hat{t}} & y_t & y_{-g} & 0 \\ z_{\hat{g}\times\hat{t}} & z_t & z_{-g} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

WHY?

$$R_{view} = \begin{bmatrix} x_{\hat{g}\times\hat{t}} & y_{\hat{g}\times\hat{t}} & z_{\hat{g}\times\hat{t}} & 0 \\ x_t & y_t & z_t & 0 \\ x_{-g} & y_{-g} & z_{-g} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

将一系列操作写成矩阵形式，如上图

注意！

- Rotate g to -Z, t to Y, (g x t) To X

这个从“歪”的坐标轴旋转回正的坐标轴，不太好写。

但是这个变换的逆过程，即：从正的坐标轴旋转到“歪”的坐标轴，是好写的，

于是我们先写从“正”坐标轴变换到“歪”坐标轴的变换矩阵，再求其逆矩阵，就可以得到待求的变换矩阵。

又因为旋转矩阵是正交矩阵，所以他的逆矩阵就只需要转置一下就可以得到了！

注意，不但相机要做这个变换，其他物体也要做这个变换，因为我们想让相机看到的景物相对不变。

（以上部分个人认为非常巧妙和关键！）

View / Camera Transformation

- Summary
  - Transform objects together with the camera
  - Until camera's at the origin, up at Y, look at -Z
- Also known as ModelView Transformation
- But why do we need this?
  - For projection transformation!

接下来“可能是最困难部分”：

Projection Transformation

- Projection in Computer Graphics
  - 3D to 2D
  - Orthographic projection
  - Perspective projection

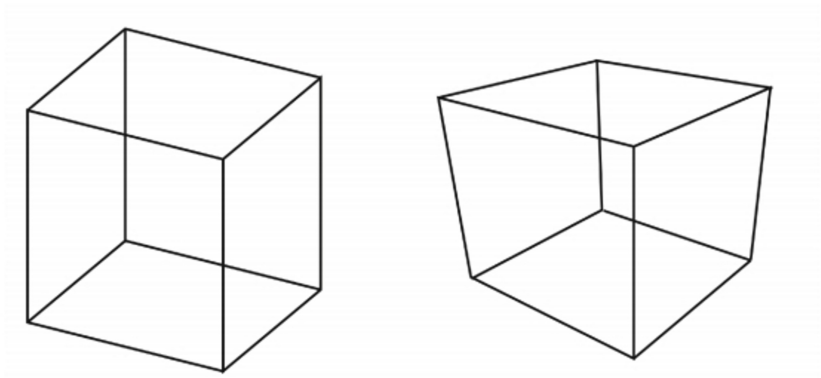


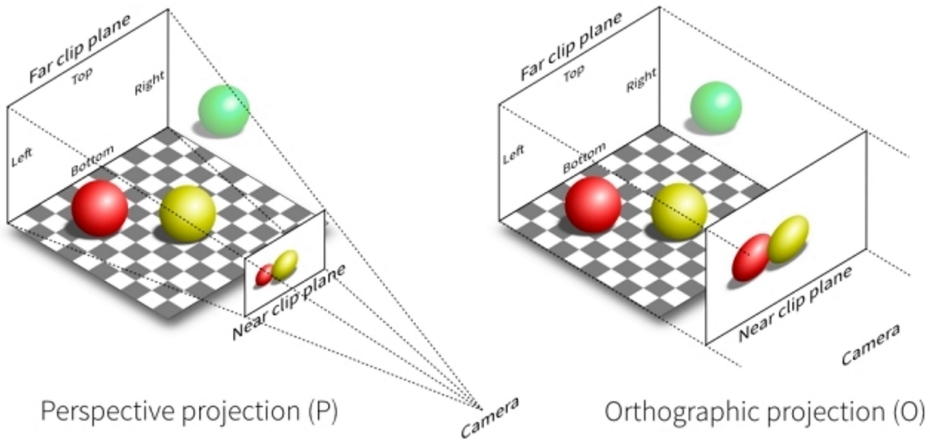
Fig. 7.1 from *Fundamentals of Computer Graphics, 4th Edition*

