计算机图形学

第十章 光照绘制技术

华东师范大学计算机学院

王长波　教授

提出问题

- **真实感场景绘制：**
  - 计算机图形输出设备上绘制出能够以假乱真的景象。

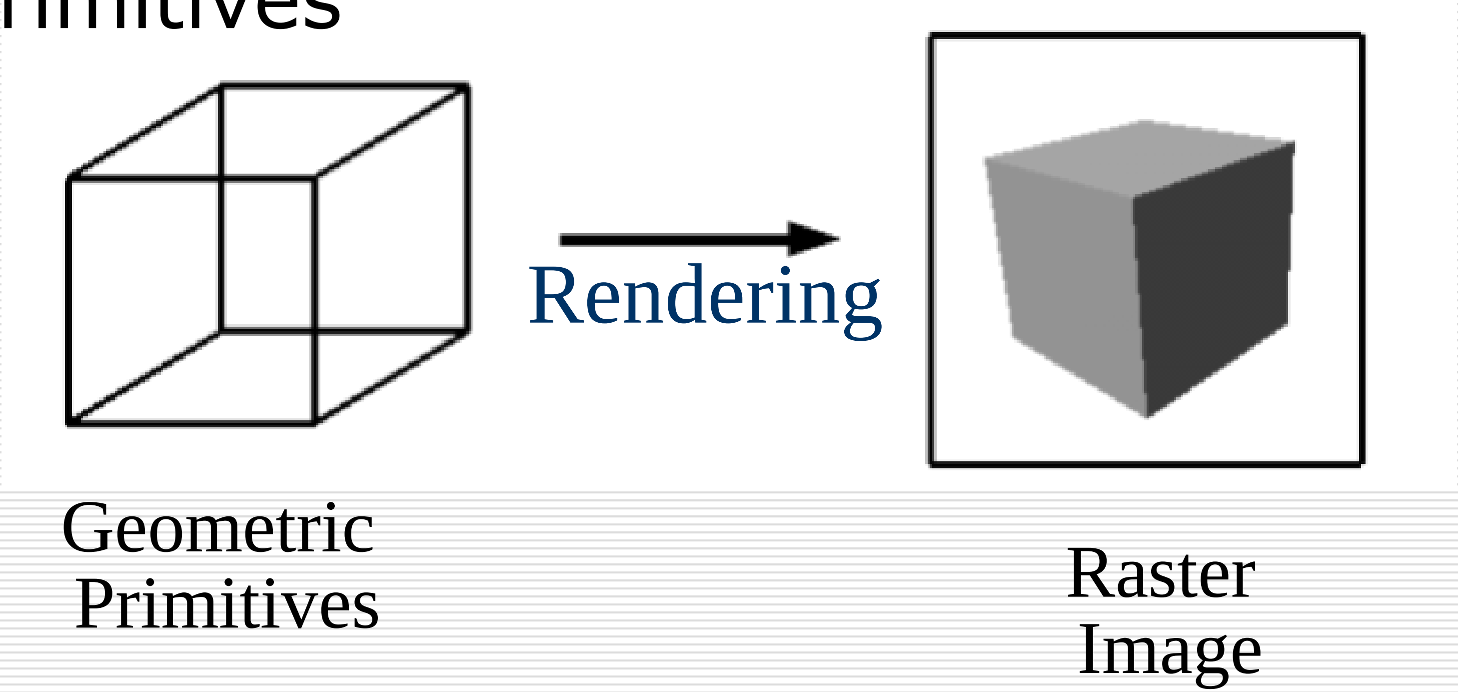
How graphics generate pretty pictures?

纲 要

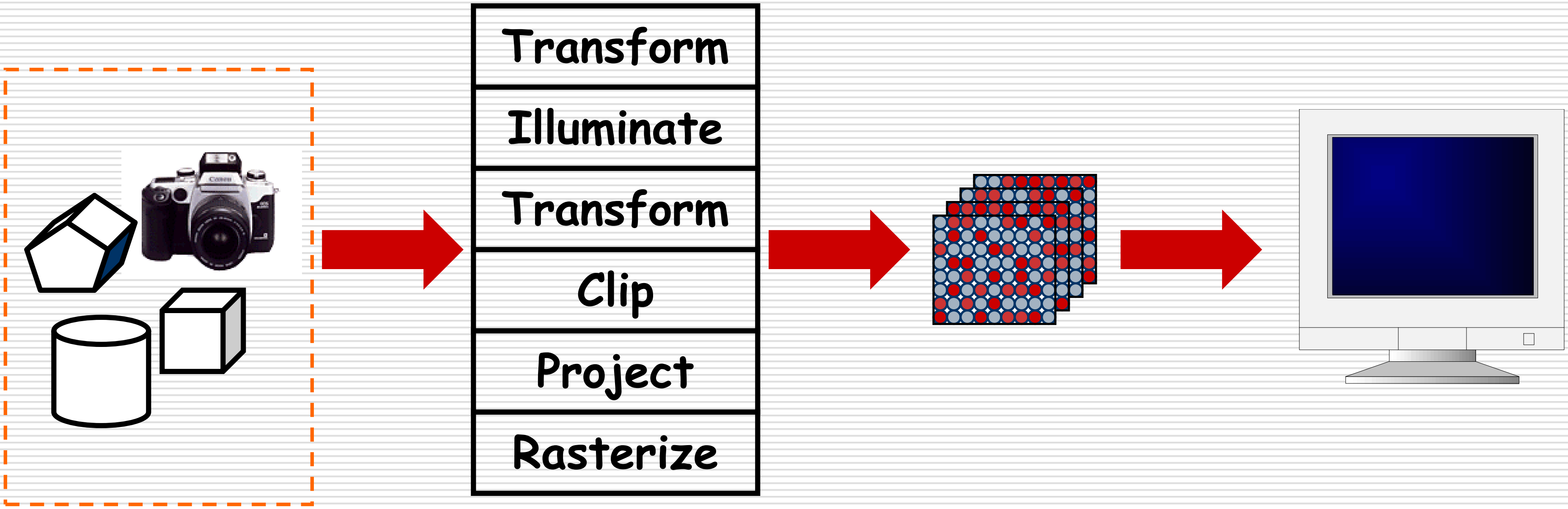
- 场景绘制流水线
- 颜色成像原理
- 光照模型,灯光设置
- D3D程序实现

Rendering

- Generate an image from geometric primitives



Rendering 3D Scenes



Model & Camera Parameters

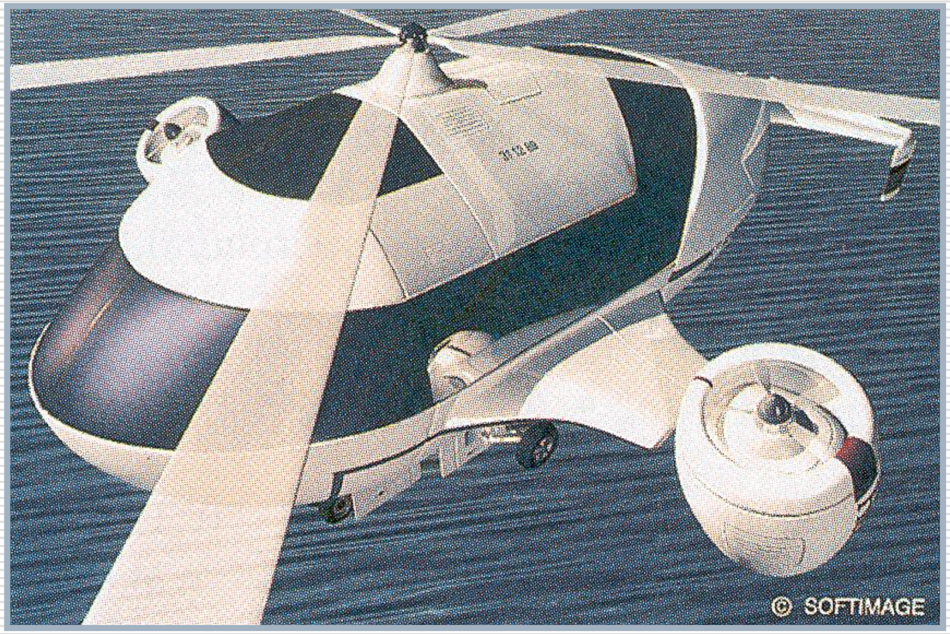
Rendering Pipeline

Framebuffer

Display

3D Model

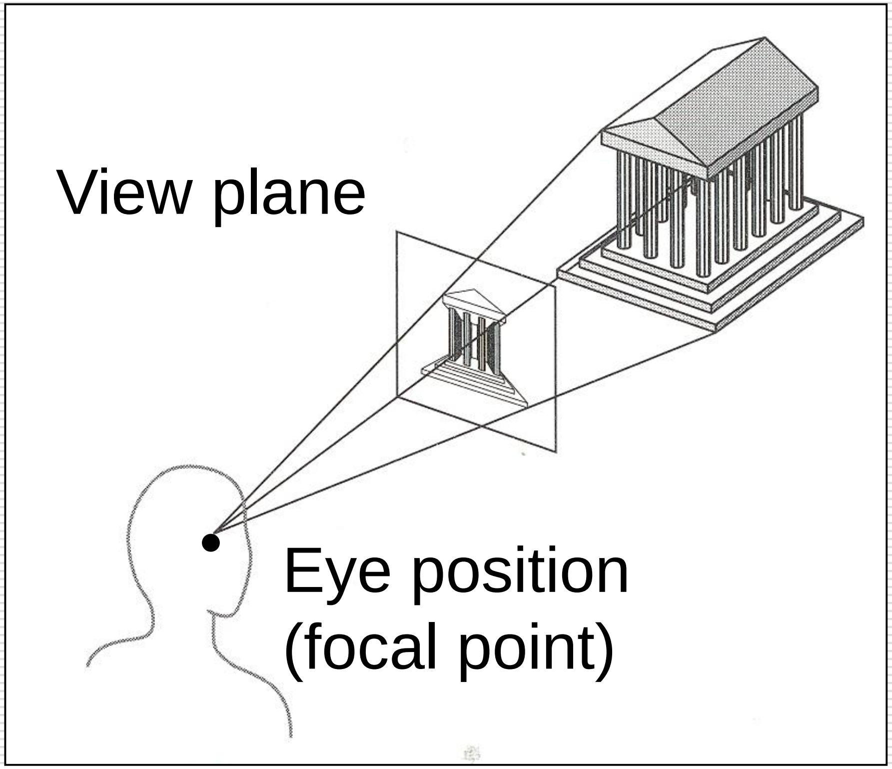
- 3D Geometric Primitives
  - Point
  - Line segment
  - Polygon
  - Polyhedron
  - Curved surface
  - Solid object
  - etc.

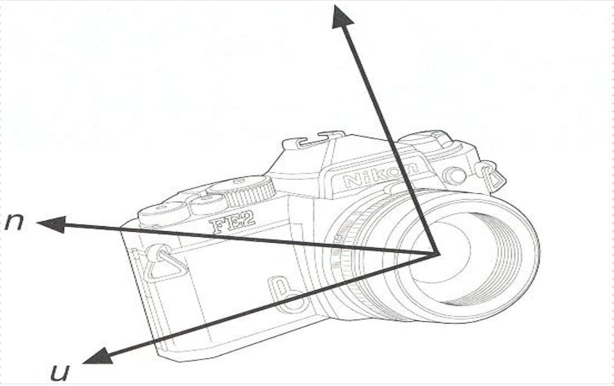


H&B Figure 10.46

Camera Models

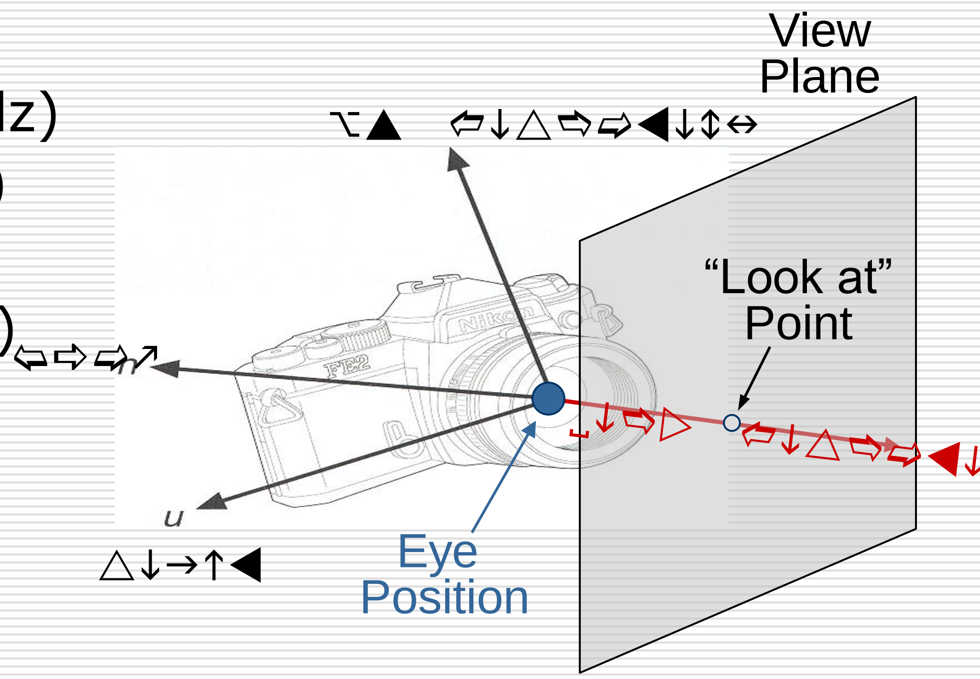
- Camera is as man's eye:



