Computer Graphics

Shi-Min Hu

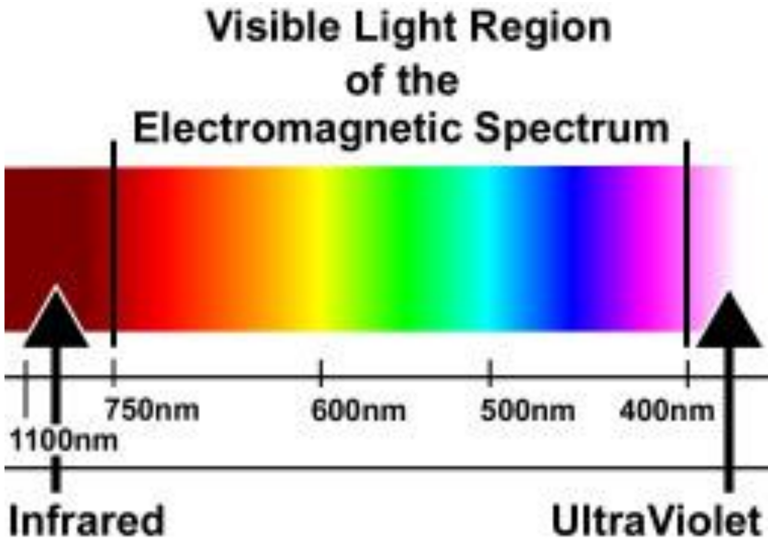
Tsinghua University

Overview of Today

- Some important concepts in Graphics
  - Color
  - Image & Pixel (象素)
  - Triangle Mesh
  - Lighting & Shading （绘制）
- Transformation (变换) and Viewing

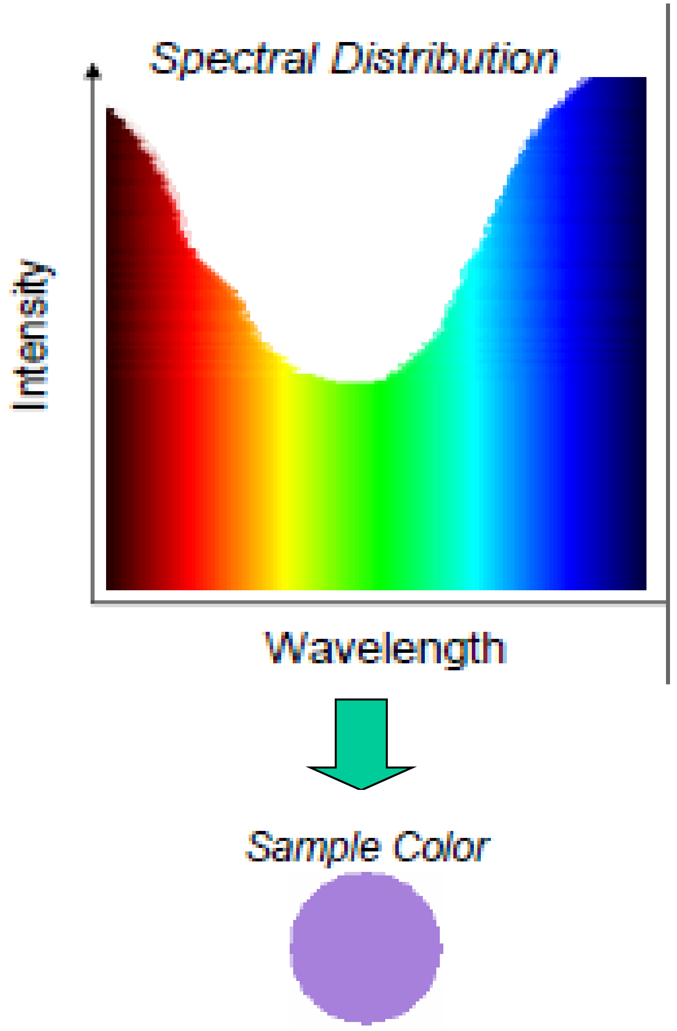
Color Perception

- **What is Color?**
  - Colors are the sensations that arise from light energy of different wavelengths
    - electromagnetic waves (电磁波) with different wavelengths are corresponding to different colors
    - Human's eye are sensitive for electromagnetic wave of wavelengths between 380 to 760 nm



Spectral Distribution of Light

- "Light" is a mixture of many wavelengths, each with some intensity
  - E.g., *White Light means light include all* wavelengths with the same intensity.
- *spectral distribution*: intensity as a function of wavelength over the entire spectrum



- So, colors can be represented as such distribution function
- However, use spectrum distribution for color representation is too complicated, and the corresponds is multiple-to-one.
- There exits a situation as “different spectrum distribution with the same color” ”(异谱同色)

RGB Color Space

- We first introduce color space for color representation. RBG is commonly used color space in Graphics
  - Color is represented as RGB triple (r,g,b);
  - People perform almost all operations on each channel separately
  - usually, each color value is a float value range from 0 to 1, or 0-255 if we use 8 bit integers.

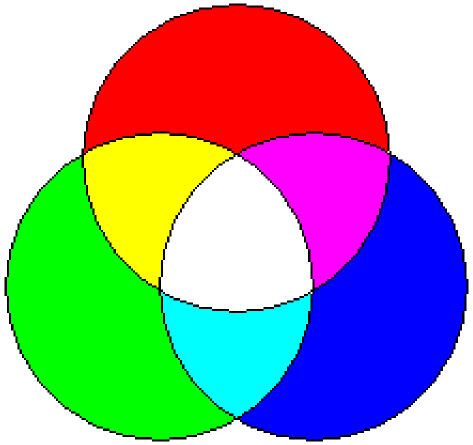
RGB Color Space

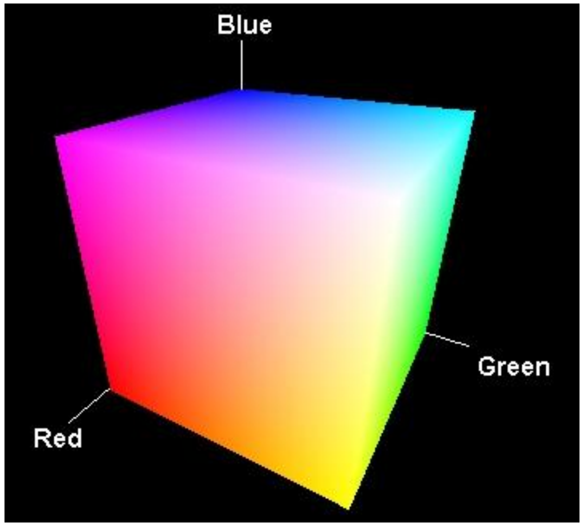
- Color are represented as combinations of three primaries: red, green, blue)
  - $\mathbf{C} = r\mathbf{R} + g\mathbf{G} + b\mathbf{B}$
  - The reason why we pick red, green, and blue?
    - essentially because of the structure

      of our visual system
    - Our visual system is sensitive to those three colors
- A demo for Color

RGB Color Space

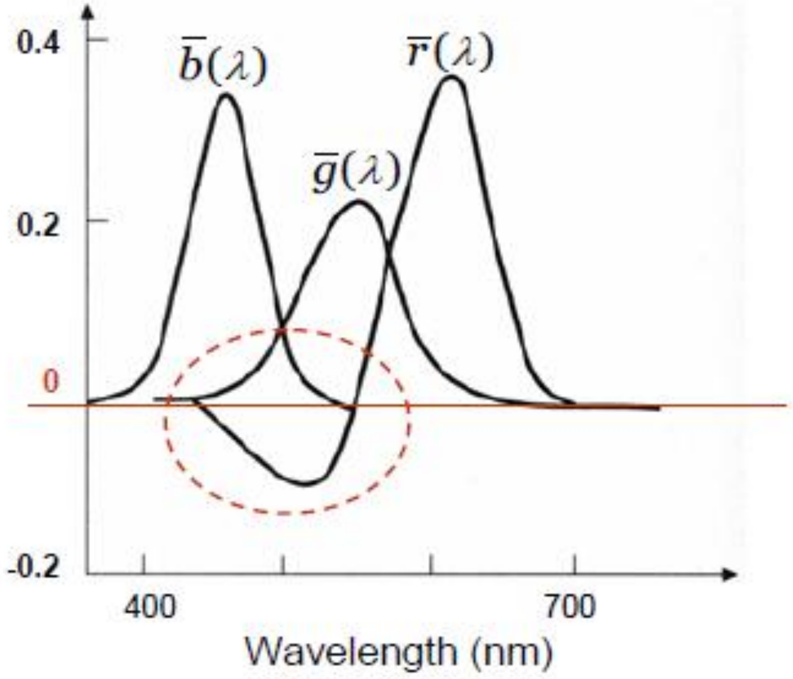
- Color are represented as combinations of three primaries: red, green, blue)
  - $\mathbf{C} = r\mathbf{R} + g\mathbf{G} + b\mathbf{B}$
  - The reason why we pick red, green, and blue?





RGB Color Space

- Unfortunately ,Some colors cannot be written as combinations of RGB triples, because some parts of the red curve is negative



Other Color Spaces

- So some other color space are also used
  - CMY
  - HSV
  - CIE XYZ

CMY

- CMY :The other set of primaries besides RGB
  - Cyan(青), magenta（品红）, and yellow — complements （补色）of red, green, blue

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

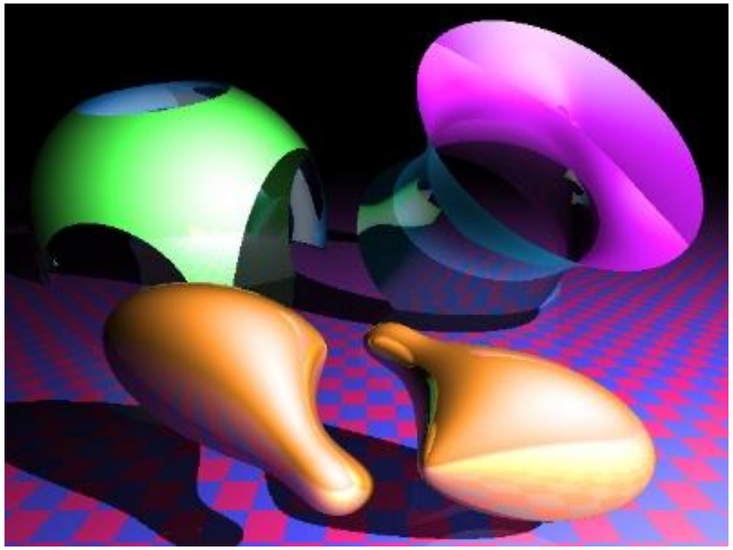
- It’s called **subtractive** primaries（减色系统）
  - RGB are additive primaries — start with black, add up to white
  - CMY — start with white and subtract colors from white

HSV Color space

- True color pictures with 3 components (RGB), each has 256 possible value, so we have up to 16,777,216 possible colors.
- Controlling this huge amount of possibilities is almost impossible without the HSV space.

HSV provides an intuitive method for color selection.