左：正交投影（没有近大远小）

右：透视投影（更像人眼看到的场景）

Projection Transformation

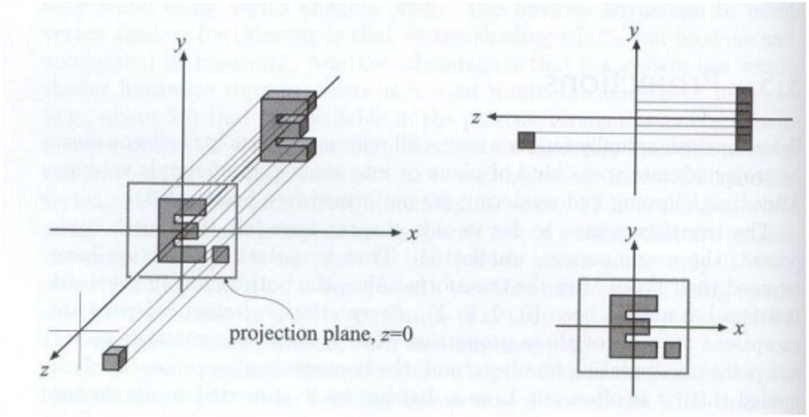
- Perspective projection vs. orthographic projection



https://stackoverflow.com/questions/36573283/from-perspective-picture-to-orthographic-picture

Orthographic Projection

- A simple way of understanding
  - Camera located at origin, looking at -Z, up at Y (looks familiar?)
  - Drop Z coordinate
  - Translate and scale the resulting rectangle to $[-1, 1]^2$



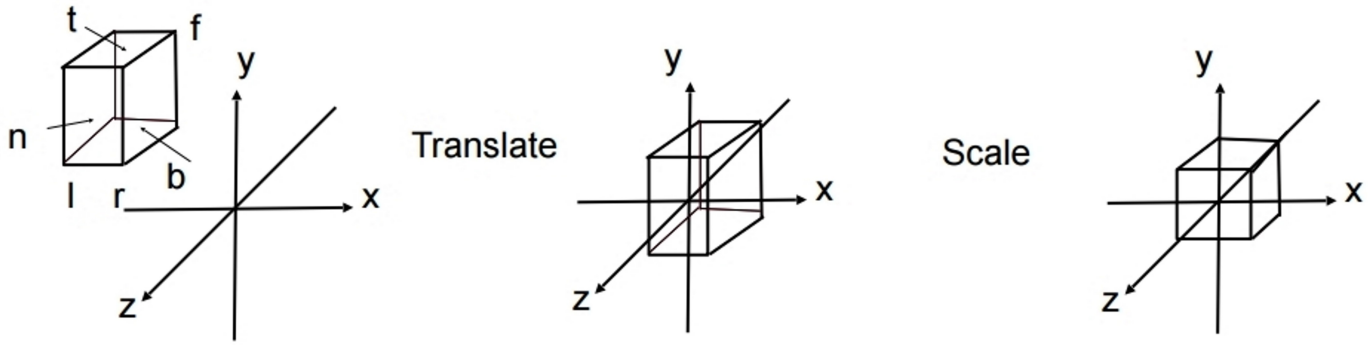
正交投影：假设光线都是平行的

上左图：扔掉z轴

上右图：扔掉x轴

Orthographic Projection

- In general
  - We want to map a cuboid [l, r] x [b, t] x [**f, n**] to the “canonical (正则、规范、标准)” cube $[-1, 1]^3$