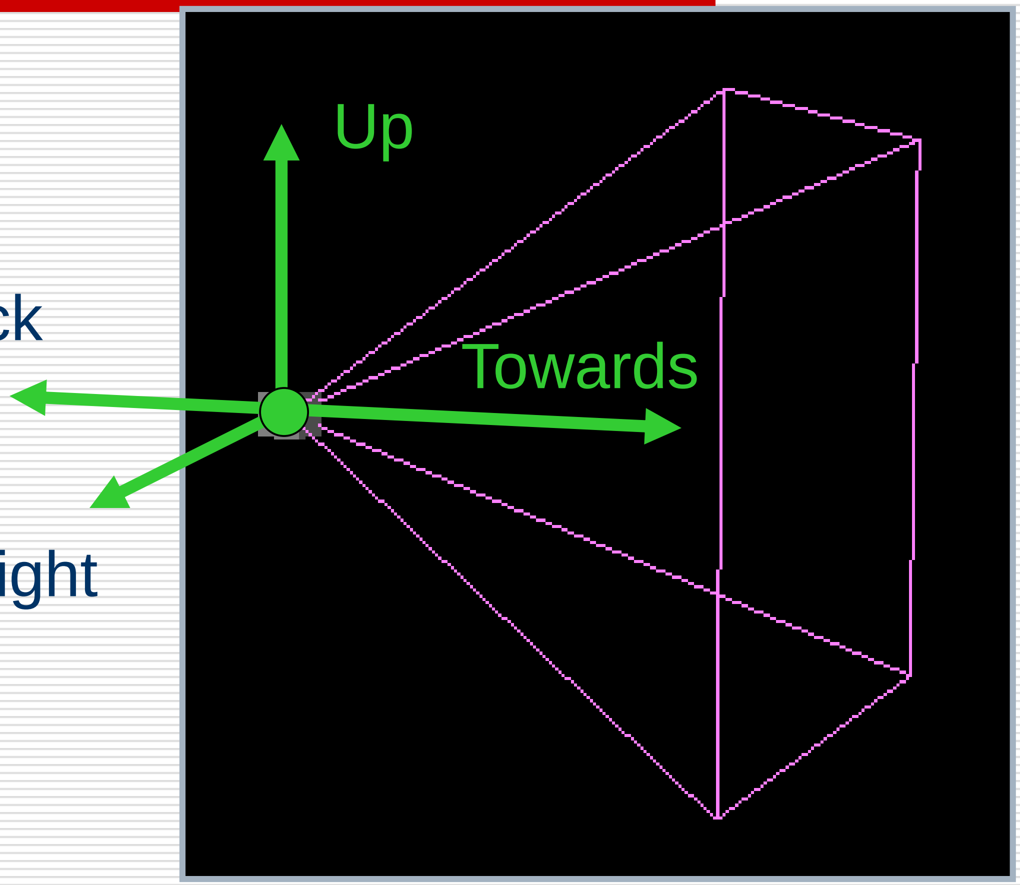


Camera Parameters

- Position
  - Eye position (px, py, pz)
- Orientation
  - View direction (dx, dy, dz)
  - Up direction (ux, uy, uz)
- Aperture
  - Field of view (xfov, yfov)
- Film plane
  - “Look at” point
  - View plane normal

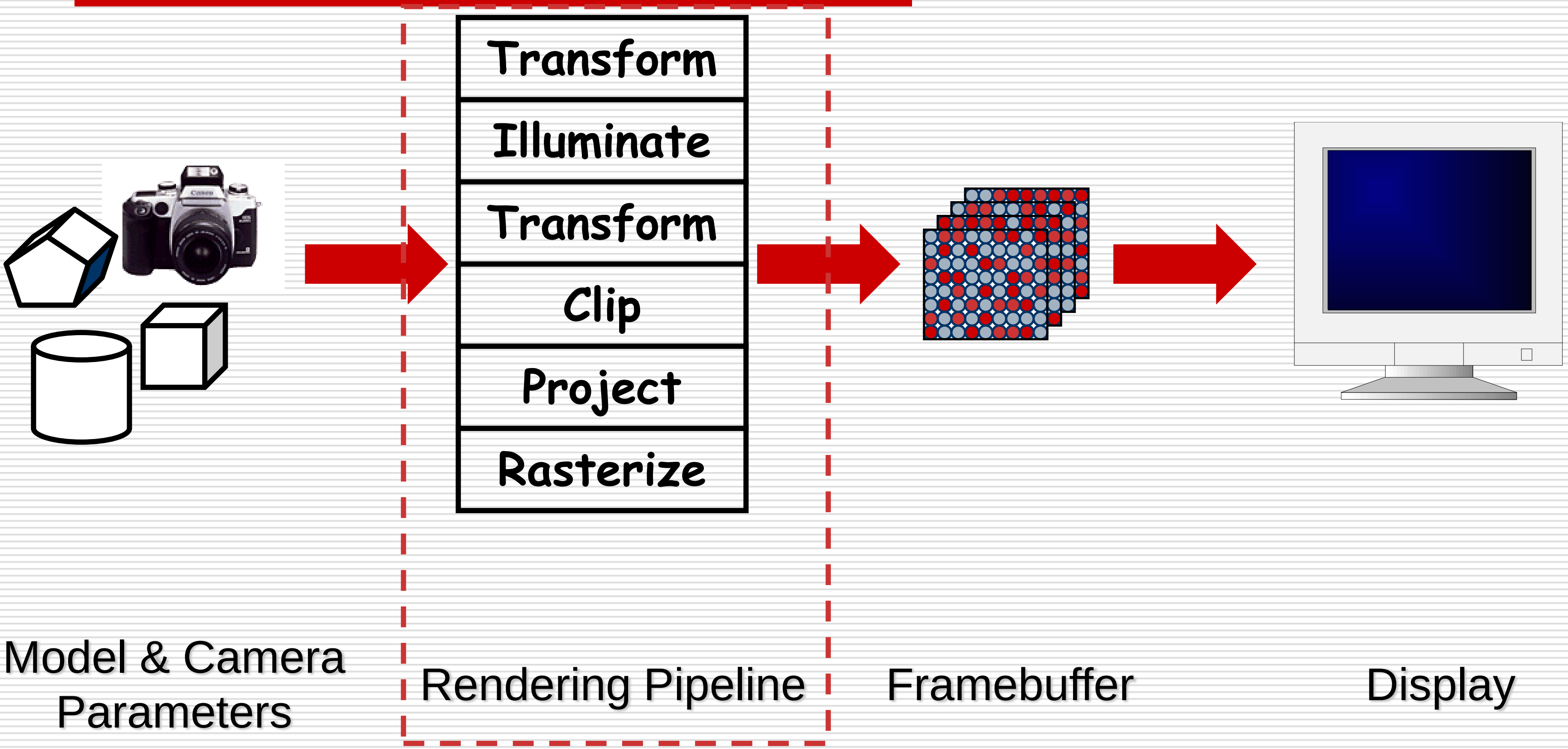


Moving the camera



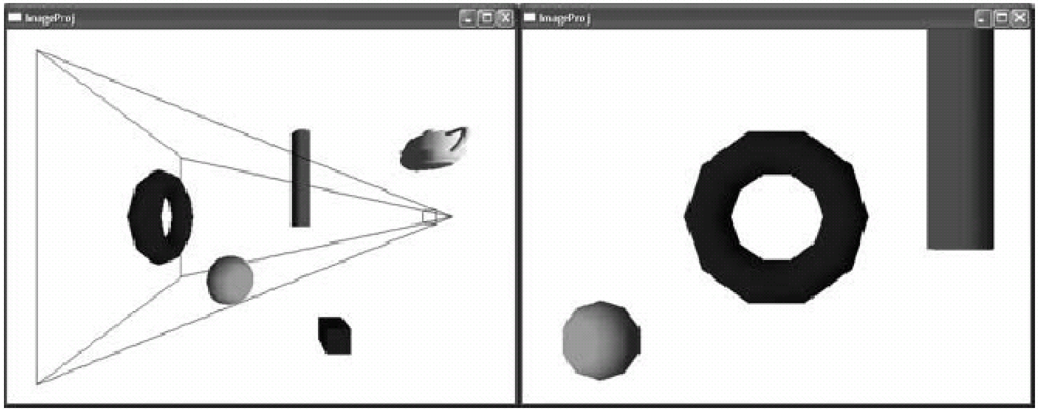
View Frustum

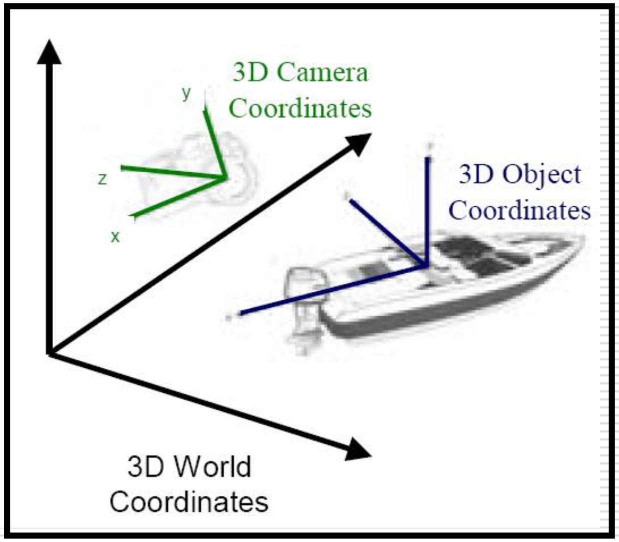
The Rendering Pipeline



Rendering: Transformations

- 3D scene → 2D image .





- They are used in three ways:
  - *Modeling transforms*
  - *Viewing transforms (Move the camera)*
  - *Projection transforms (Change the type of camera)*

The Rendering Pipeline: scene

Scene graph
Object geometry

Modeling Transforms

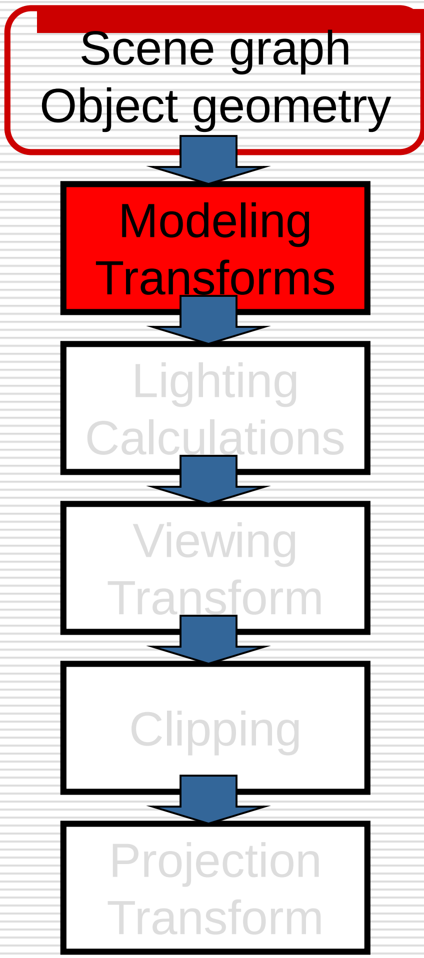
Lighting Calculations

Viewing Transform

Clipping

Projection Transform

The Rendering Pipeline: model transforms



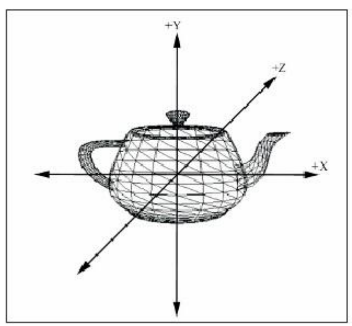
Objective:

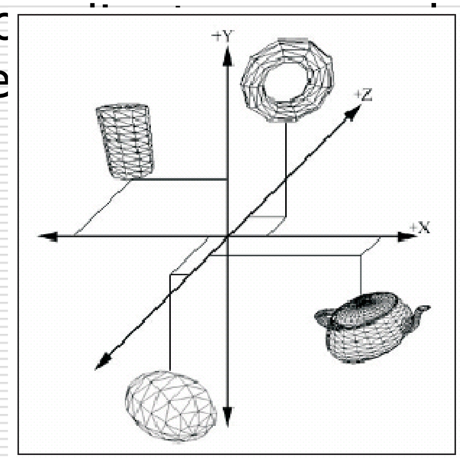
•All vertices of scene in shared 3-D “world” coordinate system

• Modeling transforms

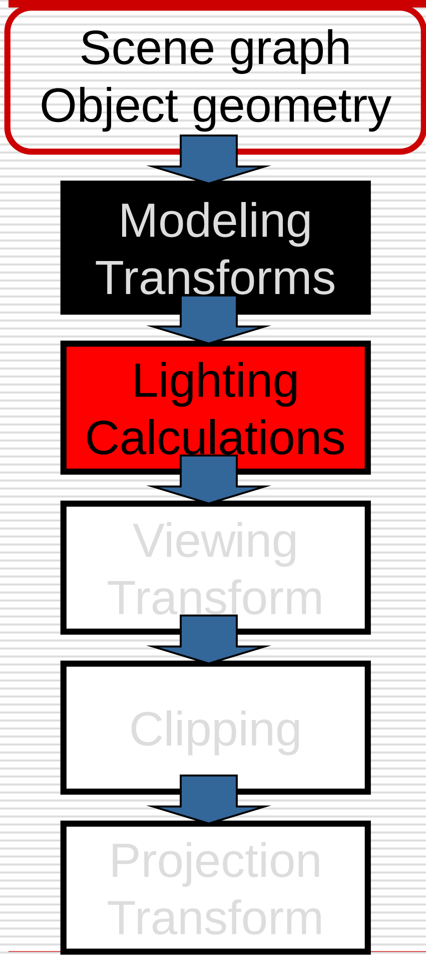
Size, place, scale, and rotate objects and parts of the model

• Object co
coordinate





The Rendering Pipeline: lighting calculations

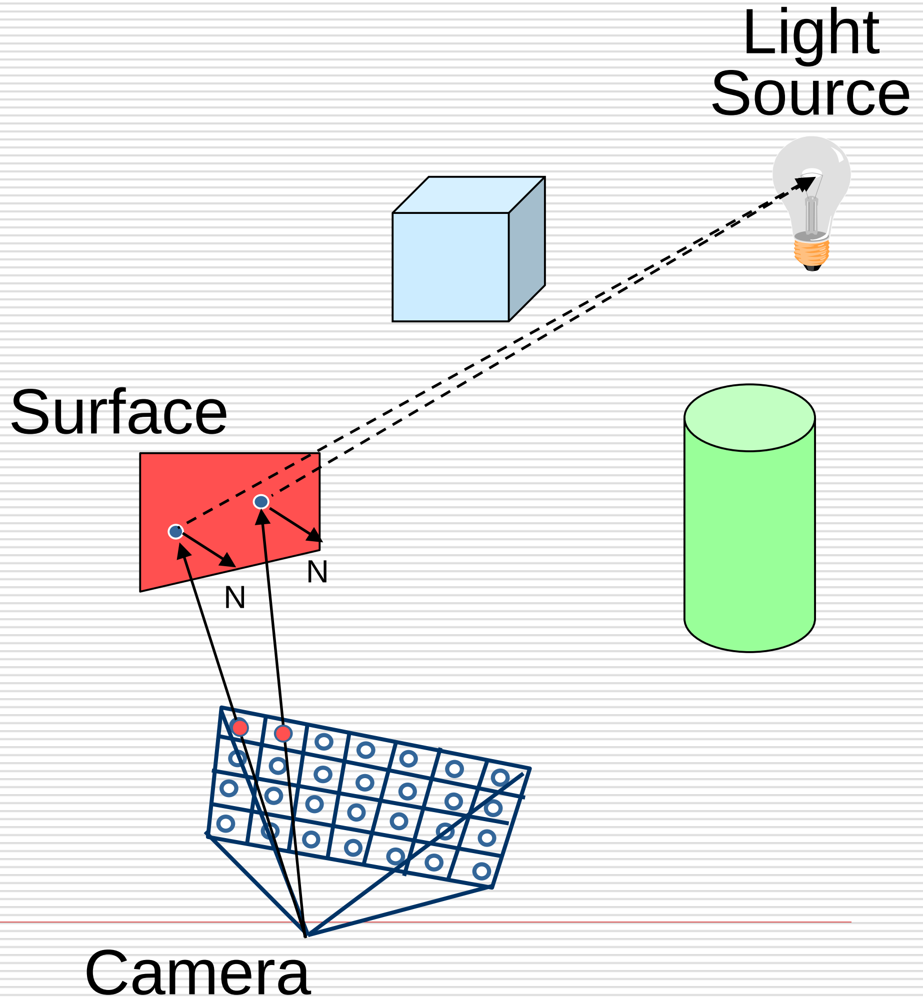


Objective :