- Applications: image processing, fractal pictures, ray tracing…



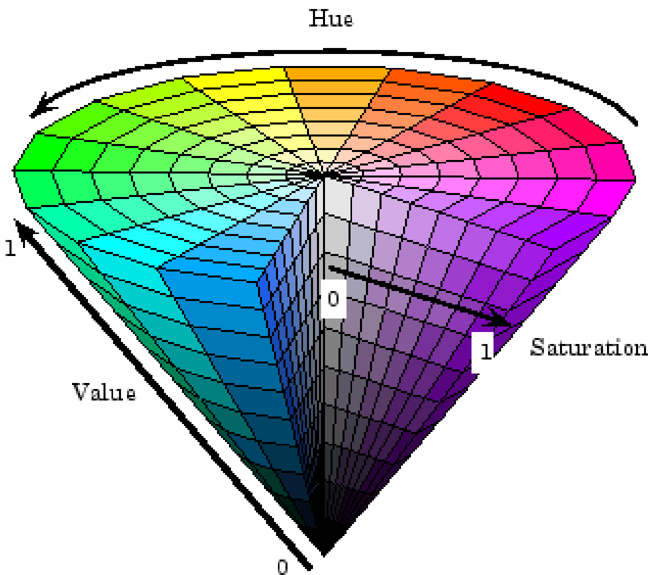
HSV

- HSV: a color space in cone-shaped
  - Hue (色调) means the base color, which is a main factor indicates difference between colors.
  - Saturation(饱和度) is for purity of color (decrease = adding white)
  - Value of brightness(亮度)

    Luminance of lights,

    (decrease = adding black

    more user-friendly than RGB

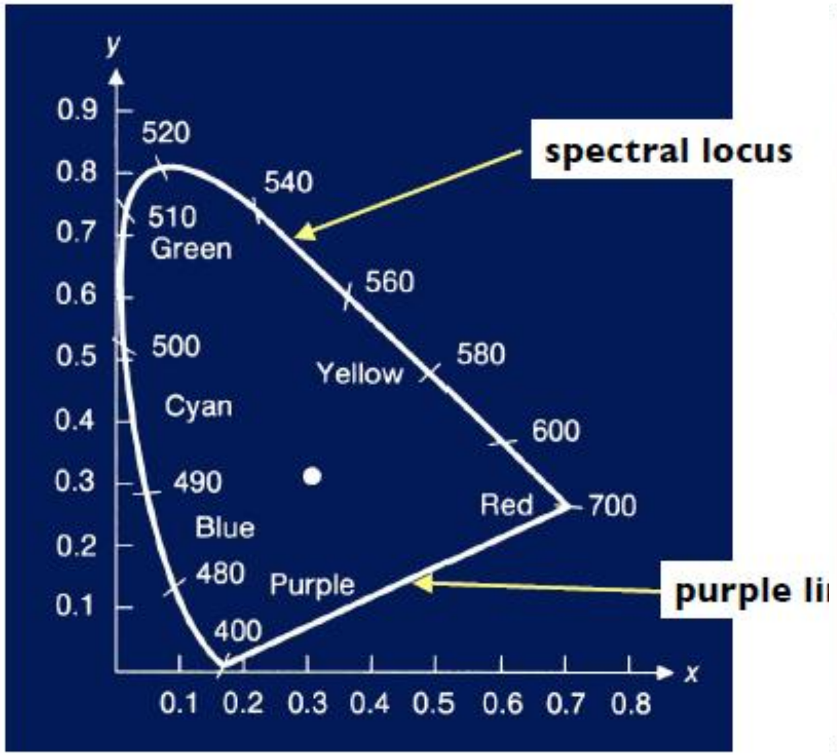


CIE XYZ

- CIE XYZ color space
  - Proposed by CIE (International Commission on Illumination) in 1931
- It can represent all perceptible colors (while RGB not)
  - most used in color science
  - based on human perception studies

CIE XYZ (Chromaticity Diagram)

This diagram shows how to use XYZ to represent colors. Z is equal to 1 – x – y.



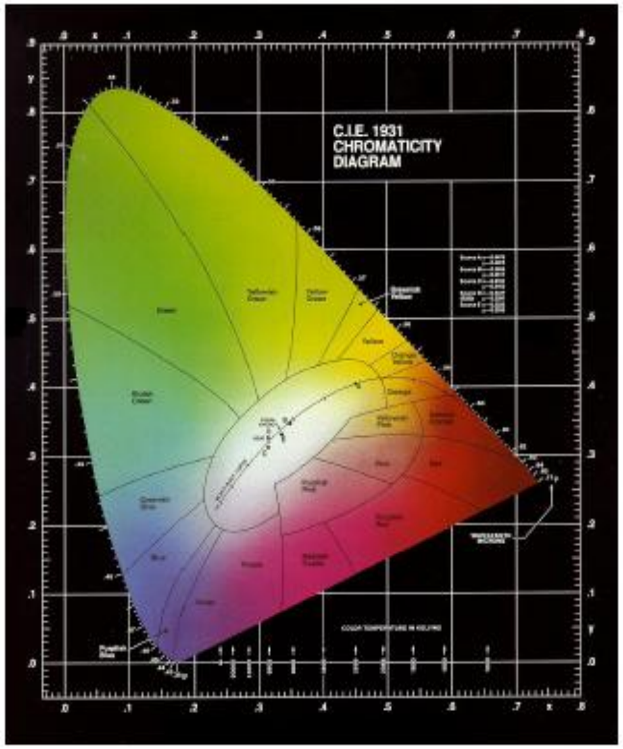
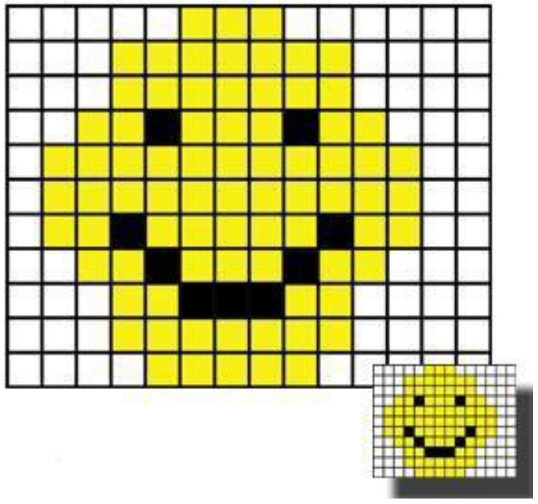


Image & Pixel

- Image
  - image could be treated as a 2-dimensional discrete function f(x,y)
  - each discrete grid is called pixel (象素)
  - Usually, each pixel has a RGB(or RGBA) value for color image or

    a scalar value for gray image



Triangle Mesh

- What’s objective of graphics?
  - Given a 3D scene and a camera position, generate (render) a 2D image.
  - What is the data structure/representation of 3D scene?
    - For Simple primitives, such as sphere, cube…
    - Complex models, usually use triangle mesh or parameter curves/surfaces

- Mesh Description: usually contains a list of faces F, and a list of vertices V
  - A list of faces F = $(f_1,f_2,\ldots,f_n)$
    - Each face is a triangle
  - A list of vertices V = $(v_1,v_2,\ldots,v_n)$
  - Each face in F is a list of indices in V

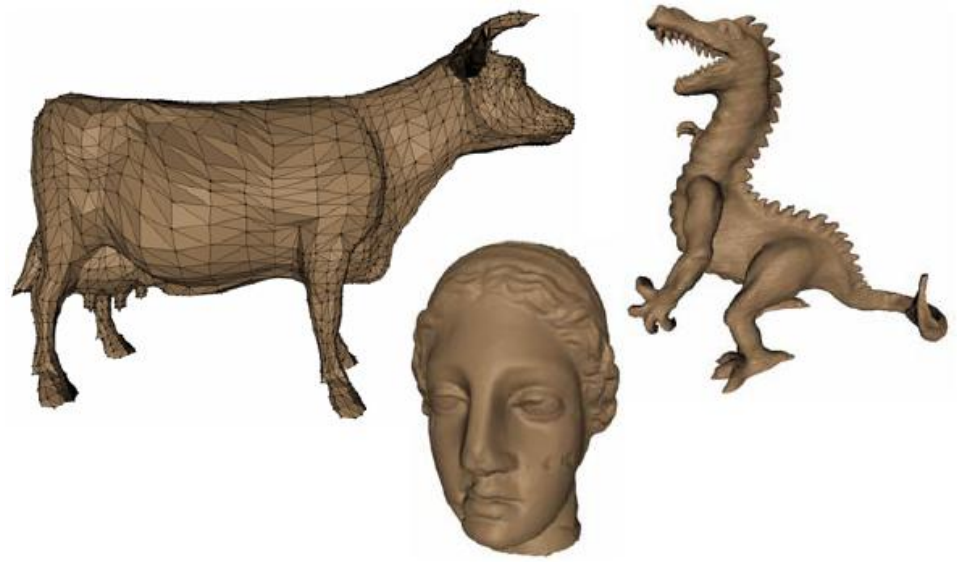
    e.g.

    f_1 --(v_1,v_2,v_3), f_2 --(v_4,v_5,v_6),

    f_3 --(v_7,v_8,v_9), …

Some Samples

- Examples of triangle mesh models, a bull, a dragon and a head model. We draw triangles over bull model, and show the other two models by shading.

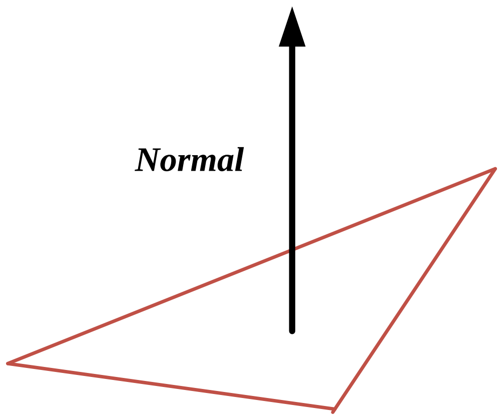


Normal

- Face Normal
  - Each face has a normal direction to define the orientation
- Vertex Normal
  - Each vertex is shared by m faces
  - Compute normal by Interpolation

    $Nv= (N_{f1}+N_{f2} \ldots + N_{fm})/m$
  - or by area based Interpolation

    $Nv= (|F1|*N_{f1}+|F2|*N_{f2} \ldots + |Fm|*N_{fm})/(|F1|+|F2|+\ldots|Fm|)$

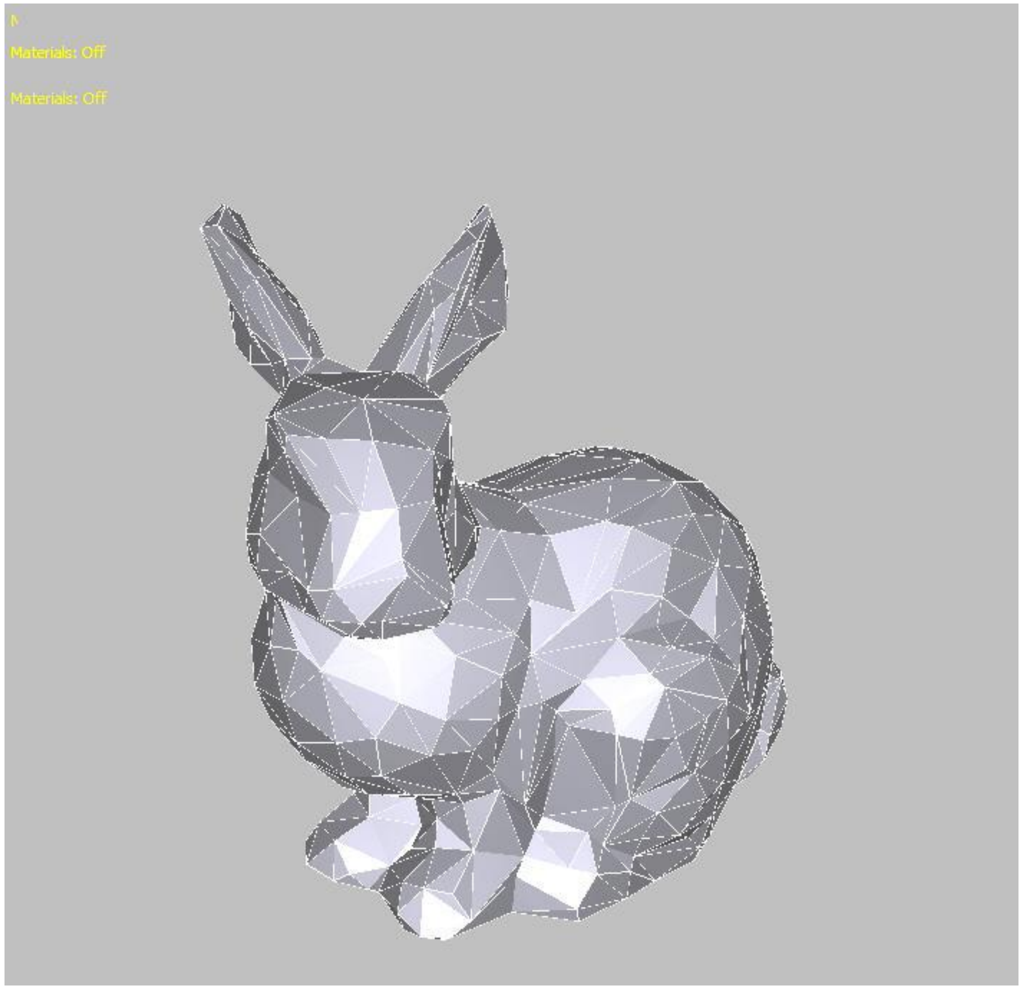


Triangle Mesh

- Rendering of mesh models
  - In each point of the mesh surface, we need to specify a color
    - By Color: Given constant color to each face
    - By lighting
      - Suppose a light exists in the scene and illuminates the scene.
      - How to calculate lighting?