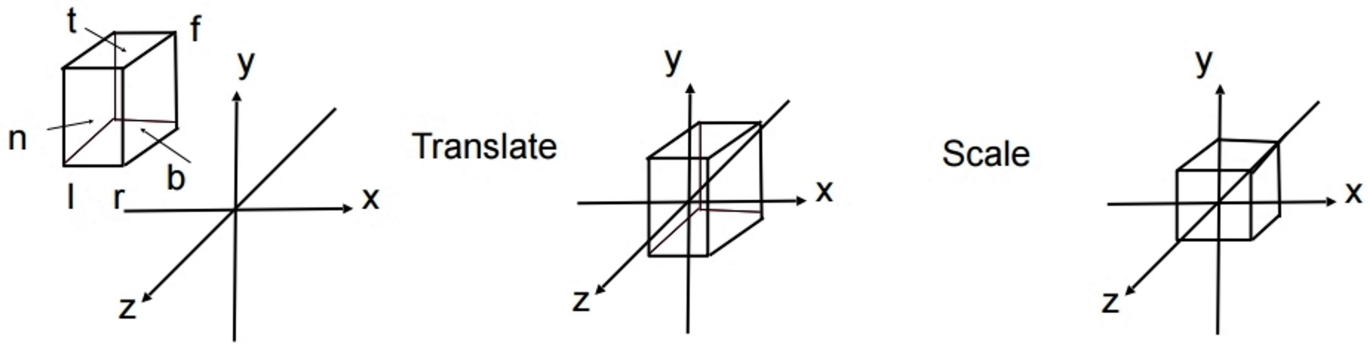


正交投影

注意坐标：左小右大，下小上大，远小近大（这是为了符合右手坐标系）

Orthographic Projection

- Slightly different orders (to the “simple way”)
  - Center cuboid by translating
  - Scale into “canonical” cube



平行投影的操作：

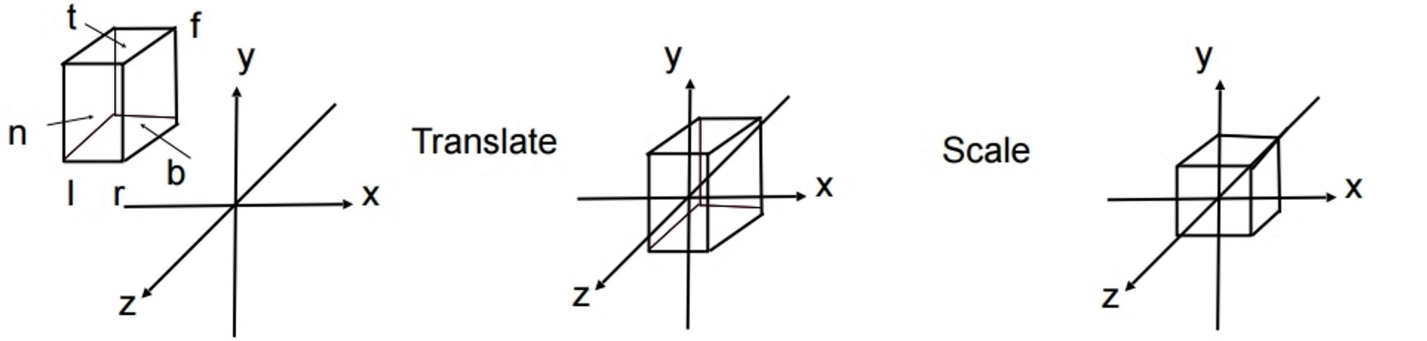
首先定义空间中一个立方体，试图将它映射成最右边那个正方体的形状（最右边称之为标准立方体）

经过了平移、缩放的操作。

Orthographic Projection

- Transformation matrix?
  - Translate (**center** to origin) first, then scale (length/width/height to **2**)

$$M_{ortho} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{n-f} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -\frac{r+l}{2} \\ 0 & 1 & 0 & -\frac{t+b}{2} \\ 0 & 0 & 1 & -\frac{n+f}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