- All geometric primitives are illuminated

Lighting Simulation

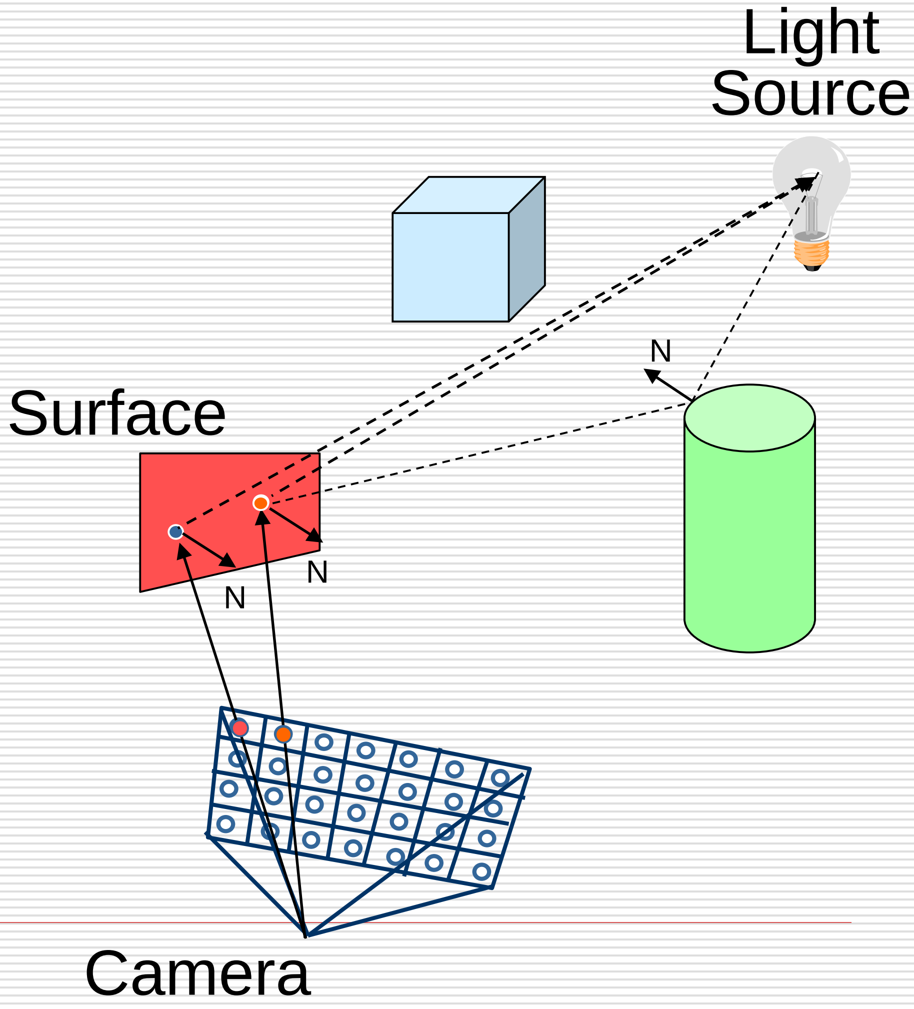
- Lighting parameters
  - Light source emission
  - Surface reflectance
  - Atmospheric attenuation
  - Camera response



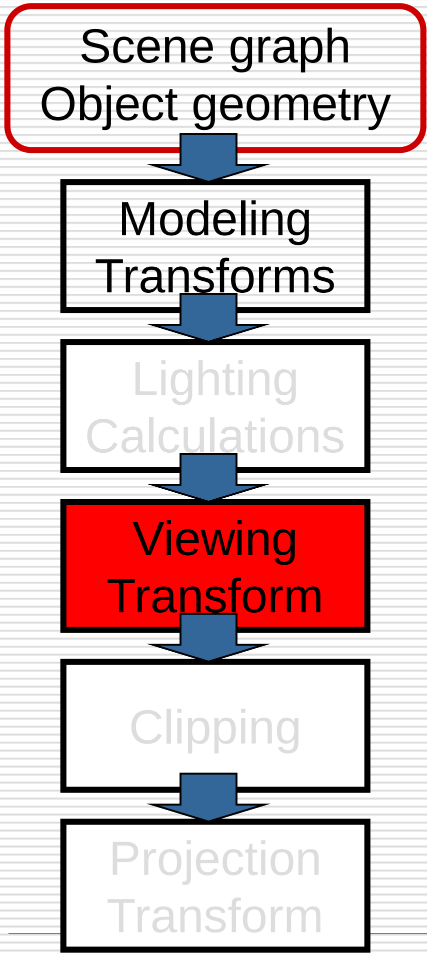
Lighting Simulation

- Direct illumination
  - Ray casting
  - Polygon shading
- Global illumination
  - Ray tracing
  - Monte Carlo methods
  - Radiosity methods

More on these methods later!



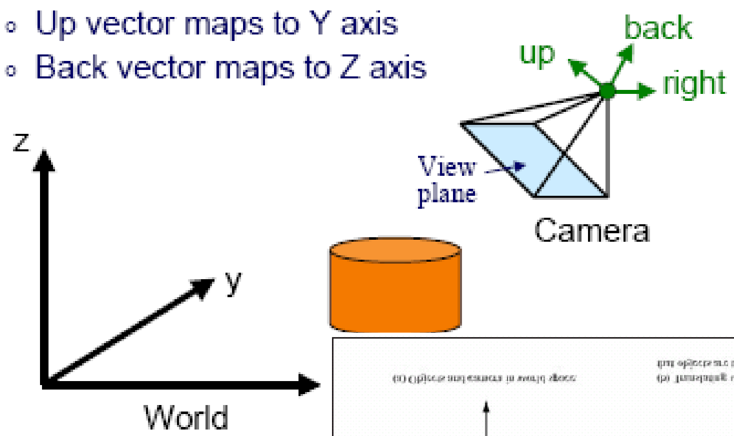
The Rendering Pipeline: viewing transform

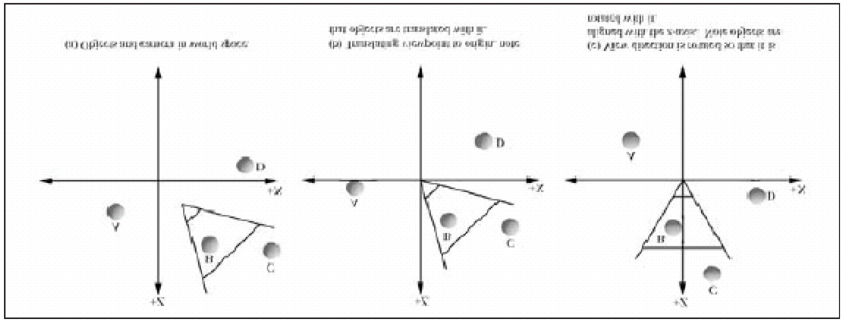


Objective :

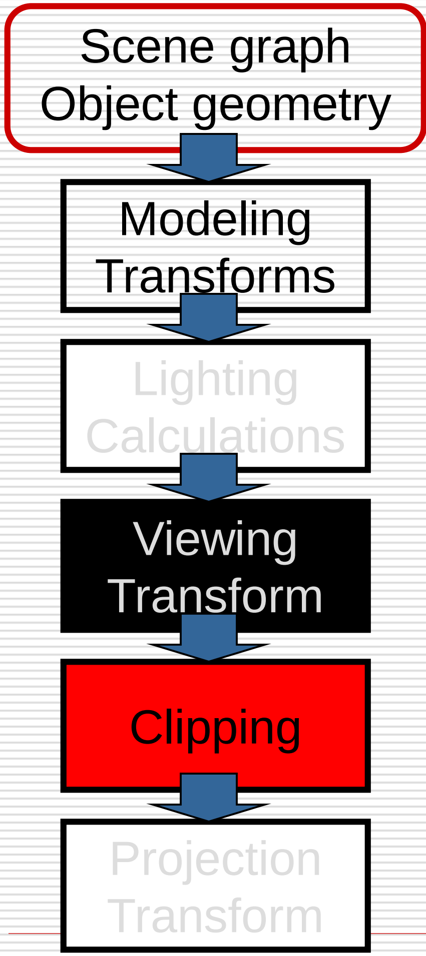
- Scene vertices in 3-D "view" or "camera" coordinate system

- Mapping from world to camera coordinates
  - Eye position maps to origin
  - Right vector maps to X axis
  - Up vector maps to Y axis
  - Back vector maps to Z axis



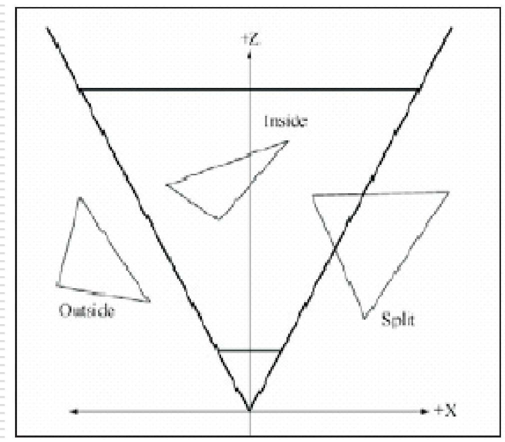


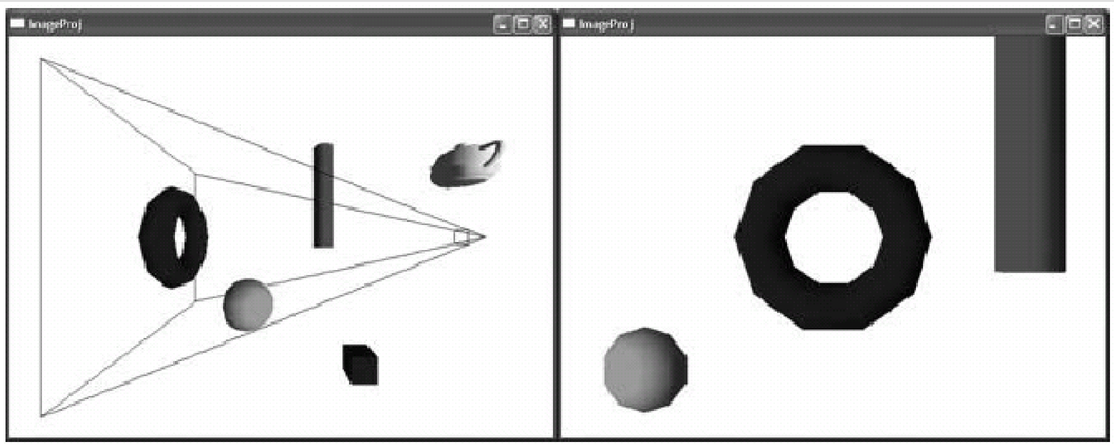
The Rendering Pipeline: Clipping



Objective :

- Remove geometry that is out of view





The Rendering Pipeline: 3-D

Scene graph
Object geometry

Modeling Transforms

Lighting Calculations

Viewing Transform

Clipping

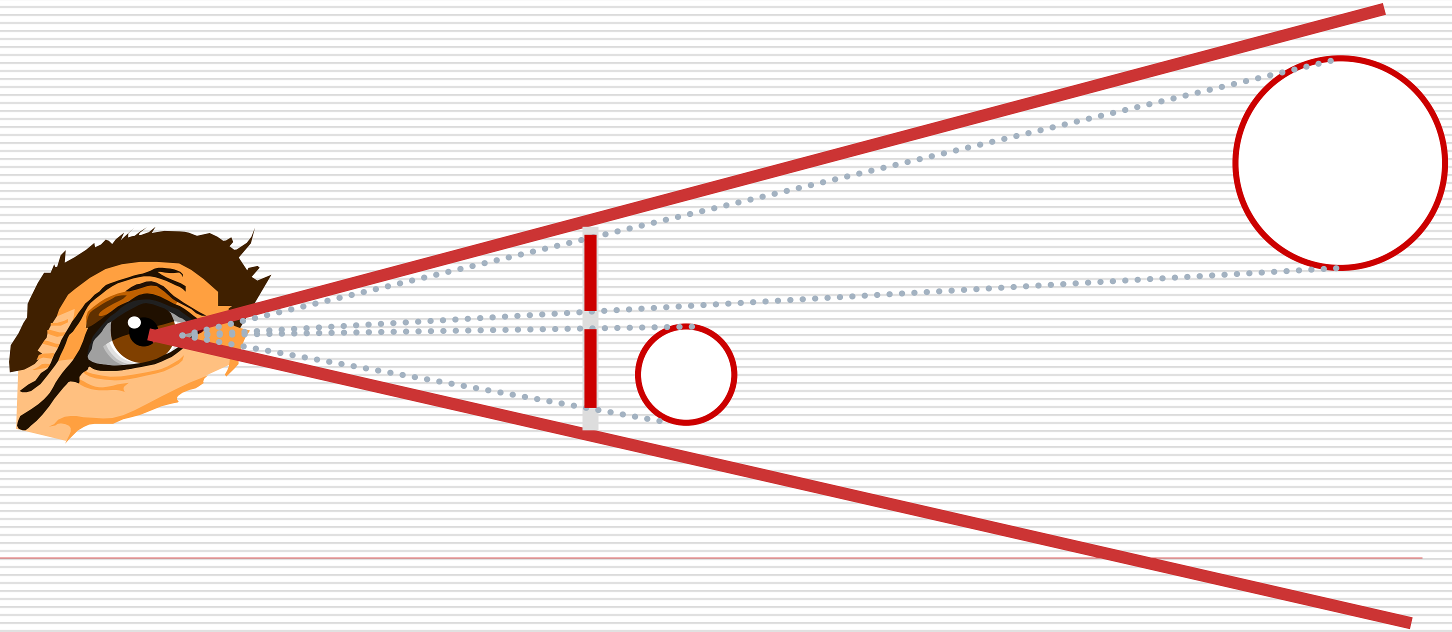
Projection Transform

Result:

- 2-D screen coordinates of clipped vertices

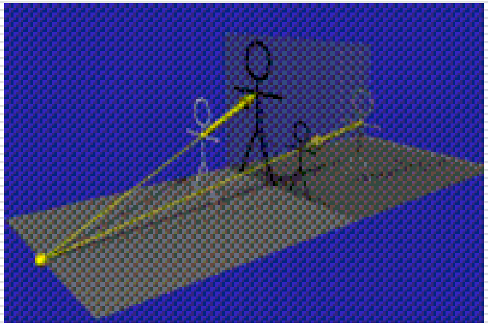
Rendering: Projection Transformations

- Projection transform
  - Apply perspective foreshortening
    - Distant = small: the *pinhole camera* model
  - View coordinates ⇨ screen coordinates

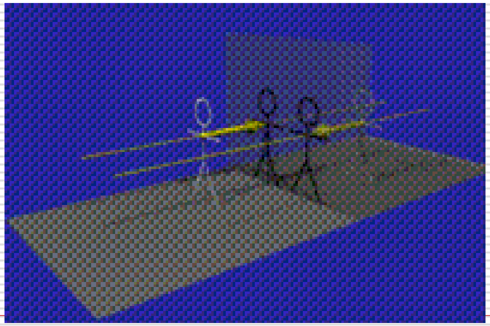


Rendering: Transformations

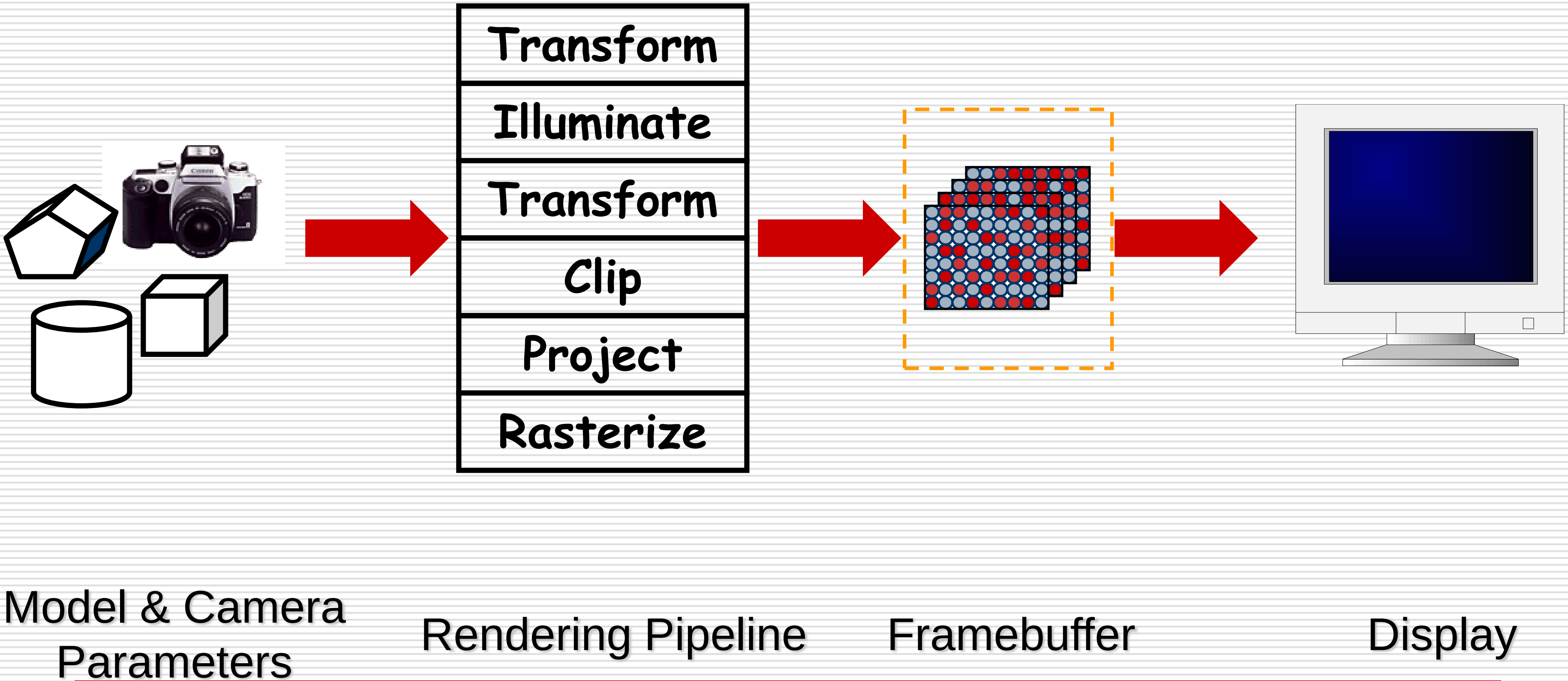
- Perspective Camera



- Orthographic Camera

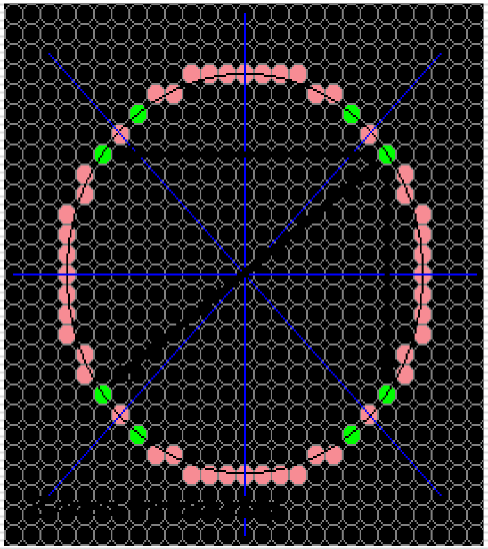


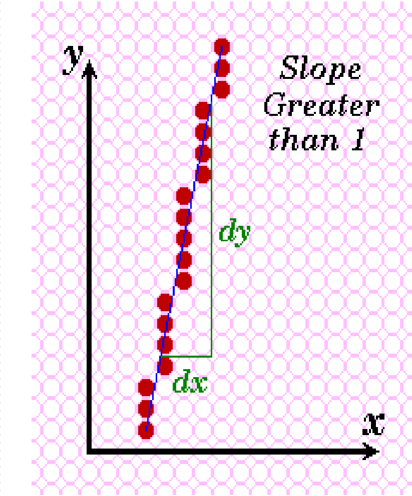
Rendering 3D Scenes

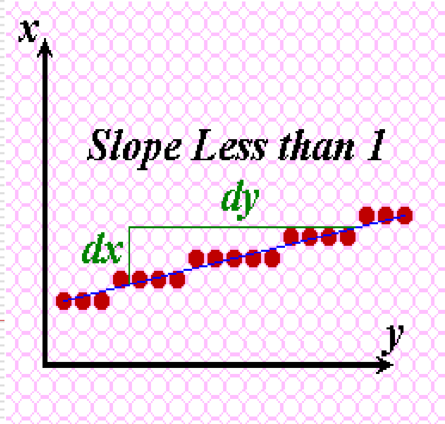


Rasterize

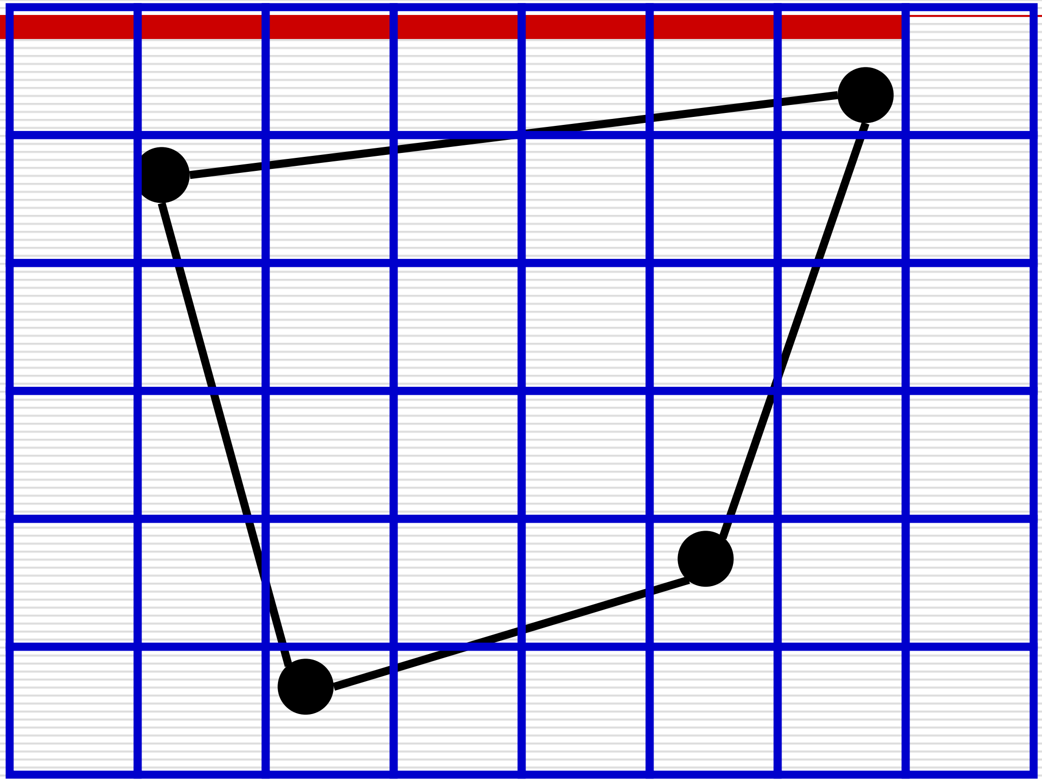
- Convert screen coordinates to pixel colors