Rendering results

- Wireframe
- Constant Color
- Lighting



Lighting model

- To calculate the intensity of light, we need an lighting model (illumination model)
  - Local Lighting
    - Concerned with how objects are directly illuminated by light sources
  - Global Lighting
    - Includes shadow effects
    - Includes lighting effects from locations other than light sources, such as reflections, refractions

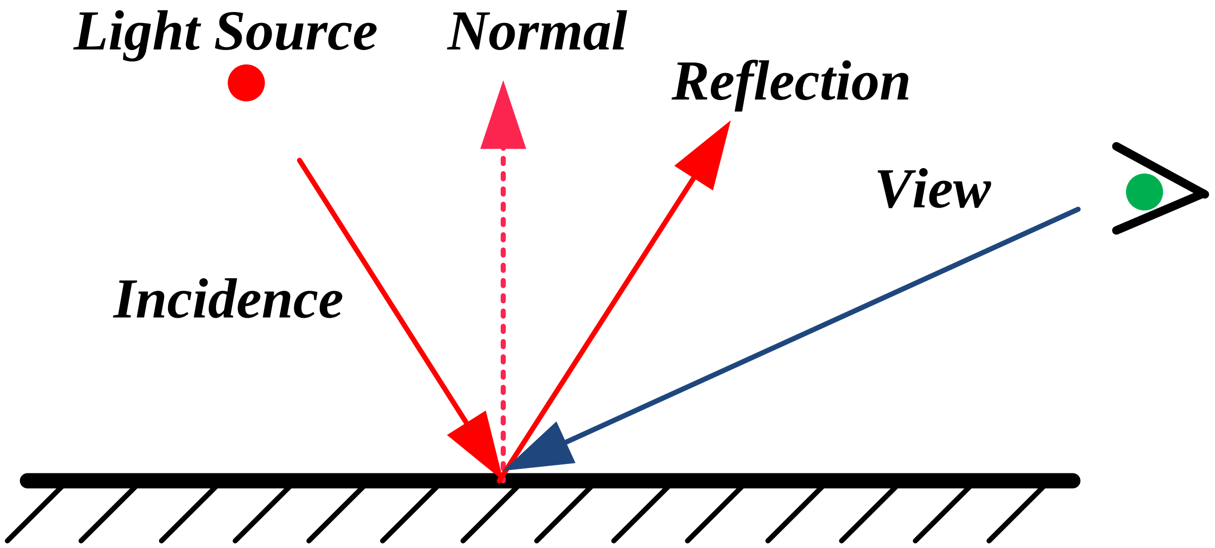
History of Lighting

- In 1967, Wylie: first added lighting effects into rendering
  - Intensity is inverse of the distance to the light
- In 1970, Bouknight: introduced the first lighting model:
  - Lambert diffuse reflection(漫发射) + ambient (No specular lighting)（环境光）
  - Communication of ACM

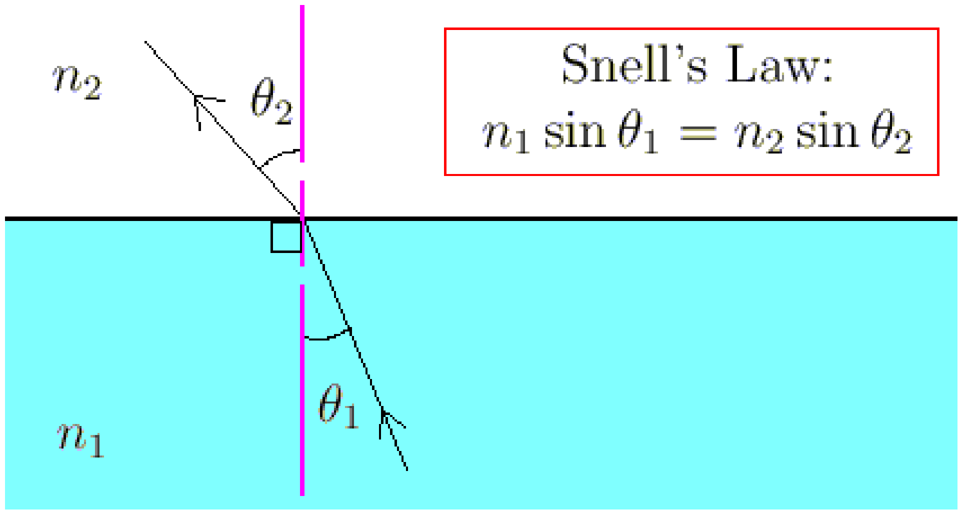
- In 1971, Gourand: gourand shading
  - Lambert diffuse + Bicentric interpolation
  - IEEE transactions on Computers
- In 1975, Phong: proposed extended the model by further considering specular:
  - Diffuse（漫发射） + ambient（环境光） + specular（高光）
  - The most influenced lighting model
  - Communication of ACM

Related Physics: The propagation of light

- The propagation of lights obeys law of reflection:
  - the angle of incidence = the angle of reflection
  - incoming ray, reflection ray and normal is in the same plane



- The law of refraction (also called Snell's Law):
  - the ratio of the sin of the angles of incidence and of refraction is a constant that depends on the media
  - The constant is called the relative refractive index



Snell's Law:

$$n_1 \sin\theta_1 = n_2 \sin\theta_2$$

Energy

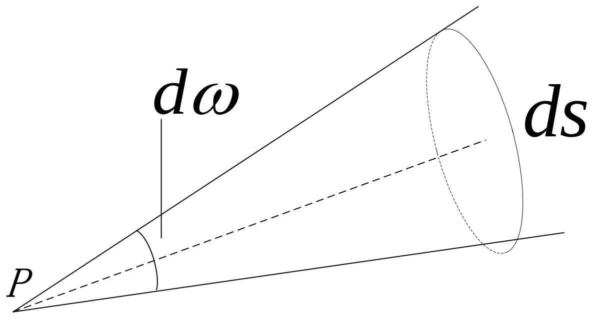
- Propagation of lights should satisfy Energy conservation :

$$I_i = I_d + I_s + I_t + I_v$$

  - **I***i* is the energy of incident light
  - **I***d* is the energy of diffuse reflection
  - **I***s* is the energy of specular reflection （镜面反射）
  - **I***t* is the energy of refraction （折射）
  - **I***v* is the energy that is absorbed

Measure of light

- Solid Angle（立体角）:
  - How big an object appears to an observer at point P $$d\omega = \frac{ds}{r^2}$$
  - Max Value: 4π
- Irradiance （辉度）E
  - E defined as: light energy per unit time arrived at per unit area
- Radiance （发光强度）I
  - I defined as: irradiance per unit solid angle

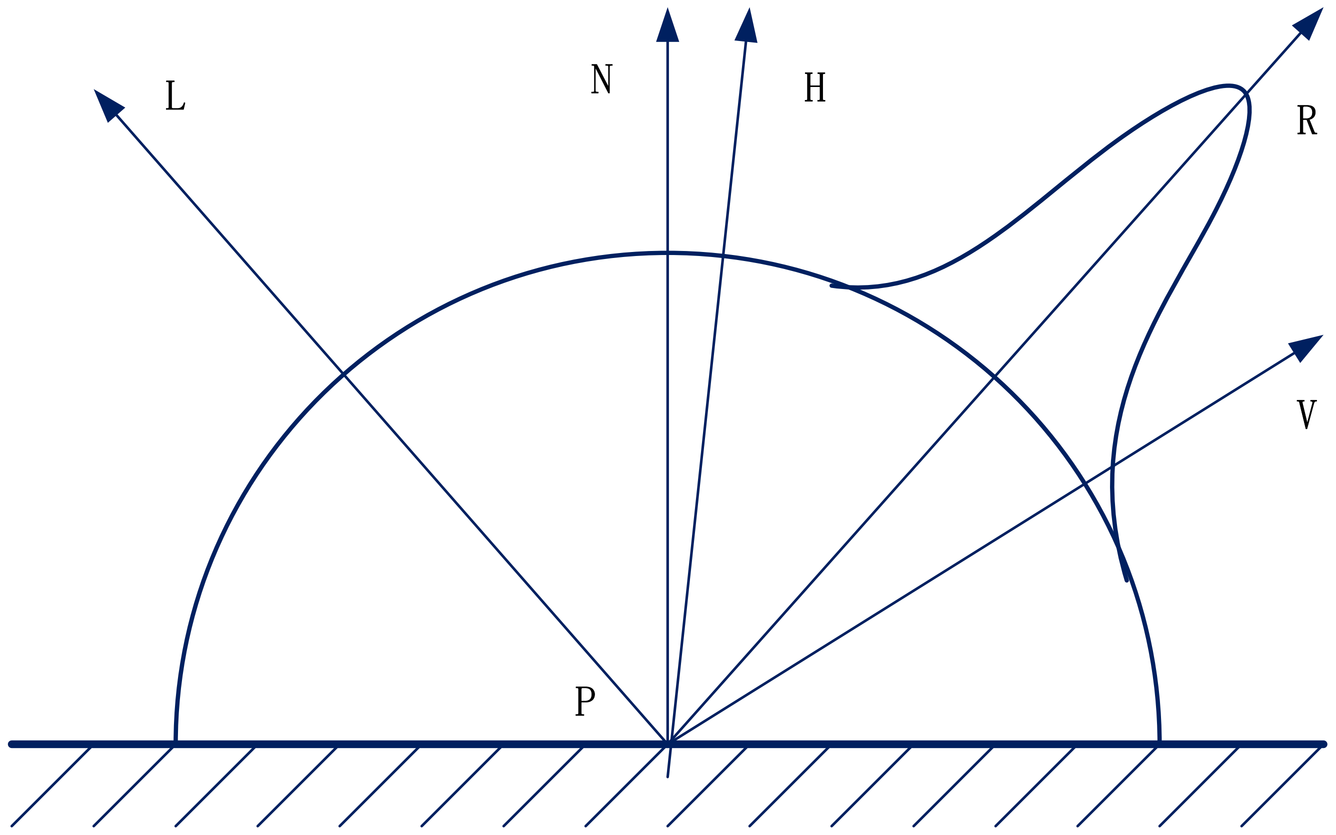


Phong Lighting Model

- Support point light or directional light
- Local lighting model
  - Decompose lighting effects to 3 parts:
    - Diffuse Reflection
    - Specular Reflection
    - Ambient lighting(to approximate the global Illumination effects)

Phong Lighting Model Illustration

L is the incidence ray, V means the direction of the viewer, N is the normal, and R is reflection of L; H is the half of L and V



Phong Lighting Model

- Diffuse reflection
  - The propagation of diffuse reflection rays is isotropic.
  - The intensity of diffuse reflection is defined as:

$$I_d = I_i K_d * (L \cdot N)$$

  - K_d is the diffuse reflection constant
  - K_d has three components k_{dr}, k_{dg}, k_{db} represent R,G,B diffuse reflection constant respectively.