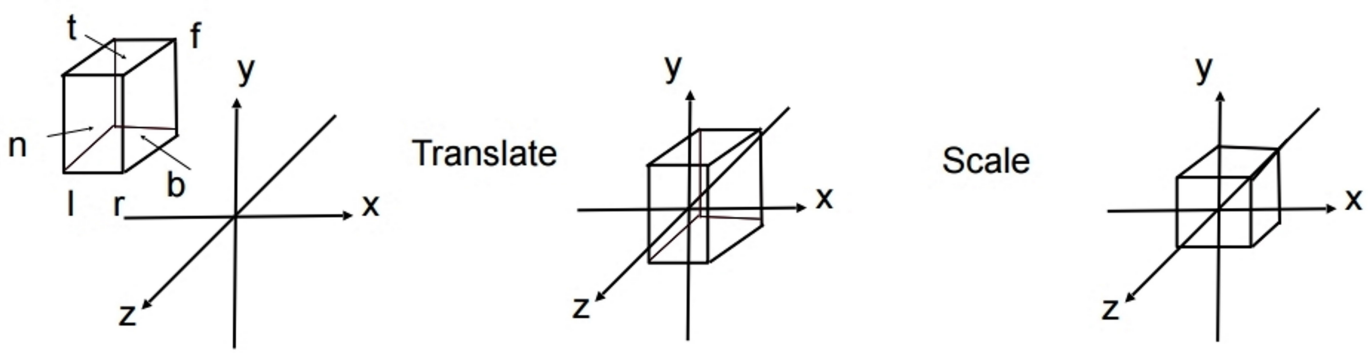


将上述操作写成矩阵形式

这就是正交投影矩阵

Orthographic Projection

- Caveat
  - Looking at / along -Z is making near and far not intuitive (n > f)
  - FYI: that's why OpenGL (a Graphics API) uses left hand coords.

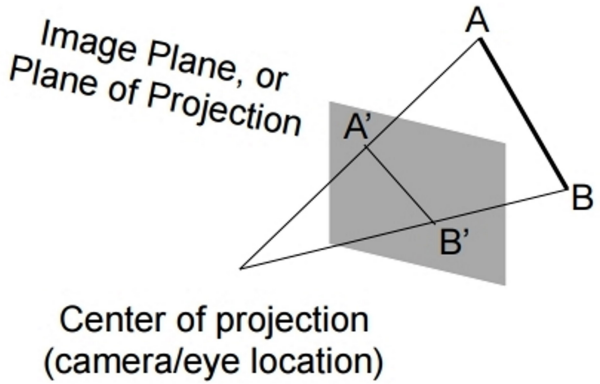


再次提醒，注意z轴是近大远小

OpenGL等API是反过来的

Perspective Projection

- Most common in Computer Graphics, art, visual system
- Further objects are smaller
- Parallel lines not parallel; converge to single point



Perspective Projection

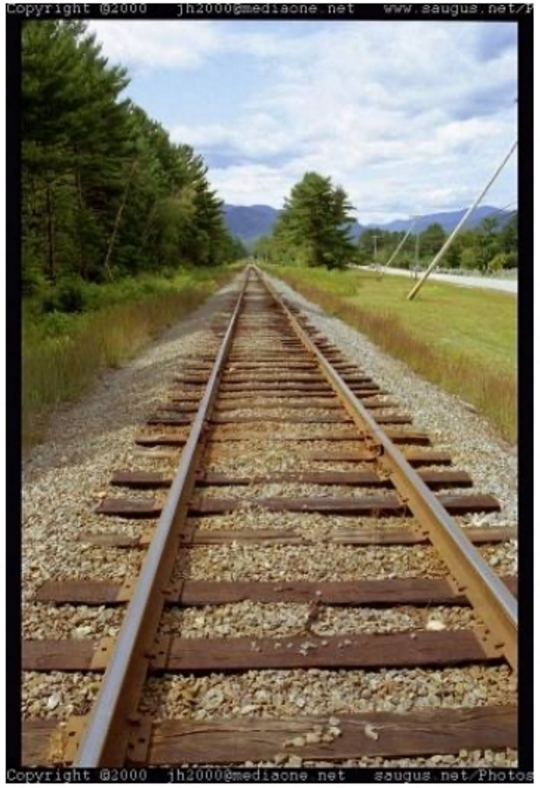
- Euclid was wrong??!!