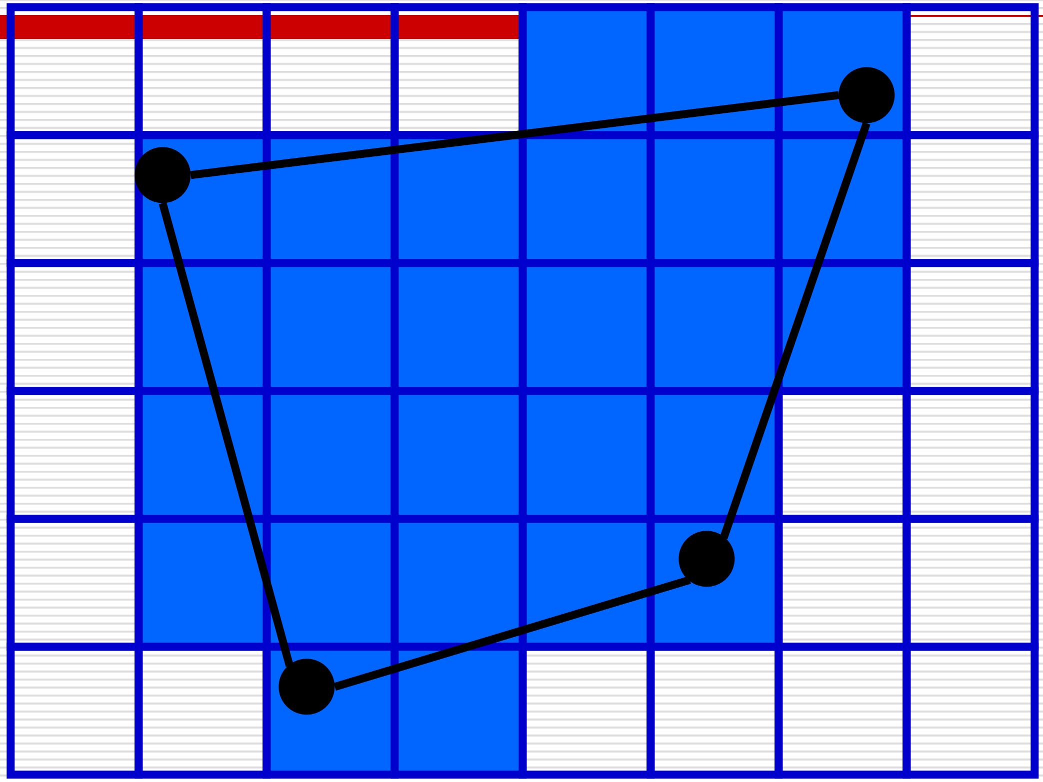
Rasterization



- Vertex T/L
- Rasterization
- Fragment
- Compositing

Quad “overlayed” over pixels

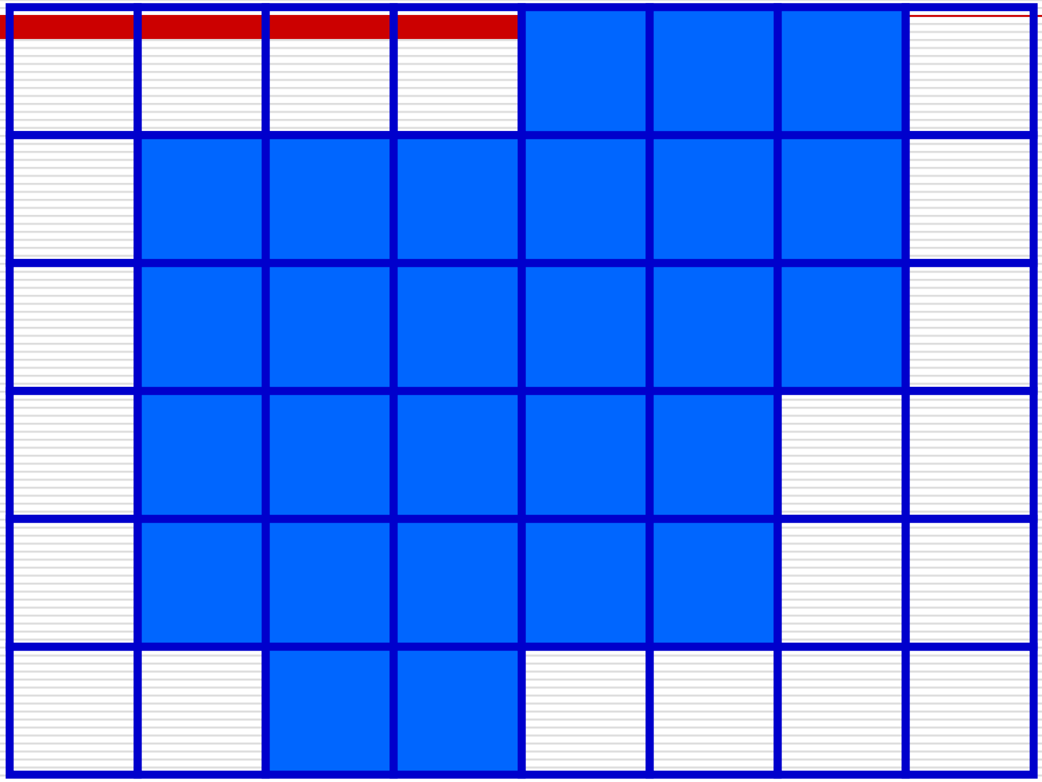
Rasterization



- Vertex T/L
- Rasterization
- Fragment
- Compositing

Fragments generated by the quad

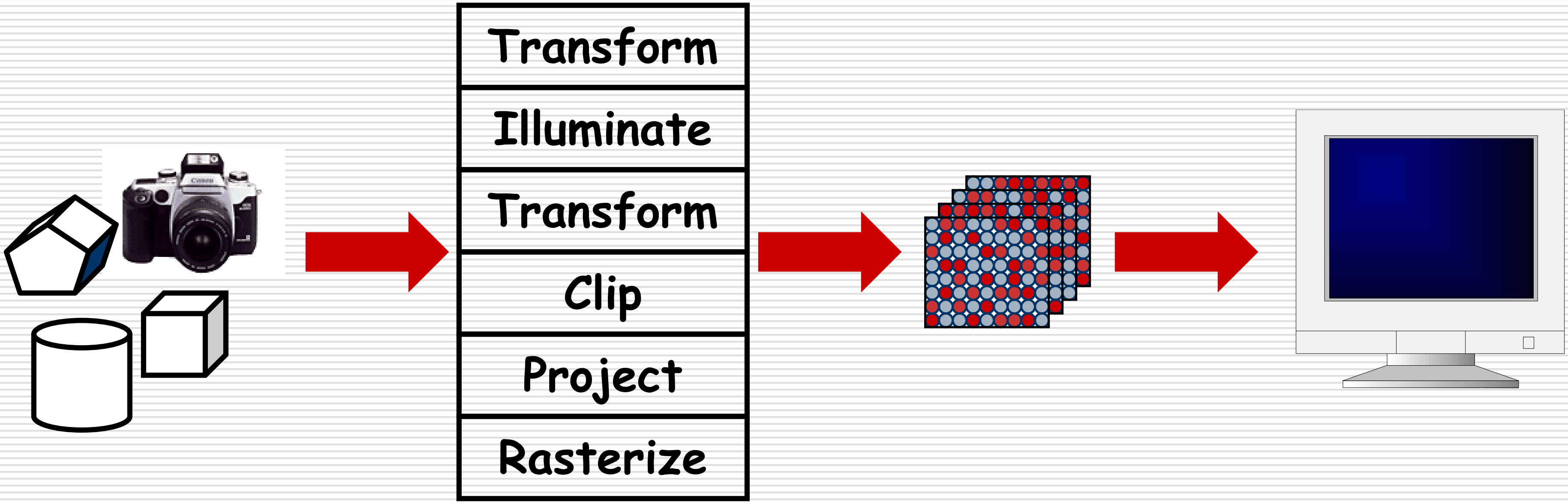
Rasterization



- Vertex T/L
- Rasterization
- Fragment
- Compositing

Fragments generated by the quad

Rendering 3D Scenes



Model & Camera Parameters

Rendering Pipeline

Framebuffer

Display

- 生成一个三维场景的步骤：
  - 场景几何造型；
  - 几何变换；
  - 取景变换；
  - 透视变换；
  - 灯光，纹理映射等

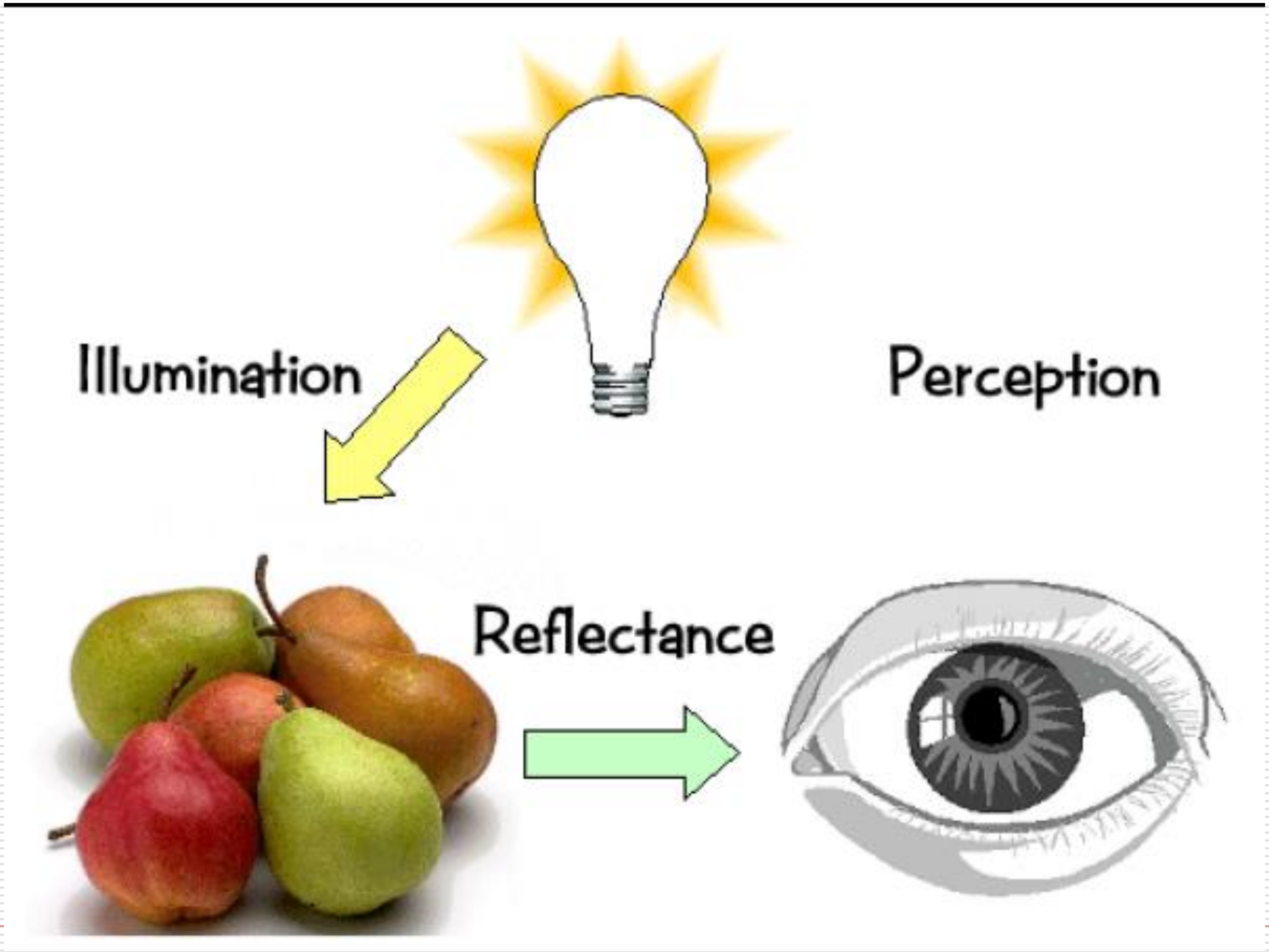
几何变换；取景变换；透视变换； → 通过矩阵变换来实现

[程序演示]

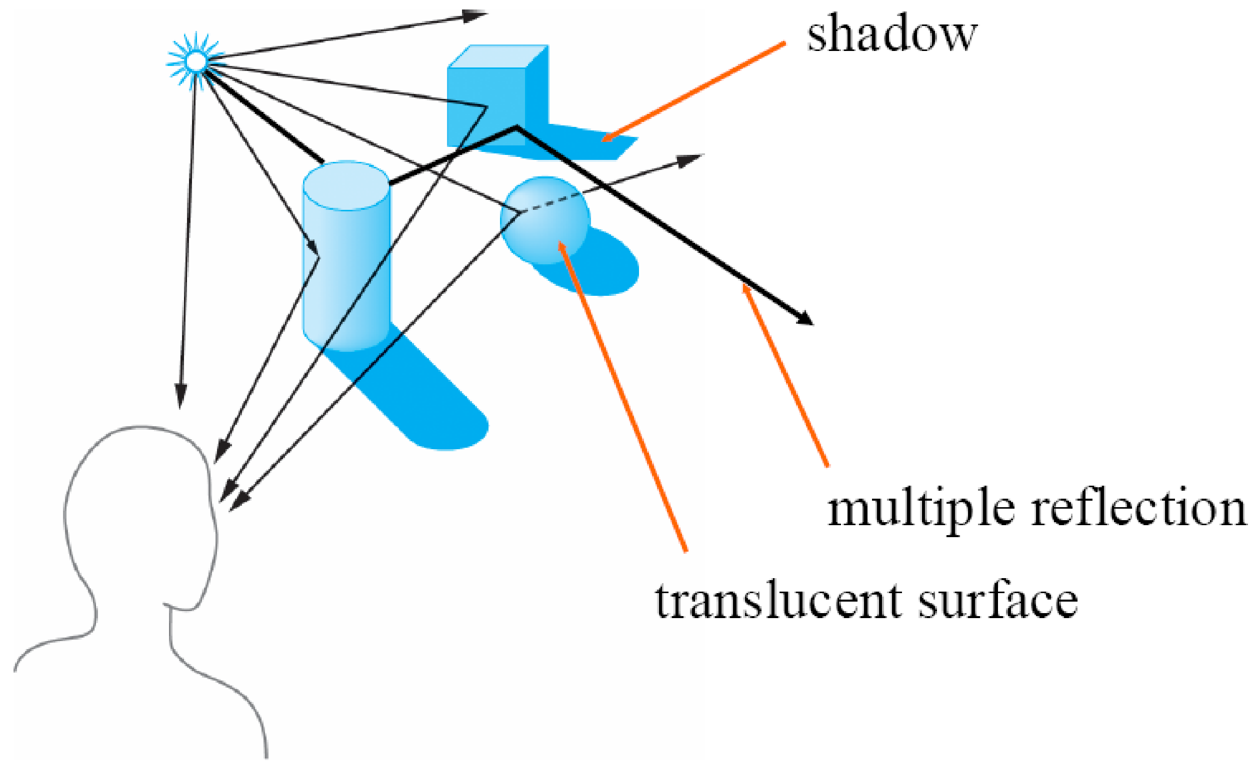
纲 要

- 场景绘制流水线
- 颜色成像原理
- 光照模型,灯光设置
- D3D程序实现

What we see the scene?

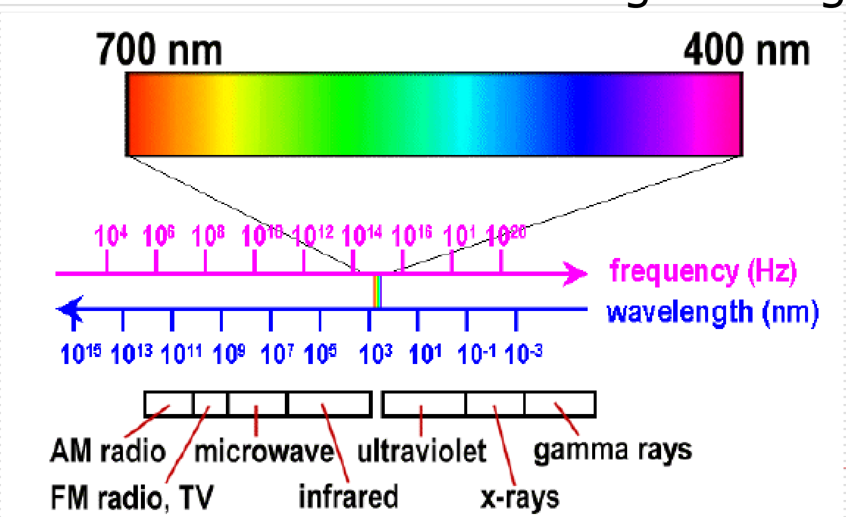


Light transport



Physics of Light and Color

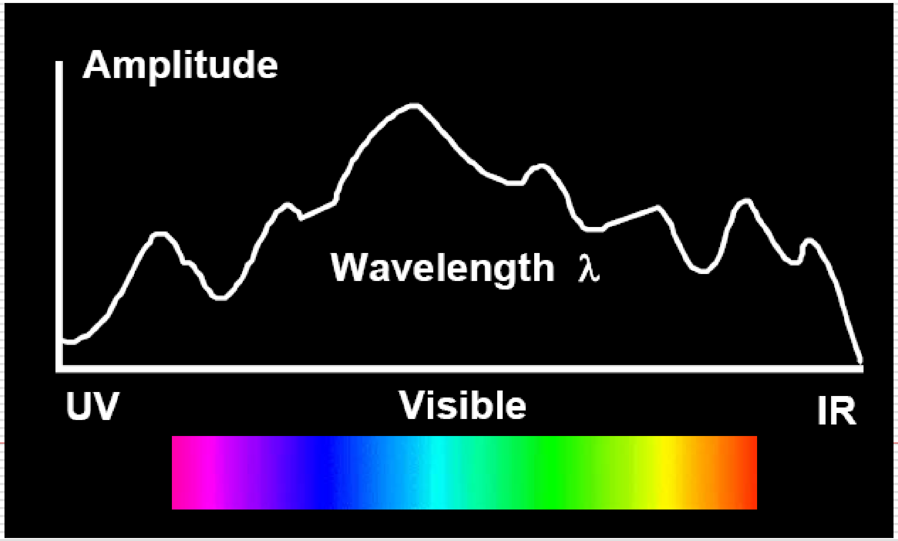
- It's all electromagnetic radiation
  - Different colors correspond to different ***wavelengths*** $\boldsymbol{\lambda}$***;***
  - Intensity of each wavelength specified by ***amplitude;***
  - ***Frequency*** $\boldsymbol{\nu = 2 \pi / \lambda}$**;**
    - long wavelength is low frequency
    - short wavelength is high frequency



We perceive EM radiation with λ in the 400-700 nm range

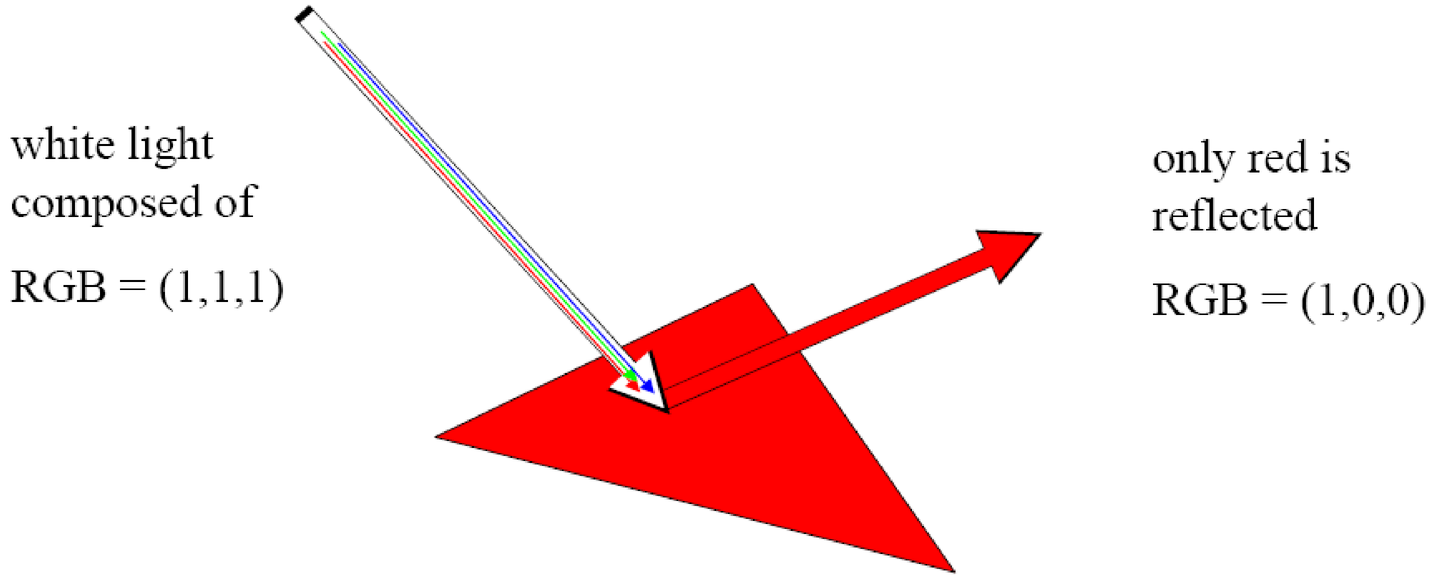
What's There vs. What We See

- Human eyes respond to "visible light"
  - tiny piece of spectrum between infra-red and ultraviolet
- Color defined by emission spectrum of light source
  - amplitude vs wavelength (or frequency) plot



Why different color?