    K_d implies the base color of the model

Phong Lighting Model

- Specular reflection
  - For glossy surfaces, the reflected rays always exist in a very narrow solid angle area, determined by the direction of reflection law
  - The intensity of specular reflection is defined as:

$$I_s = I_i K_s * (R \cdot V)^n$$

  - K_s is the specular reflection constant ; and n is the shininess constant, which decides how shiny the surface is.

Phong Lighting Model

- Ambient reflection
  - To approximate global illuminations (including indirect lighting, indirect reflection…)
  - The intensity of ambient reflection is defined as:

$$I_a = I_i K_a$$

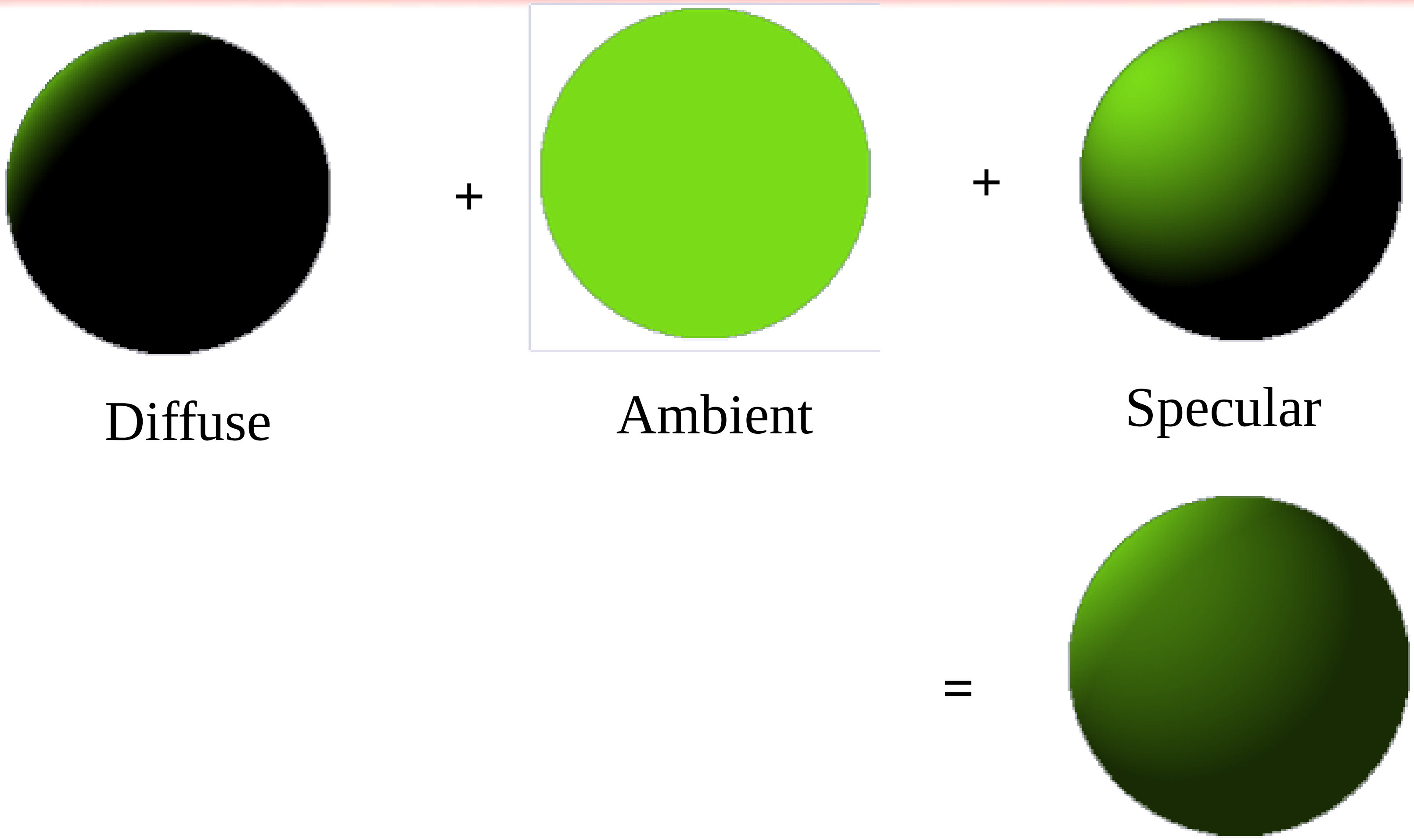
  - K_s is the ambient reflection constant

Phong Lighting Model

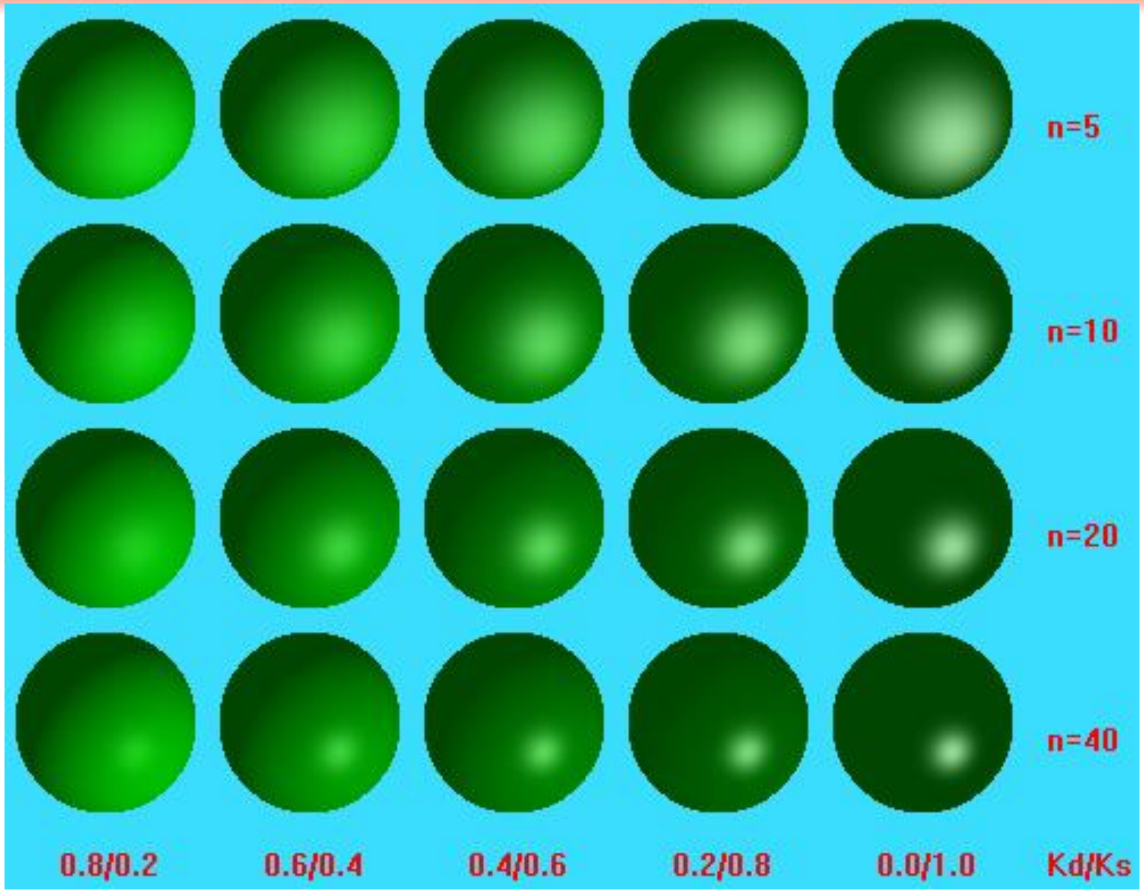
- The reflection intensity is the sum of diffuse reflection, specular reflection and ambient reflection:

$$I = I_i K_a + I_i K_s * (R \cdot V)^n + I_i K_d * (L \cdot N)$$

Example 1



Example 2



Wu Jianhua’s Demo

Demo

http://cg.cs.tsinghua.edu.cn/course/

Phong and Gouraud Shading

- Highlights on the surface are sometimes displayed with anomalous（不规则） shape, and the linear intensity interpolation can cause bright oe dark intensity streaks called Mach Band
- **Solution:** Lighting or other parameters are calculated in each vertex of the polygon, then in each point inside the polygon, the intensity of lighting is computed through bicentric interpolation between the vertices.

- Two main algorithms
  - Gouraud shading: interpolation of intensity,
  - Phong shading: Interpolation of normal,

Gouraud shading

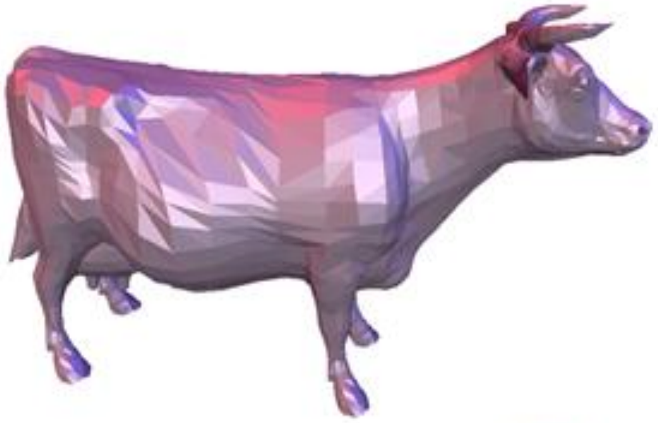
- Proposed by Gouraud in 1971
  - Also called Gouraud interpolation
- Computation
  - first calculate the color value of each vertex
  - Interpolate the color values calculated at each vertex

Phong shading

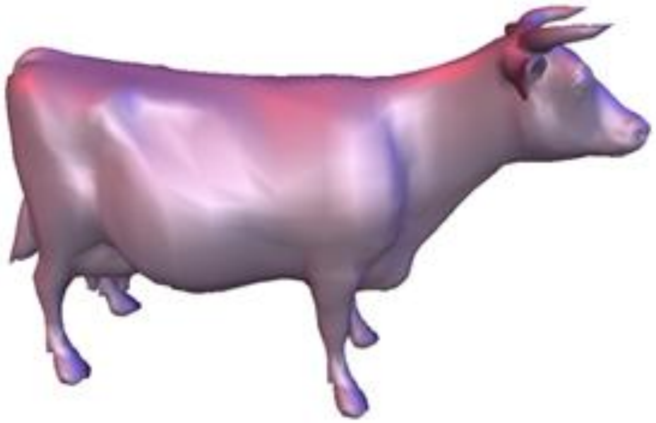
- Developed by Bui Tuong Phong in his 1973 PhD dissertation
  - Note: it is different from Phong lighting model
- Computation
  - Different from Gouraud shading, it does not interpolate color values, it interpolate surface normal, and use the interpolated normal to calculate the color value.

Comparison

only by lighting model



by Phong shading



Transformation (变换) and Viewing

- **Why Transformation and Viewing**
  - Computer graphics renders high-quality color images of a scene which is composed of 3D geometric models.
  - Transformations is extremely important in graphics, With them, you can position, reshape and animate objects, lights and cameras

Why transformation: an example

- **Suppose we already have a procedure to draw a unit square [0,1]*[0,1]**
- **If you want to write another procedure to draw a 2D rectangle with one corner at (lox, loy) and another diagonal corner at (hix, hiy).**

One Solution

- **Rewrite the drawing rectangle procedure**

```
drawRect(lox, loy, hix, hiy) {
        glBegin(GL_QUADS);
        glVertex2f(lox, loy);
        glVertex2f(hix, loy);
        glVertex2f(hix, hiy);
        glVertex2f(lox, hiy);
        glEnd();
}
```

- **Drawing a rectangle could be done like this, how about drawing a complex model such as a teapot?**

Solution using Transformation

```
drawRect(lox, loy, hix, hiy) {
     glTranslate(lox, loy);
     glScale(hix-lox, hiy-loy);
     drawUnitSquare(0,0,1,1);
}
```

- This is a solution you should use in almost all graphics application for better speed, modularity, flexibility, …

What is a Transformation(变换)?