In geometry, **parallel** lines are lines in a plane which do not meet; that is, two lines in a plane that do not intersect or touch each other at any point are said to be parallel. By extension, a line and a plane, or two planes, in three-dimensional Euclidean space that do not share a point are said to be parallel. However, two lines in three-dimensional space which do not meet must be in a common plane to be considered parallel; otherwise they are called skew lines. Parallel planes are planes in the same three-dimensional space that never meet.

Line art drawing of parallel lines and curves.

Parallel lines are the subject of Euclid's parallel postulate.[1] Parallelism is primarily a property of affine geometries and Euclidean geometry is a special instance of this type of geometry. In some other geometries, such as hyperbolic geometry, lines can have analogous properties that are referred to as parallelism.

https://en.wikipedia.org/wiki/Parallel_(geometry)



Perspective Projection

- Before we move on
- Recall: property of homogeneous coordinates
  - (x, y, z, 1), (kx, ky, kz, k != 0), **(xz, yz, z^2, z != 0)** all represent the same point (x, y, z) in 3D
  - e.g. (1, 0, 0, 1) and (2, 0, 0, 2) both represent (1, 0, 0)
- Simple, but useful

透视投影：使用最广泛的投影，近大远小

回忆一下数学基础：

齐次坐标，如果我们对点的坐标所有分量同时乘以k，他表示的还是原来那个点。

简单但是有用！

Perspective Projection

- How to do perspective projection
  - First “squish” the frustum into a cuboid (n -> n, f -> f) ($M_{persp->ortho}$)
  - Do orthographic projection (M_{ortho}, already known!)