- A surface appears red under white light because the red component of the light is reflected and the rest is absorbed:



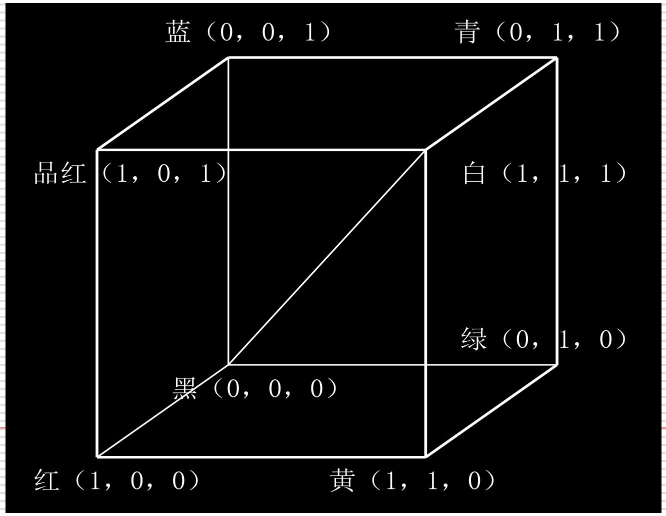
How to show color

- **Use Color Model**!

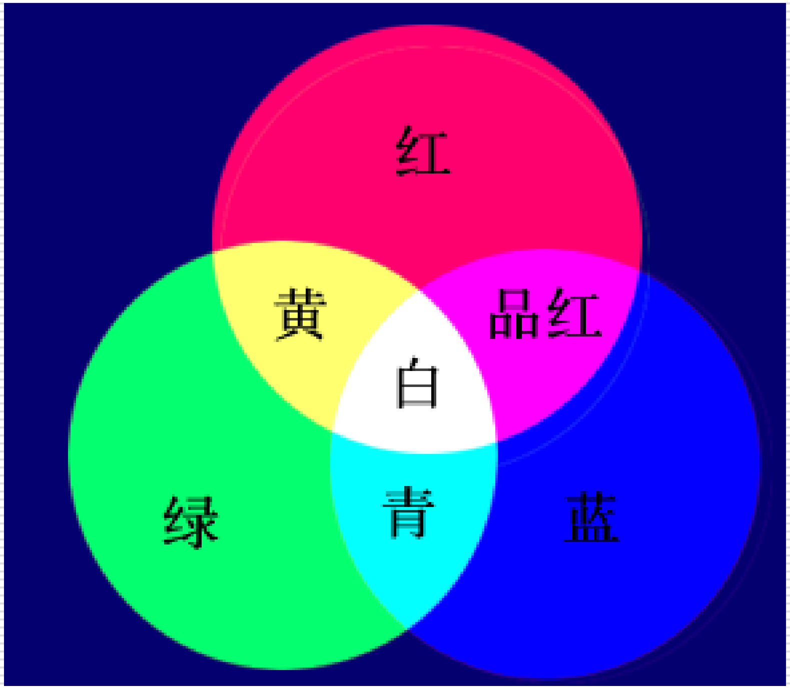
- Often have 3 color channels:
  - TV would be much more complex if we perceived the full spectrum
    - transmission would require much higher bandwidths
    - display would require much more complex methods
  - real-time color 3D graphics is feasible
  - any scheme for describing color requires only three values
  - lots of different color spaces--related by 3x3 matrix
  - transformations

常用颜色模型

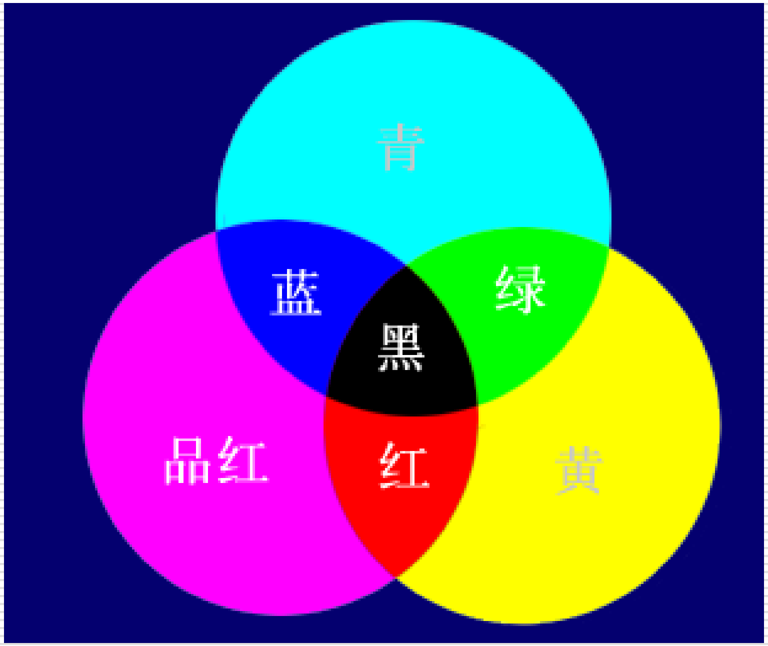
- RGB颜色模型
  - 加色模型,常使用于彩色光栅图形显示设备中
  - 真实感图形学中的主要的颜色模型
  - 采用三维直角坐标系
  - RGB立方体



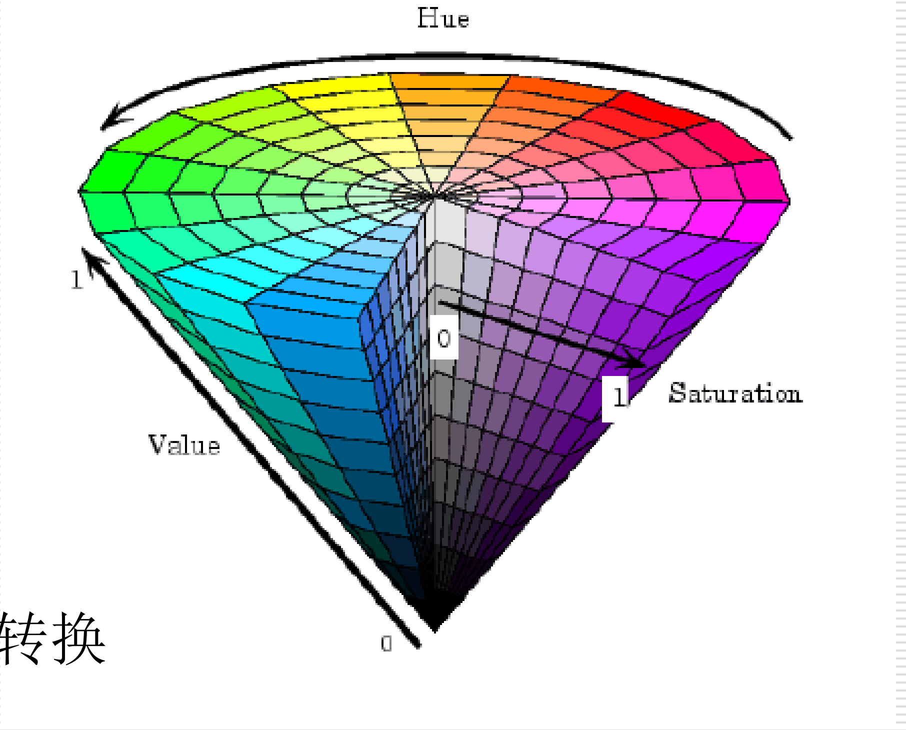
- 红、绿、蓝原色混合在一起可以产生复合色.
- 三原色混合效果:
- 计算机上表示任意颜色:
  - RGB(r,g,b);
    - r,g,b为1-255之间的整数
    - r,g,b为0-1之间的浮点数



- CMY颜色模型
  - 减色模型.
  - 白色-红色=青色
    白色-绿色=品红色
    白色-蓝色=黄色
  - 用于印刷硬拷贝设备

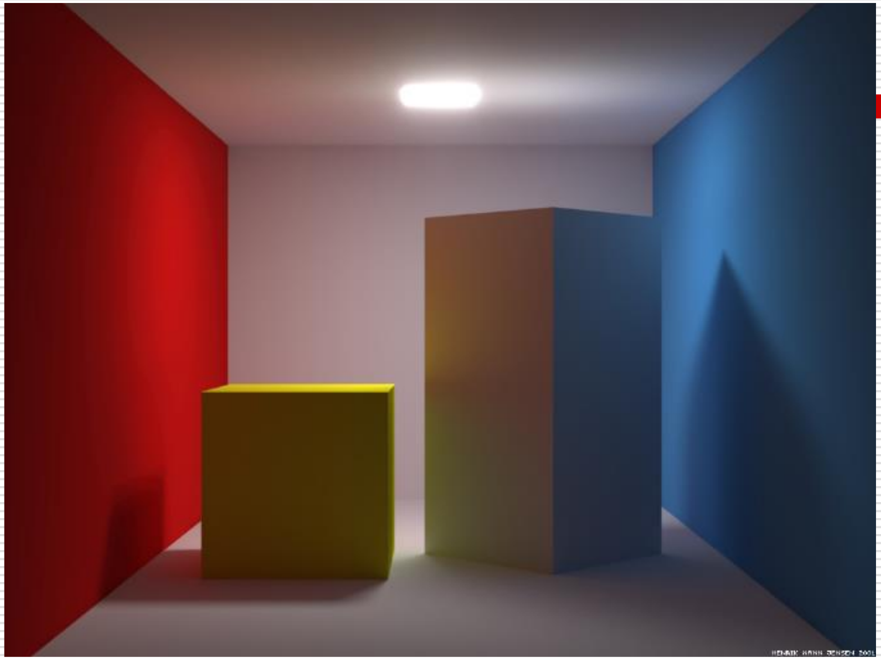


- HSV颜色模型
  - HSV颜色模型是面向用户的
    - Hue: 色度
    - Saturation: 饱和度
    - Value: 亮度
  - 不同颜色模型之间可以互相转换



纲 要

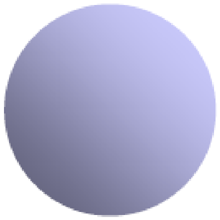
- 场景绘制流水线
- 颜色成像原理
- 光照模型,灯光设置
- D3D程序实现

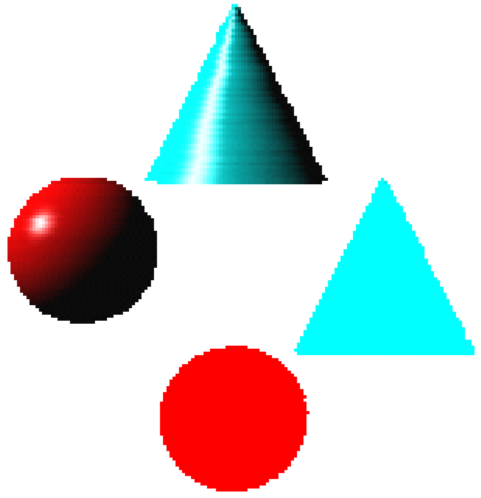


- Suppose we build a model of a sphere using many polygons and color it with `glColor`

- We get something like:

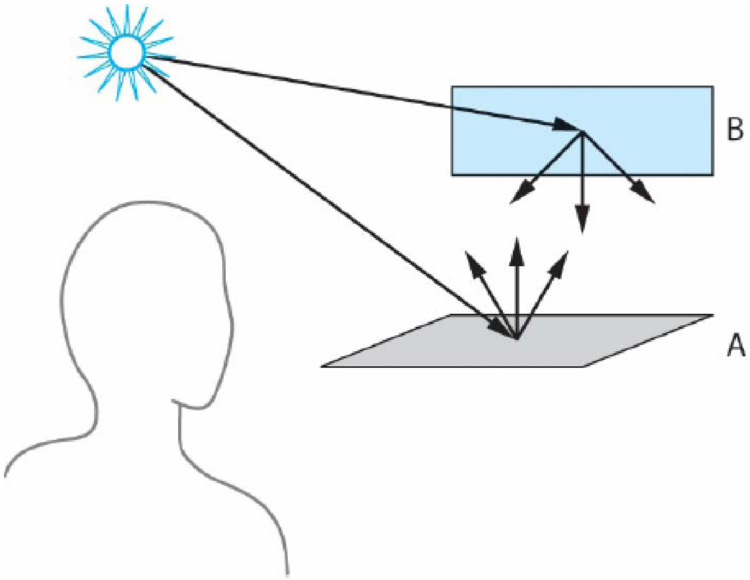
- But we want:



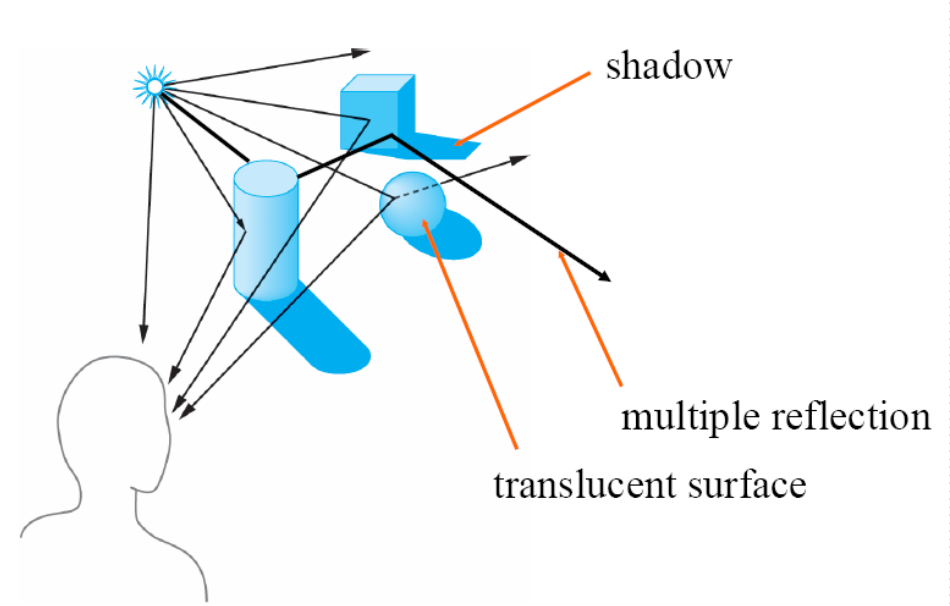


Light scattering

- Light strikes A
  - Some scattered
  - Some absorbed
- Some of scattered light strikes B
  - Some scattered
  - Some absorbed
- Some of this scattered light strikes A
- and so on…

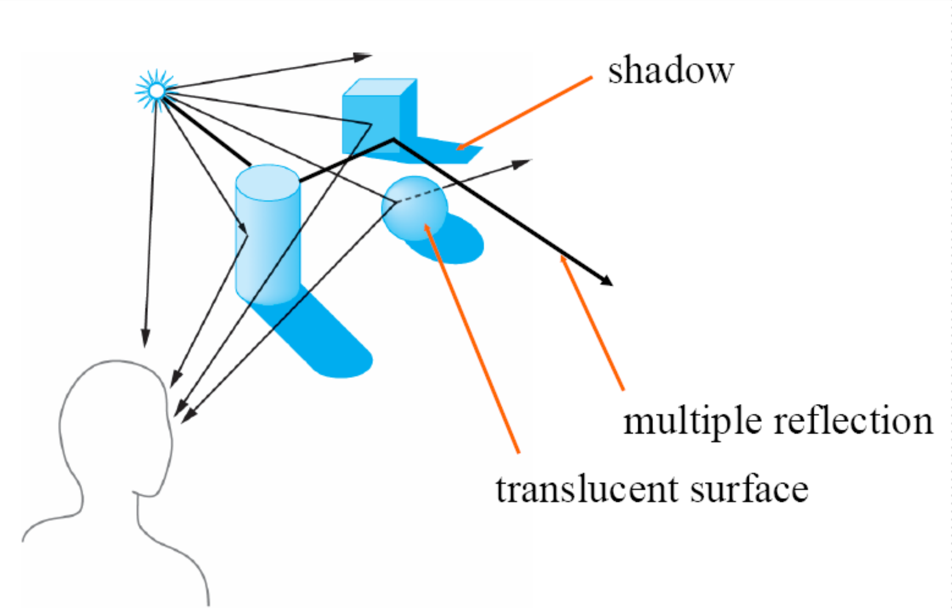


- For each vertex, light transports into eye:
  - Directional ray
  - Reflection ray
  - Multiple reflection ray
  - Environment



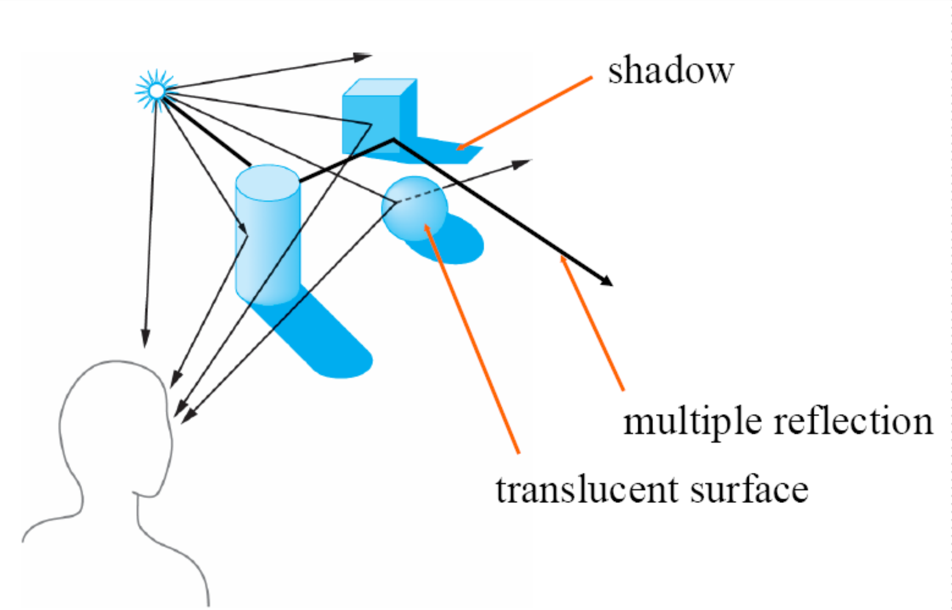
It is very slow if we integrate light coming from all points on the source.

- For each vertex, light transports into eye:
  - Directional ray
  - Reflection ray
  - Multiple reflection ray
  - Environment



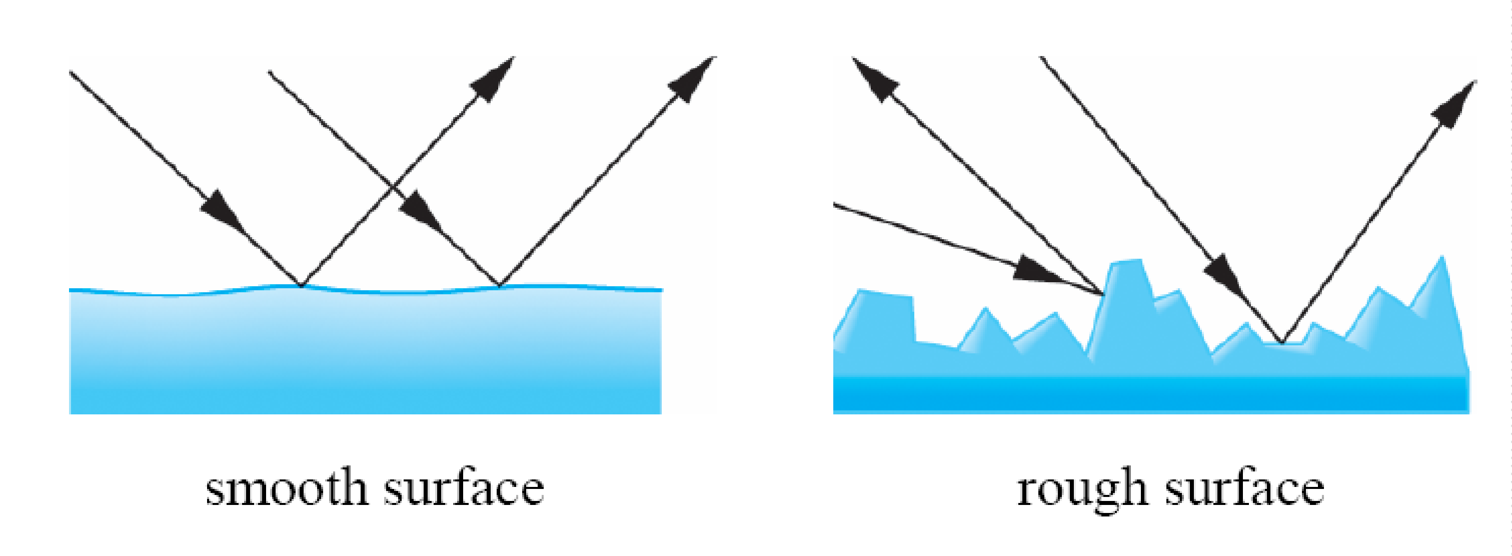
We can ignore the multiple reflection between objects.

- For each vertex, light transports into eye:
  - Directional ray
  - Reflection ray
  - Multiple reflection ray
  - Environment



We can ignore the multiple reflection between objects, that is local lighting model.

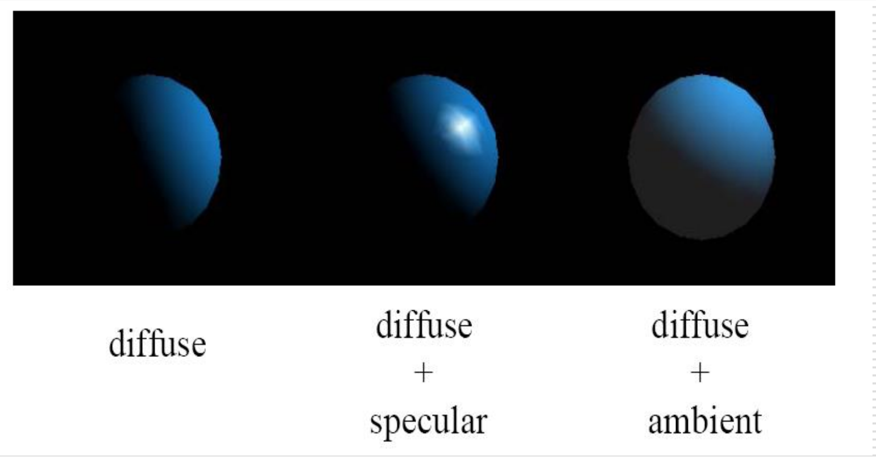
- Surface Types

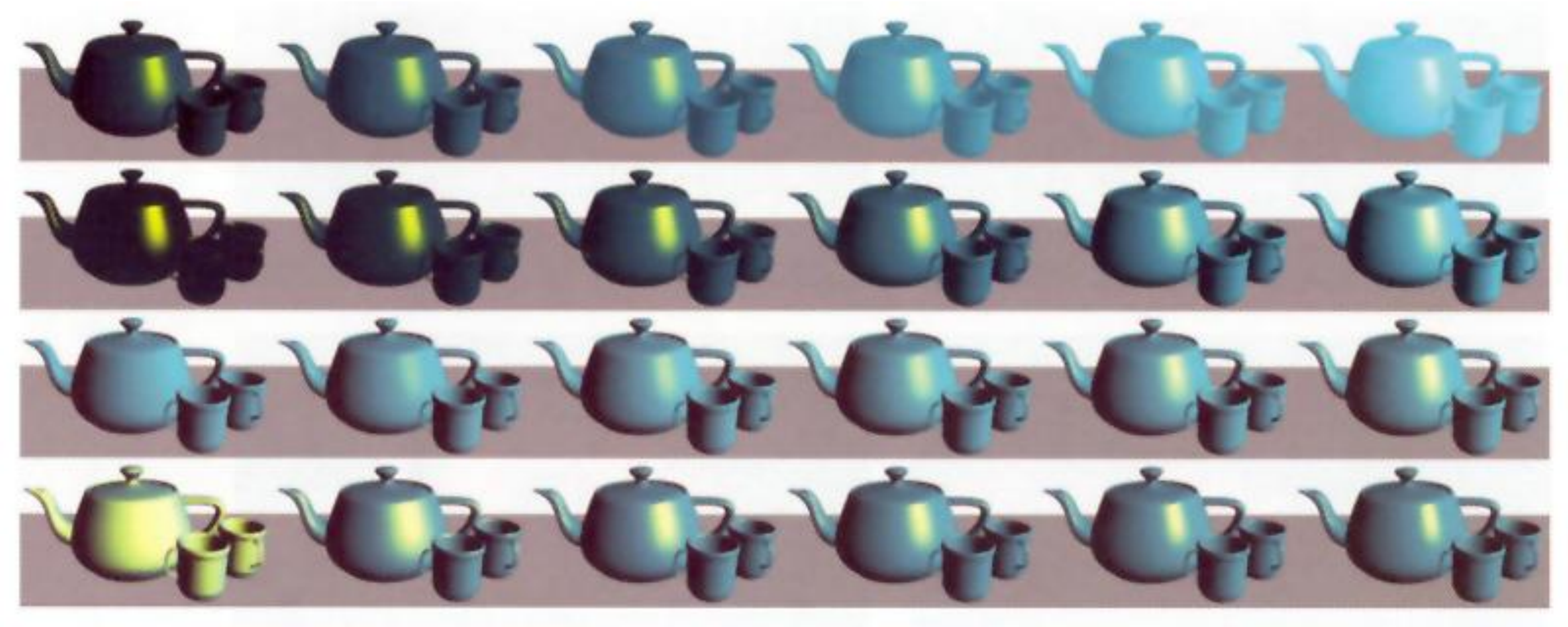


The Phong Lighting Model:

- A simple model that can be computed rapidly;
- Can model a range of surfaces from rough to Smooth;

- Has three components:
  - Diffuse : rough
  - Specular : shinny
  - Ambient :
    - background light level





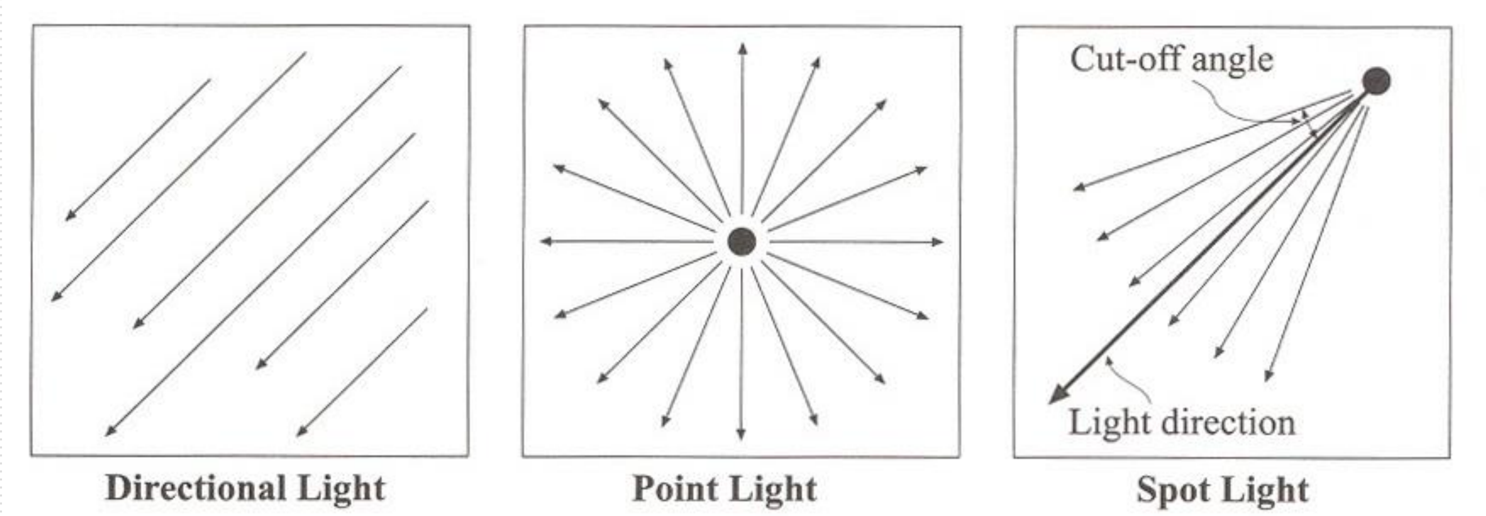
First line: Ambient change;

Second line: Diffuse change;

Third line: Specular change ;

光源类型:

- 方向光：放在离物体无穷远的地方,如：*太阳光*
- 点光源：点光源可看成发射光子的点。
- 聚光灯：方向矢量$\mathbf{s}_{dir}$、发散角$\mathbf{s}_{cut}$、Spot exponent $\mathbf{s}_{exp}$(控制光从中心向周围的衰减).