- A function that maps a point *x* to another point *x*':

  Applications: Morphing, deformation, viewing, projection, real-time shadows, …

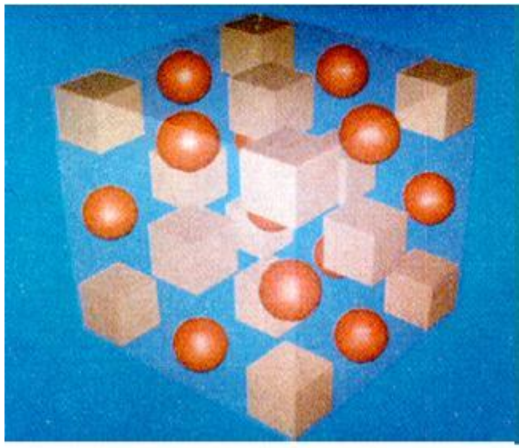


Fig 1. Undeformed Plastic

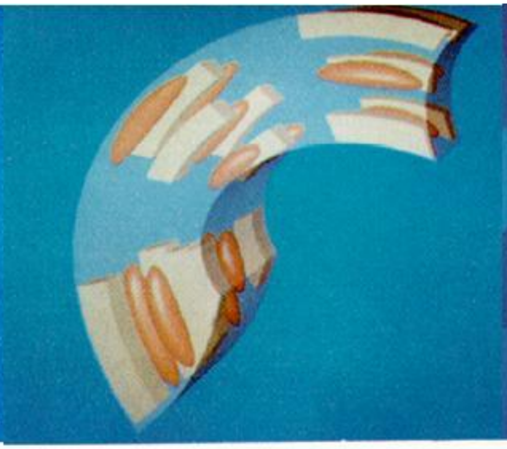
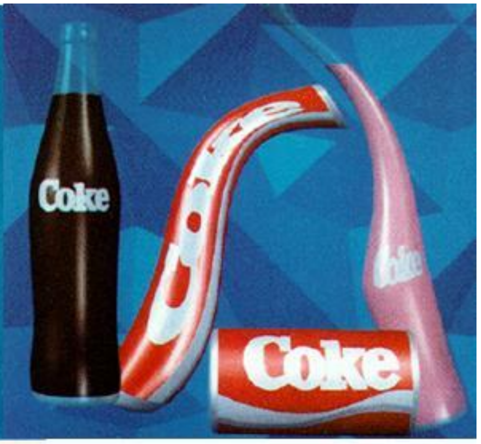
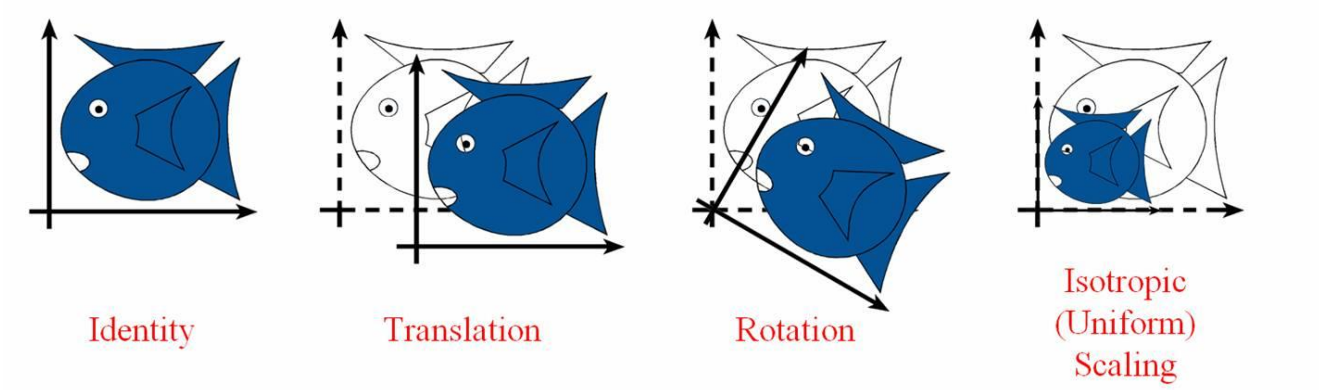


Fig 2. Deformed Plastic



Simple Transformations



- Let's see some intuitive transformation from left to right:
  - **Identity(不变) , translation(平移), rotation(旋转) and isotropic scaling(均衡缩放)**
- Transformations can be combined together
  - First rotate, then scale, then translate
- Transformations are invertible(可逆) operators

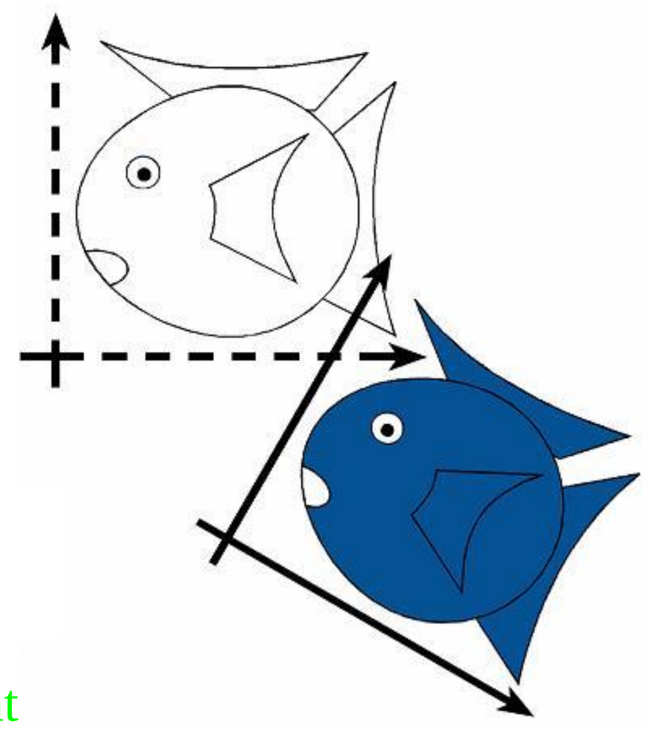
Transformation Classification(分类)

- Transformations could be classified into these categories:
  - Rigid-body Transformation (刚体变换)
  - Similarity Transformation(相似变换)
  - Linear Transformation(线性变换)
  - Affine Transformation(仿射变换)
  - Projective Transformation(投影变换)

Rigid-body Transformation

- Preserve distance, angle and size

- Include
  - Identity
  - translation
  - Rotation
  - and combinations of them
- Isotropic Scaling is not rigid body transformat

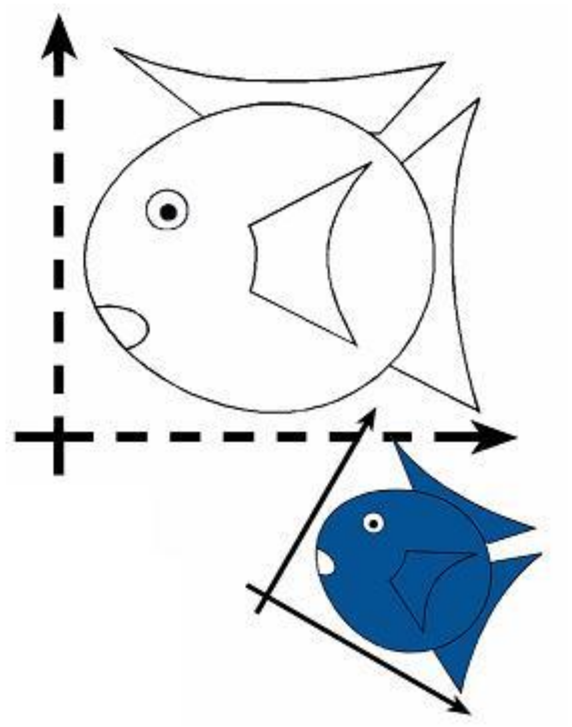


Similarity Transformation

- Preserve angle

- Include
  - Identity
  - Translation
  - Rotation
  - Isotropic Scaling

- not include general scaling

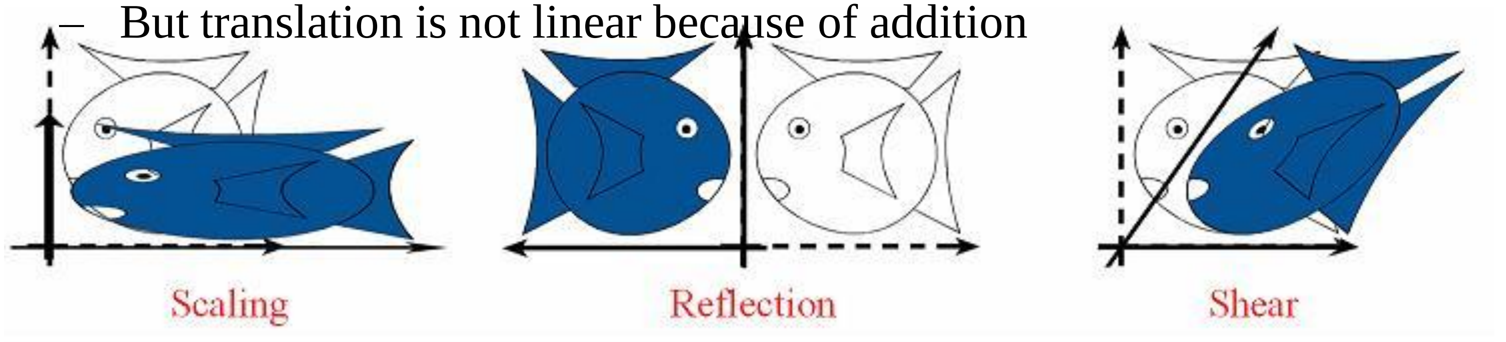


Linear Transformation

- Preserve linearity of addition and scalar multiplication :

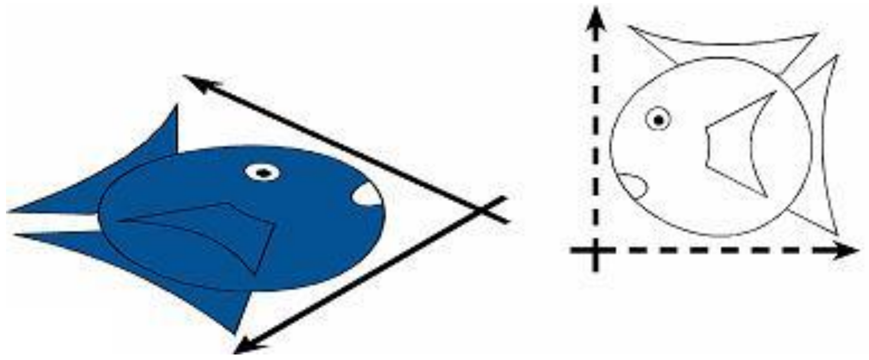
  L(p+q) = L(p) + L(q) **aL(p) = L(ap)**

- Include
  - **Identity, Rotation, Scaling**
  - **Reflection(对称), Shear(切变、剪切)**
  - Shear in the x direction, pull the top to the right and the bottom to the left
  - But translation is not linear because of addition