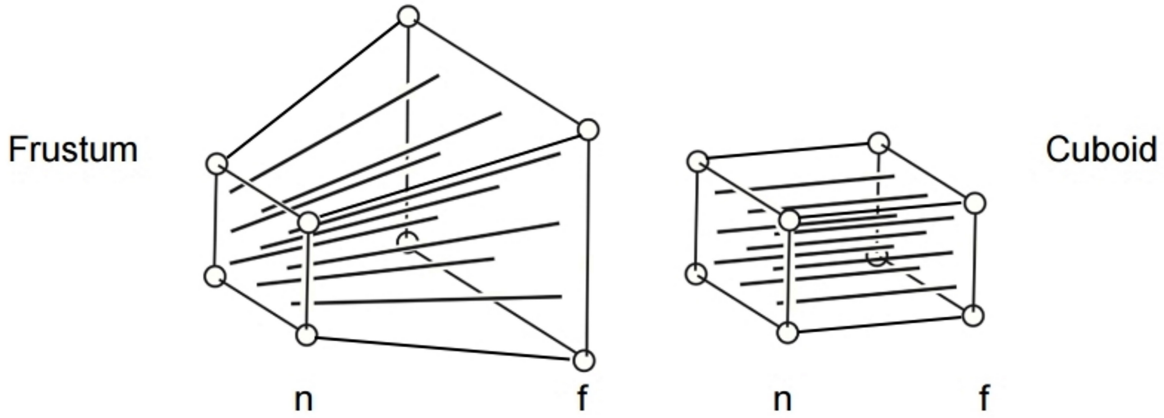


Fig. 7.13 from *Fundamentals of Computer Graphics, 4th Edition*

如何做透视投影？一个方法是分为以下两步：

1）先将frustum远平面，挤压成和近平面一样大（从左图变成右图）

2）再做正交投影，投影到近平面

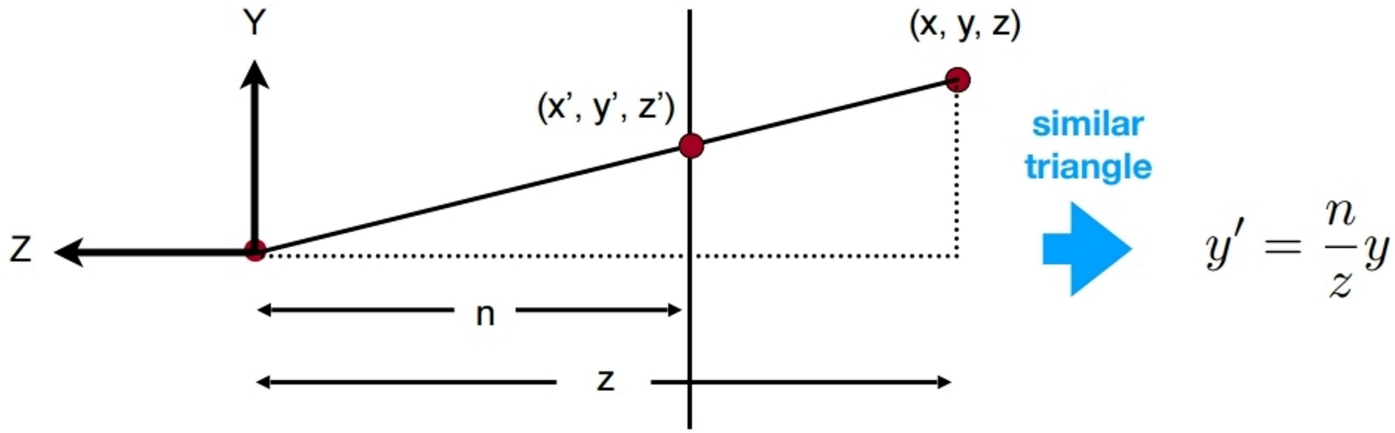
上述操作过程中几点假设：

1）近平面保持不变

2）z值保持不变，只是向内收缩

Perspective Projection

- In order to find a transformation
  - Recall the key idea: Find the relationship between transformed points (x', y', z') and the original points (x, y, z)



Perspective Projection

- In order to find a transformation
  - Find the relationship between transformed points (x', y', z') and the original points (x, y, z)

$$y' = \frac{n}{z}y \qquad x' = \frac{n}{z}x \text{ (similar to y')}$$

- In homogeneous coordinates,

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \Rightarrow \begin{pmatrix} nx/z \\ ny/z \\ \text{unknown} \\ 1 \end{pmatrix} \overset{\text{mult. by z}}{==} \begin{pmatrix} nx \\ ny \\ \text{still unknown} \\ z \end{pmatrix}$$

挤压这一步怎么做?

上图是从侧面观察frustum

用相似三角形知识可以得到新坐标的表达式，但是第三个分量目前还不知道

这里利用之前讲的那个性质：

齐次坐标，如果我们对点的坐标所有分量同时乘以k，他表示的还是原来那个点！

表示的点不变

Perspective Projection

- So the “squish” (persp to ortho) projection does this

$$M_{persp\to ortho}^{(4\times 4)}\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} nx \\ ny \\ \text{unknown} \\ z \end{pmatrix}$$

- Already good enough to figure out part of $M_{persp->ortho}$

$$M_{persp\to ortho} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ ? & ? & ? & ? \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

WHY?

整理一下，原来远平面上的点经过投影以后变成了$(nx, ny, \text{未知}, z)$

这样已经能够推导出变换矩阵的大部分内容了。还剩第三行

Perspective Projection

- How to figure out the third row of $M_{persp->ortho}$
  - Any information that we can use?

$$M_{persp\to ortho} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ ? & ? & ? & ? \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

- Observation: the third row is responsible for z'
  - Any point on the near plane will not change
  - Any point's z on the far plane will not change

那么第三行求解呢？注意两个性质

（1）任何近平面上的点不会改变（也就是对于任意的$(x, y, n, 1)$，经过这个矩阵变换后，点的位置仍然不变）

（2）任何远平面上的点，z值不会改变

Perspective Projection

- Any point on the near plane will not change

$$M_{persp\to ortho}^{(4\times 4)}\begin{pmatrix}x\\y\\z\\1\end{pmatrix}=\begin{pmatrix}nx\\ny\\\text{unknown}\\z\end{pmatrix}$$

replace z with n

$$\begin{pmatrix}x\\y\\n\\1\end{pmatrix}\Rightarrow\begin{pmatrix}x\\y\\n\\1\end{pmatrix}==\begin{pmatrix}nx\\ny\\n^2\\n\end{pmatrix}$$