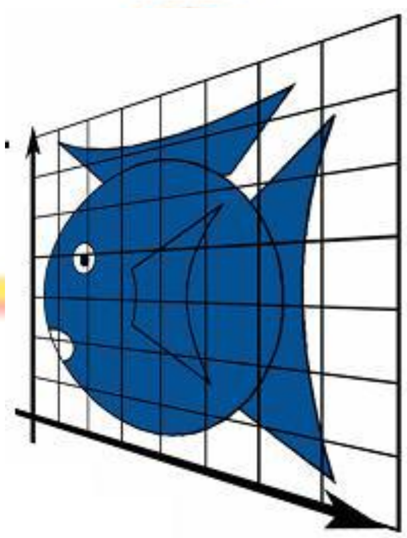
Affine Transformation

- Preserve parallel lines
  - If two lines are parallel before transformation, they are still parallel after transformation

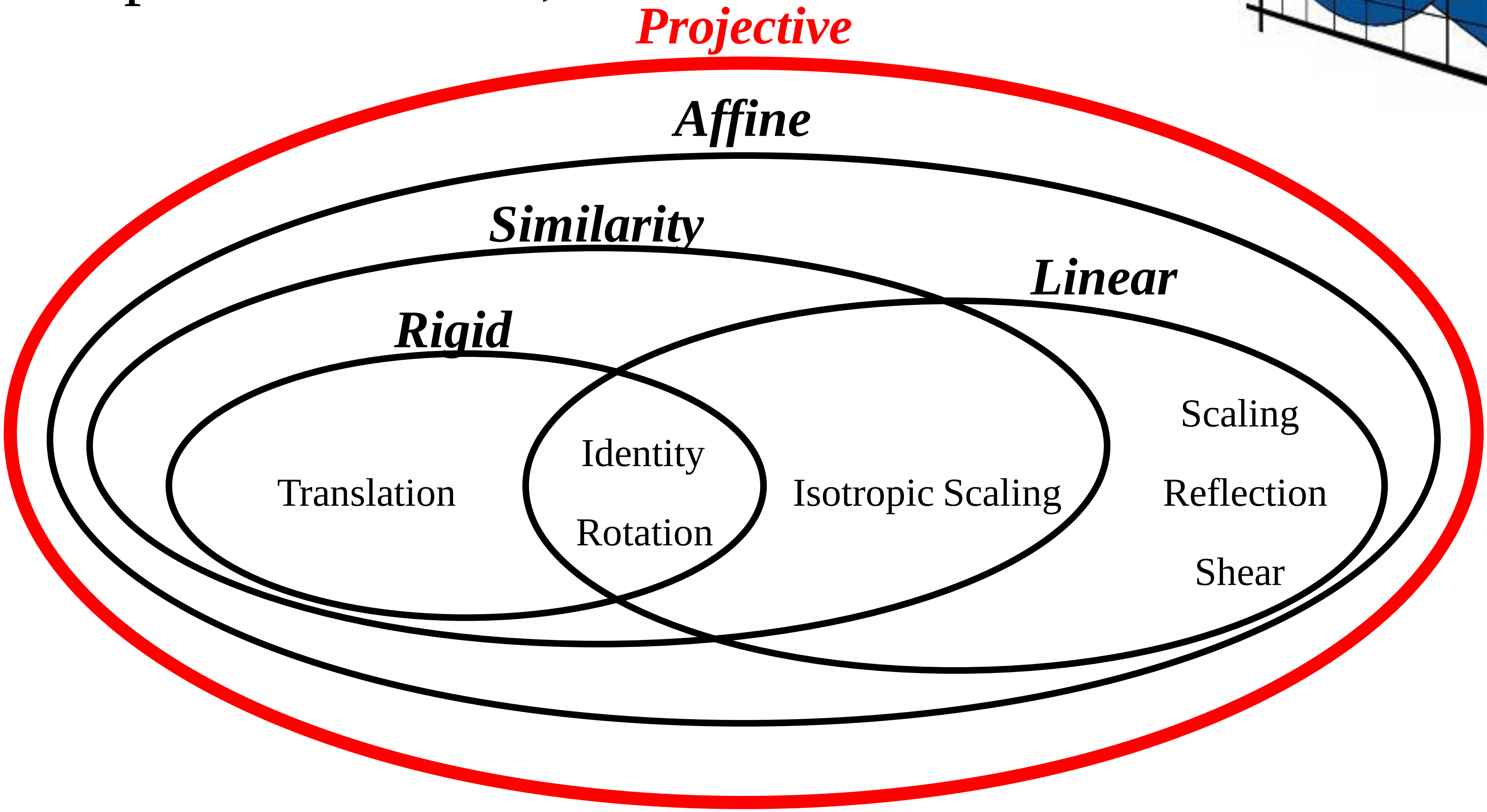


- Include
  - linear transformation
  - similarity transformation

Projective Transformation



- preserves lines, produce projective visual effects



Transformation Representation

- Take 2D as an example:

$$x' = ax + by + c \qquad (1)$$

$$y' = dx + ey + f \qquad (2)$$

- Represent it in vector-matrix form:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} a & b \\ d & e \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{bmatrix} c \\ f \end{bmatrix}$$

$$p' = Mp + t$$

- **Need 2 variables for transformation: M and t**

$$p' = M p + t$$

- if representing *p*, *p'* in homogenous coordinates （齐次坐标） The formula is rewritten as:

$$p' = M p$$

•What’s homogenous coordinates?

Homogeneous Coordinates(齐次坐标)

- The objective of HC is to use a 4D column matrices to represent both points and vectors in 3D.

Formulation in E^2

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} a & b \\ d & e \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{bmatrix} c \\ f \end{bmatrix}$$

$$p' = M p + t$$

Homogeneous formulation

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

$$p' = M p$$

Homogeneous Coordinates(齐次坐标)

- Add an extra dimension
  - in 2D, we use 3 x 3 matrices
  - In 3D, we use 4 x 4 matrices
- Each point has an extra value, *w*

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

$$p' = \color{red}{M}\, p$$

Homogeneous Coordinates

- Most of the time w = 1, and we can ignore it

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

- If we multiply a homogeneous coordinate by an *affine matrix*, w is unchanged; by an projective matrix, the value of w will change

Homogeneous Visualization

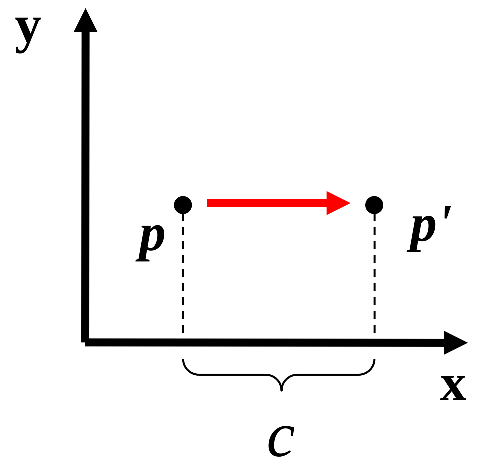
- If $w \neq 1$, can be divided by w to normalize(归一化) (homogenize)
  - So (2x,2y,2z,2) is equal to (x,y,z,1)
- w = 0?
  - this case, the homogenous coordinate represent a direction (方向) but not a point.

Translate (t_x, t_y, t_z)

- Why we introduce the extra dimension?

  Now translations can be encoded in the matrix!

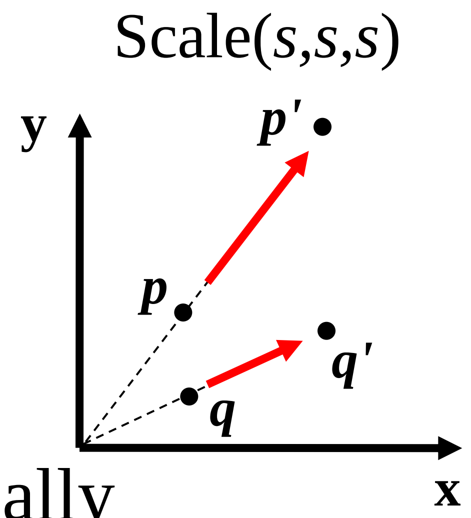
Translate(c,0,0)



unified representation of transformation

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Scale (s_x, s_y, s_z)

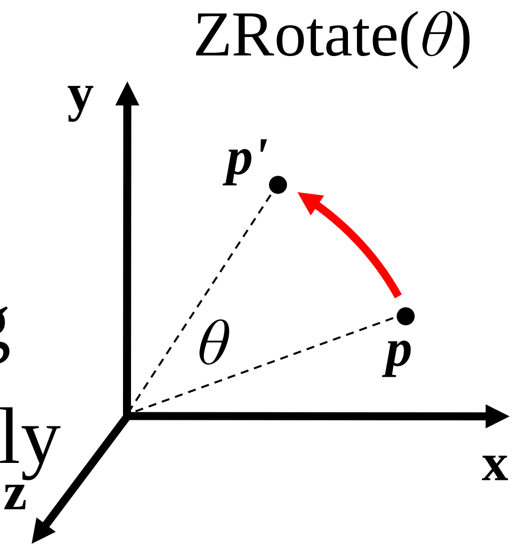


- Scale
  - For scaling, a fixed point (usually origin) is unchanged by the transformation
  - Such scaling matrix with a fixed point of origin allows independent scaling along the coordinate axes

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Rotation

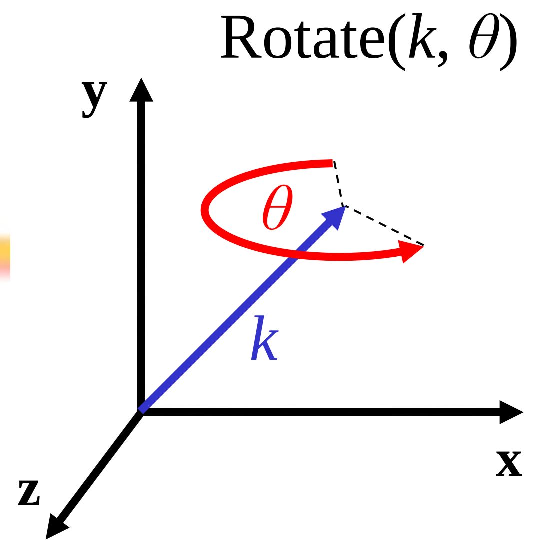
- There are 3 freedom corresponding to our ability to rotate independently about the three coordinate axes.
- About z axis



$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Demo

Rotation



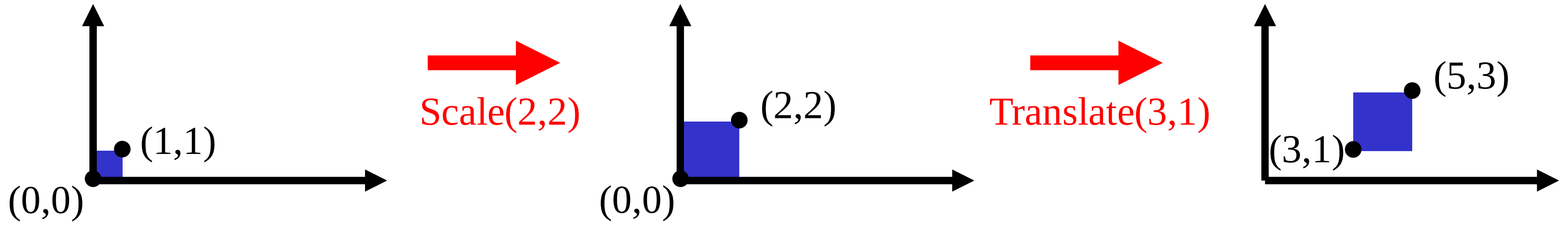
- About (k_x, k_y, k_z), a unit vector on an arbitrary axis (Rodrigues Formula)

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} k_xk_x(1-c)+c & k_zk_x(1-c)-k_zs & k_xk_z(1-c)+k_ys & 0 \\ k_yk_x(1-c)+k_zs & k_zk_x(1-c)+c & k_yk_z(1-c)-k_xs & 0 \\ k_zk_x(1-c)-k_ys & k_zk_x(1-c)-k_xs & k_zk_z(1-c)+c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

where $c = \cos\theta$ & $s = \sin\theta$

Transformation Combination(合并)

An Example: Scale then Translate



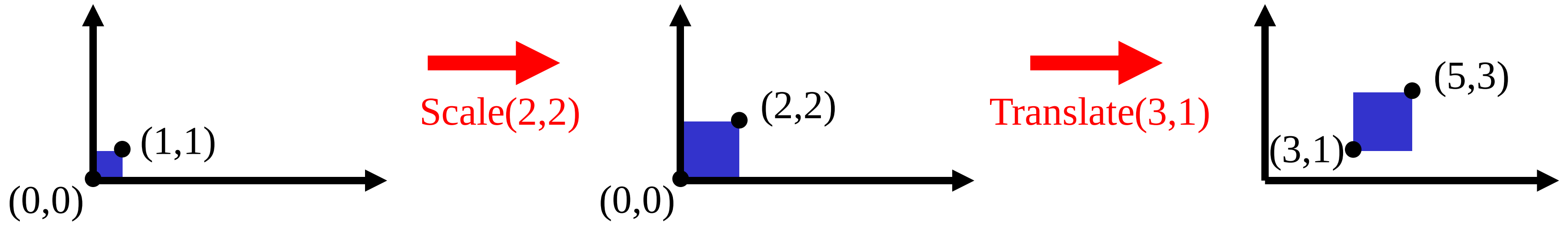
Use matrix multiplication: p' = T (S p) = TS p

$$TS = \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 3 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

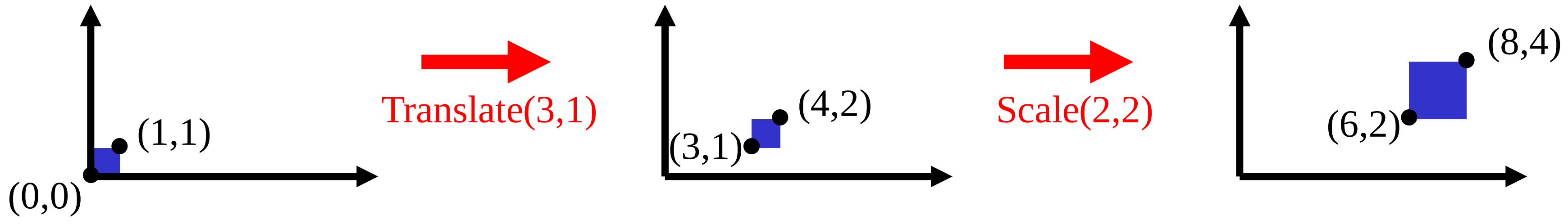
Notice: matrix multiplication is inverse

Non commutative(交换) Composition

Scale then Translate: p' = T (S p) = TS p



Translate then Scale: p' = S (T p) = ST p



Non-commutative Composition

The reason is because matrix multiplication is non-commutative TS $\neq$ ST

Scale then Translate: p' = T (S p) = TS p

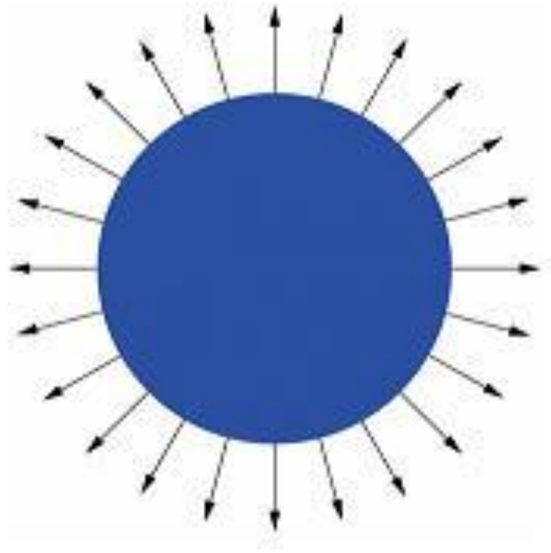
$$TS = \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 3 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

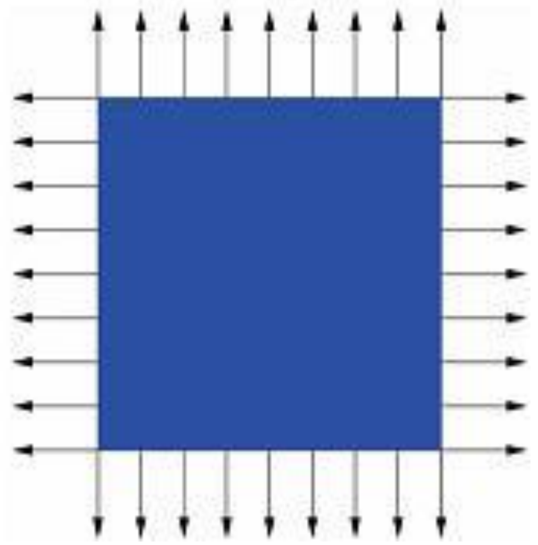
Translate then Scale: p' = S (T p) = ST p

$$ST = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 6 \\ 0 & 2 & 2 \\ 0 & 0 & 1 \end{pmatrix}$$

Transformation of Normals(法向)

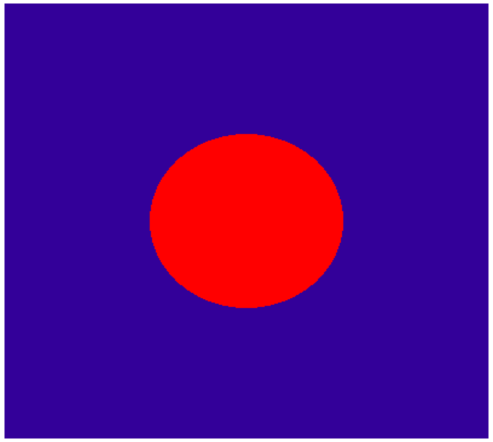
- Surface Normal: unit vector that is locally perpendicular to the surface
- One of the most important geometric property of surfaces

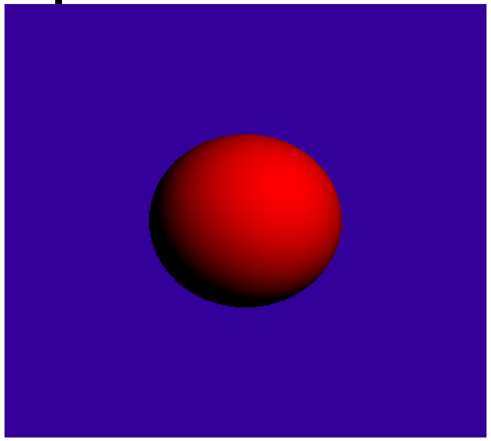




Why is the Normal important?

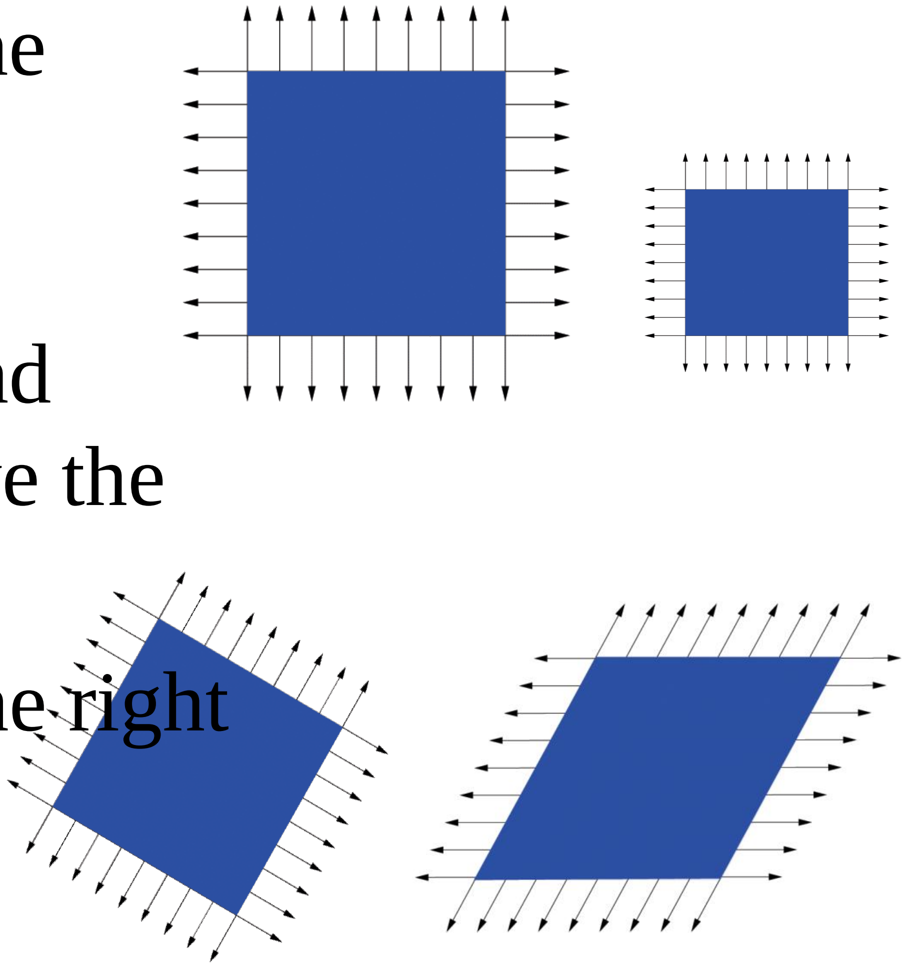
- It's used for shading — makes things look 3D!
- All kinds of lighting models utilized surface normal in the lighting computation


object color only


Diffuse Shading

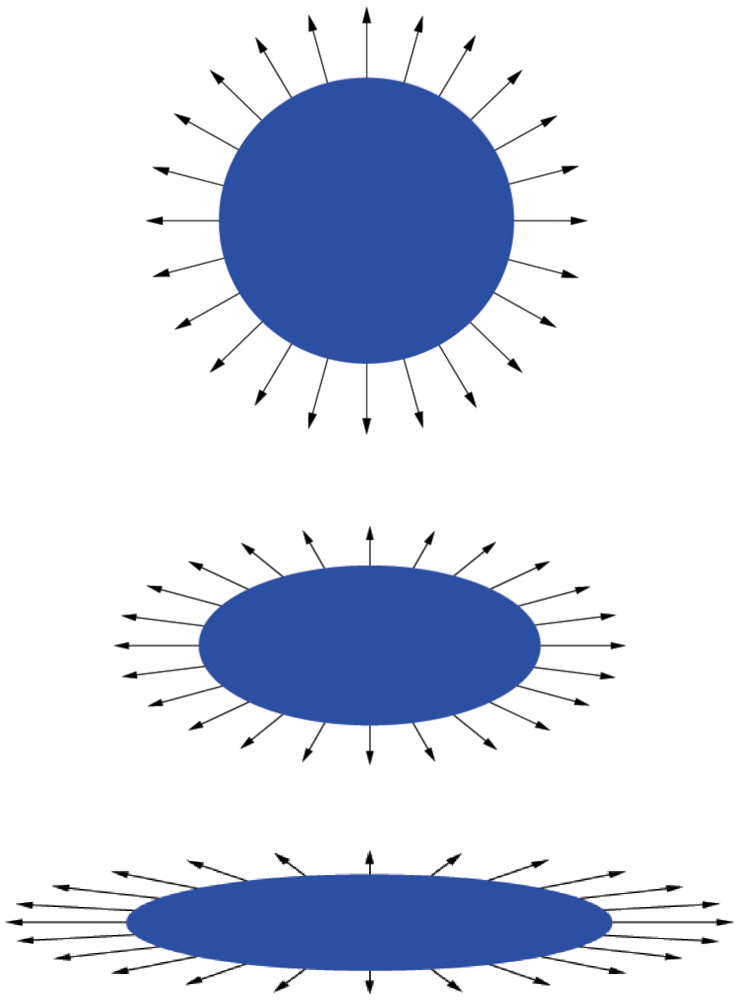
Transform Normal like Object?