- So the third row must be of the form (0 0 A B)

$$\begin{pmatrix}0 & 0 & A & B\end{pmatrix}\begin{pmatrix}x\\y\\n\\1\end{pmatrix}=n^2$$

n^2 has nothing to do with x and y

点$(x,y,z,1)$是可以通过矩阵变换得到$(nx,ny,unknwon,z)$向量的。

根据上文提到的性质（1），经过这个变换，点实际没有改变

而同时，$(x, y, z, 1)$本身可以写成$(x, y, n, 1)$（为什么把z替换成n？因为近平面的z坐标就是都是n，所以可以做这个替换。）然后同时乘以n，变成(nx, ny, n^2, n)

经过上面两个推导，可以看出，第三行前两个数一定是0

因为n^2这个分量和x和y都毫无关系，因此前两个数必定是0

这样，我们就解出了第三行前两个数，都是0

接下来求A和B

Perspective Projection

- What do we have now?

$$\begin{pmatrix} 0 & 0 & A & B \end{pmatrix} \begin{pmatrix} x \\ y \\ n \\ 1 \end{pmatrix} = n^2 \quad \Rightarrow \quad An + B = n^2$$

- Any point's z on the far plane will not change

$$\begin{pmatrix} 0 \\ 0 \\ f \\ 1 \end{pmatrix} \Rightarrow \begin{pmatrix} 0 \\ 0 \\ f \\ 1 \end{pmatrix} == \begin{pmatrix} 0 \\ 0 \\ f^2 \\ f \end{pmatrix} \quad \Rightarrow \quad Af + B = f^2$$

远平面上有一个特殊点，$(0,0,f)$经过变换挤压仍然不变

所以$(0,0,f)$经过变换仍然是$(0,0,f)$

根据近平面我们得到

$$An + B = n^2$$

根据远平面的中心点我们得到

$$Af + B = f^2$$

Perspective Projection

- Solve for A and B

$$An + B = n^2$$
$$Af + B = f^2$$

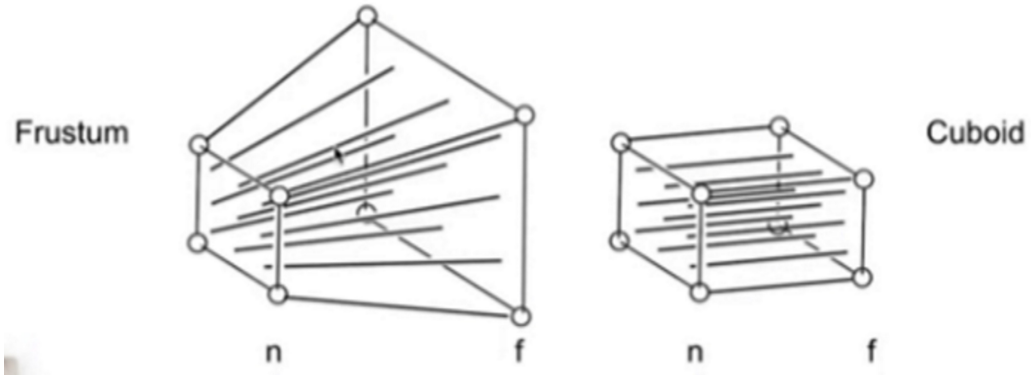
$$A = n + f$$
$$B = -nf$$

- Finally, every entry in $M_{persp->ortho}$ is known!
- What's next?
  - Do orthographic projection (M_{ortho}) to finish
  - $M_{persp} = M_{ortho} M_{persp \to ortho}$

这样我们就能解出A和B了，

这样终于把从透视投影挤压成正交投影的矩阵，解出来了！

思考题：



经过这样的挤压，frustum中间的点其z值会变大还是变小？