- Could we transform the normal like objects?
- We could find that translation, rotation and isotropic scale preserve the correct normal.
- But shear does not (the right bottom image.)



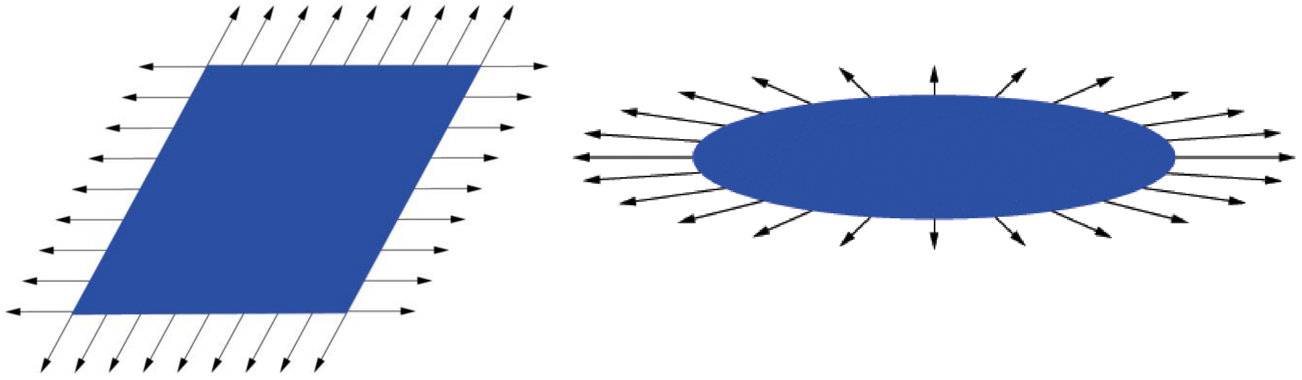
Transform Normal like Object?

- Another example
  - **We scale an sphere (but not isotropic scale).**
  - **The transformed normal is also incorrect.**

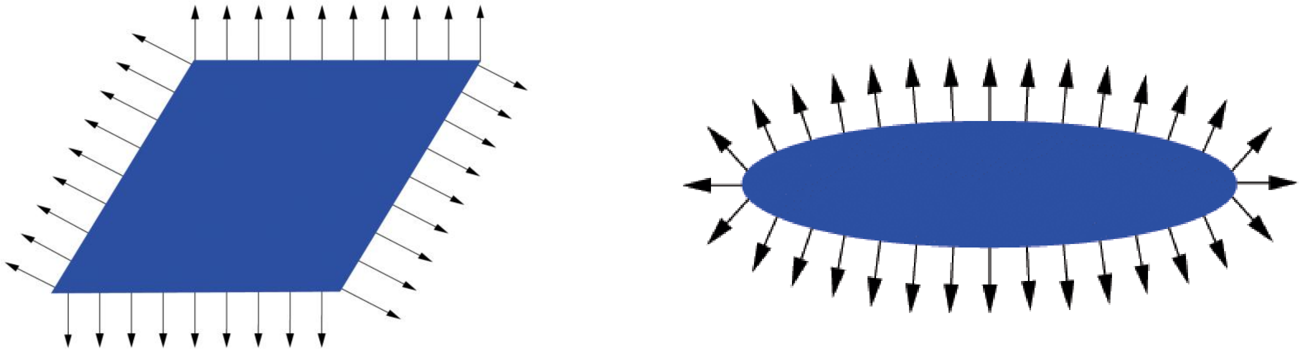


Transformation for shear and scale

Incorrect Normal Transformation



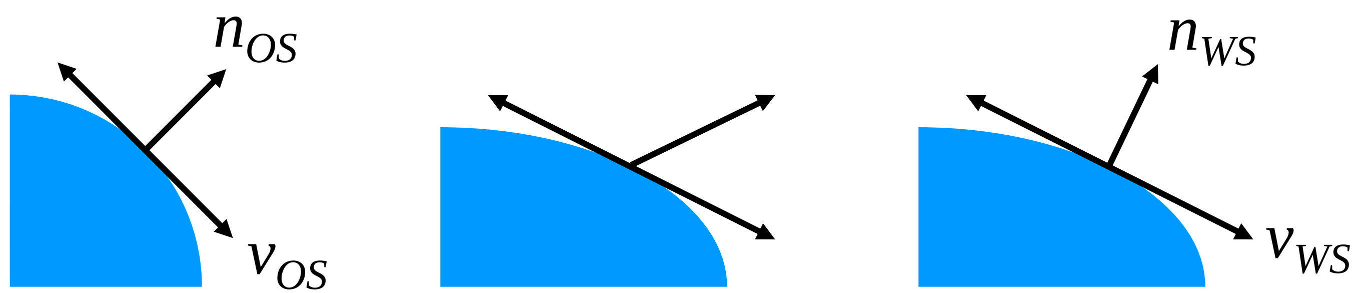
Correct Normal Transformation



how could we transform normal correctly?

How to transform normal?

- transforming the *tangent plane* (切平面) to the normal, not the normal *vector directly*



Original Incorrect Correct

Pick any vector v_{OS} in the tangent plane to transform

$$v_{WS} = \mathbf{M} \; v_{OS}$$

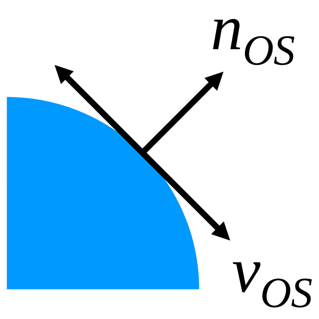
Transform tangent vector *v*
v is perpendicular to normal *n*:

Dot product $n_{OS}^{\mathbf{T}}\, v_{OS} = 0$

$$n_{OS}^{\mathbf{T}}\ (\mathbf{M}^{-1}\ \mathbf{M})\ v_{OS} = 0$$

$$(n_{OS}^{\mathbf{T}}\ \mathbf{M}^{-1})\ (\mathbf{M}\ \ v_{OS}) = 0$$

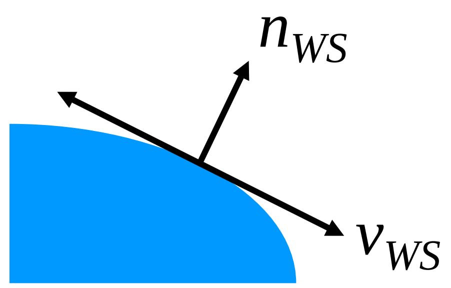
$$(n_{OS}^{\mathbf{T}}\ \mathbf{M}^{-1})\ v_{WS} = 0$$



v_{WS} is perpendicular to normal n_{WS}:

$$n_{WS}^{\mathbf{T}} = n_{OS}^{\mathbf{T}}\, (\mathbf{M}^{-1})$$

$$\boxed{n_{WS} = (\mathbf{M}^{-1})^{\mathbf{T}}\, n_{OS}}$$



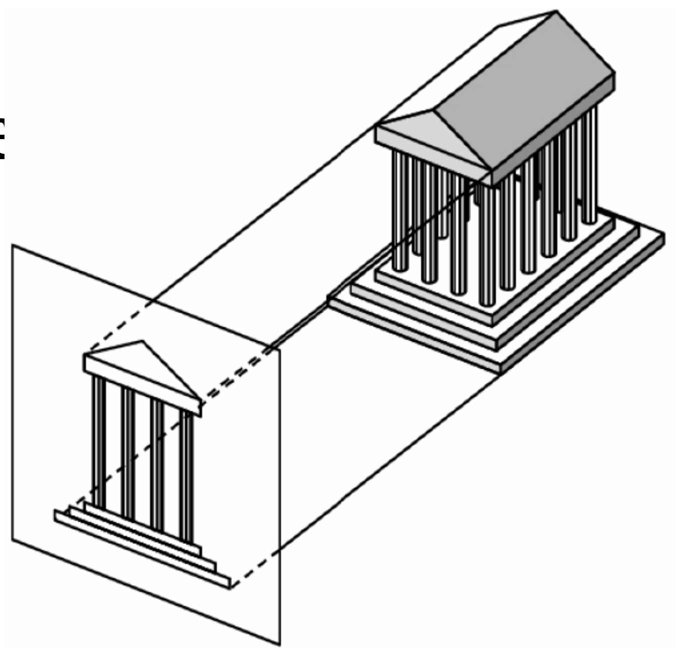
The matrix is the transpose of inverse of M

Viewing and Projection

- **Our eyes could collapse 3D world to 2D image (brain then reconstructs 3D)**
- **In computer graphics, we do it by projection**
- **two parts:**
  - **viewing transformation: camera position and orientation**
  - **perspective/orthographic(透视/投影): reduces 3D to 2D**

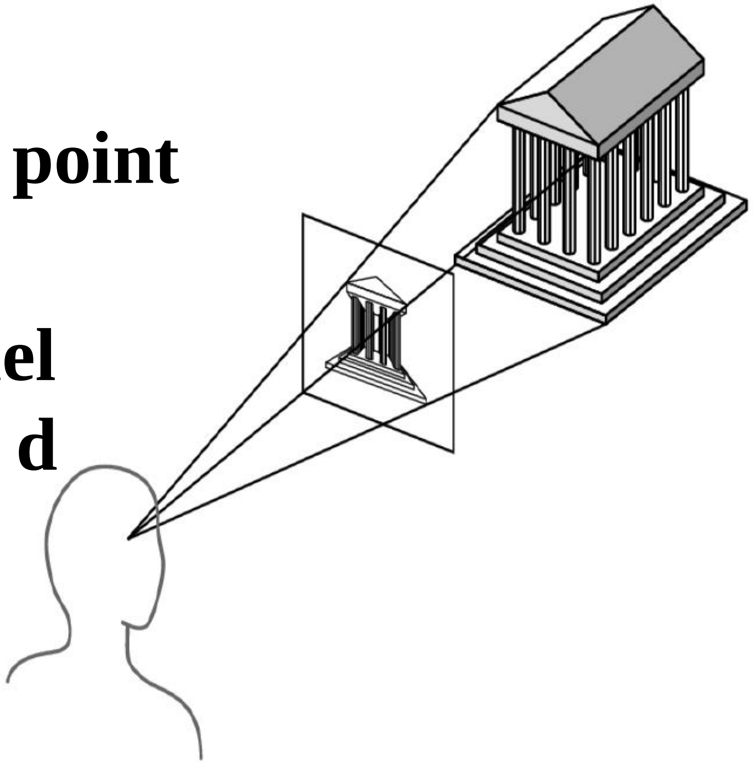
Orthographic Projection(正交投影)

- **When the focal point is at infinity, the rays are parallel and orthogonal to the image plane**
- **No perspective effects**
- **When xy-plane is the image plane,**
  (x,y,z) -> (x,y,0) ,
  orthographic projection



A Simple Perspective Projection (透视投影)

- **Focus point is at finite distance**
- **Perspective effects**
- **When**
  - **the camera is at the origin point and looks along the z-axis**
  - **The image plane is paraellel to the xy-plane at distance d**
  - **$(x,y,z) \rightarrow ((d/z)x, (d/z)y, d)$**



A Perspective Projection Matrix

- **Projection using homogenous coordinates:**
  - **Transform (x,y,z) -> ((d/z)x, (d/z)y,d)**

$$\begin{pmatrix} dx \\ dy \\ dz \\ z \end{pmatrix} = \begin{pmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & d & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

  - **Divide by the 4-th coordinate (the “w” coordinate)**

Thanks!