Phong Lighting Model

$$I_{out}(\mathbf{x}) = k_a \cdot I_a + k_d \cdot (\mathbf{l} \cdot \mathbf{n}) \cdot I_{diff} + k_s \cdot (\mathbf{h} \cdot \mathbf{n})^n \cdot I_{spec}$$

ambient diffuse specular

（1） 环境光

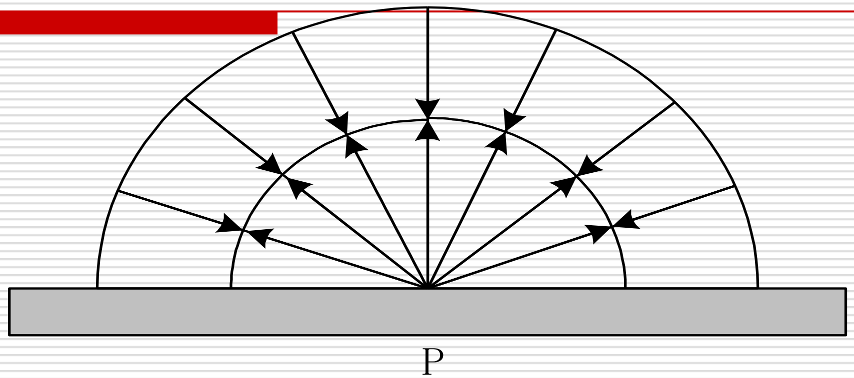


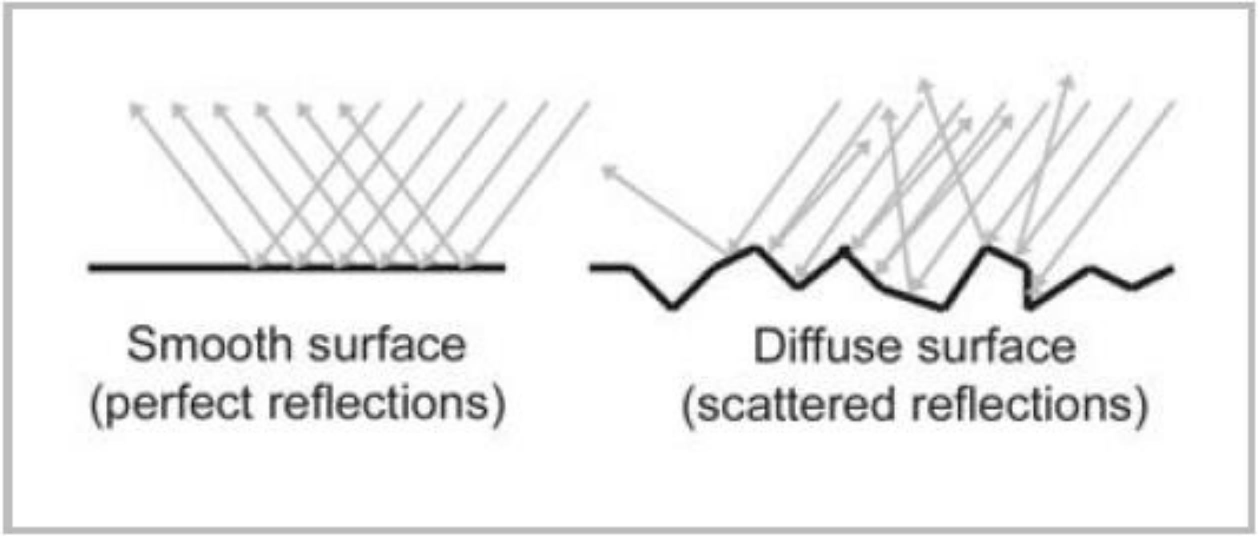
图10-1 环境光的反射

- 环境光是指光源间接对物体的影响
- 光在物体和环境之间多次反射，最终达到平衡
- 同一环境下的环境光光强分布均匀
- 近似表示：

$$I_{out}(\mathbf{x}) = k_a \cdot I_a$$

  - 为物体对环境光的反射系数

- （2） 漫反射光


Smooth surface
(perfect reflections)
Diffuse surface
(scattered reflections)

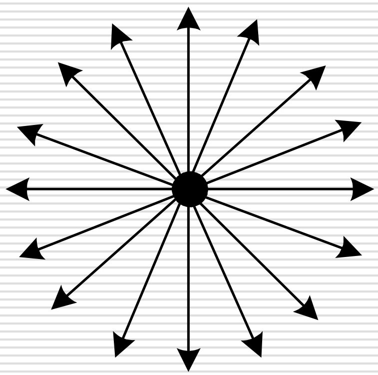


图10-2 点光源发射的光线路径

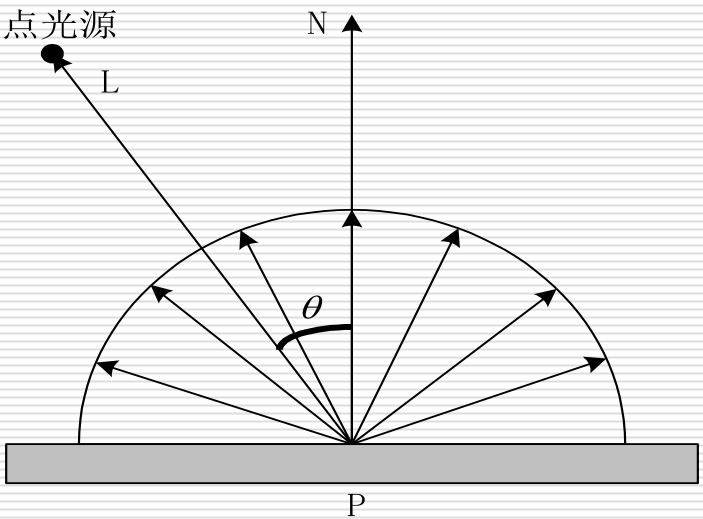


图10-3 漫反射

由Lambert余弦定理可得点P处漫反射光的强度为：

$$I_d = I_p K_d \cos\theta, \theta \in [0, \frac{\pi}{2}]$$

若L和N都已规格化为单位矢量，则：

$$I_d = I_p K_d (L \cdot N)$$

有多个点光源：

$$I_d = \sum_{i=1}^{n} I_{p,i} K_d (L_i \cdot N)$$

K_d 是与物体有关的漫反射系数， $0 < K_d < 1$

□ (3) 镜面反射光

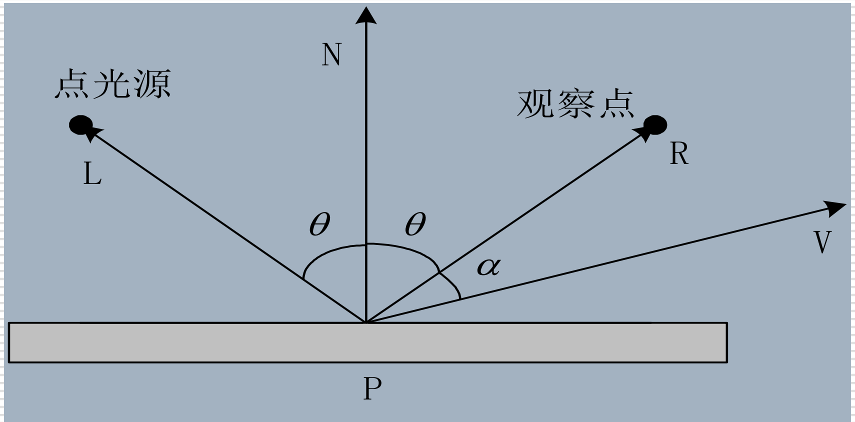


图10-4 镜面反射

镜面反射情况由Phong模型给出：

$$I_s = I_p K_s \cos^n \alpha$$

若R和V已规格化为单位矢量，则：

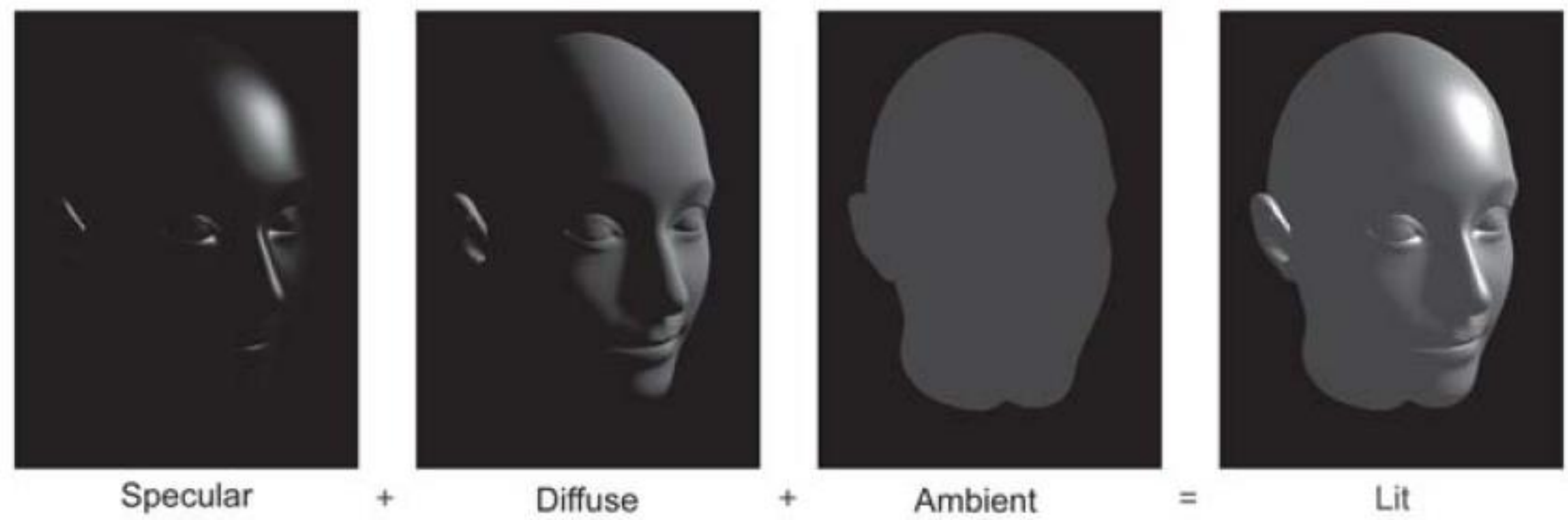
$$I_s = I_p K_s (R \cdot V)^n$$

Summarize Phong:

从视点观察到物体上任一点P处的光强度I应为环境光反射光强度I_e、漫反射光强度I_d以及镜面反射光的光强度I_s的总和：

$$I = I_e + I_d + I_s$$

$$= I_a K_a + I_p K_d (L \cdot N) + I_s K_s (R \cdot V)^n$$



（4） 光强衰减

- 一个常用的二次衰减函数可以表示为：

$$f(d) = \min(1, \frac{1}{c_0 + c_1 d + c_2 d^2})$$

- 则基本光照明函数变为：

$$I = I_a K_a + \sum_{i=1}^{n} f_i(d) I_l [K_d (L_i \cdot N) + K_s (R \cdot V_i)^{n_s}]$$

光强颜色转换

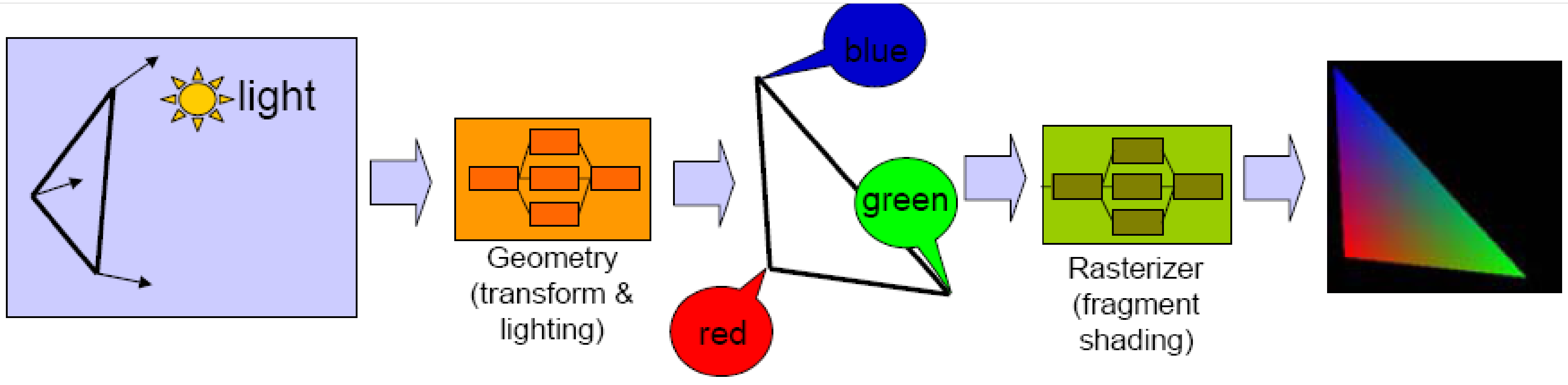
- **如果绘制的是黑白图形,只需要对应线性转换即可.**
- 对彩色图形:
  - 对应计算出每个顶点红,绿,蓝三种颜色对应的I值:

$$I_B = I_{aB} K_{aB} + \sum_{i=1}^{n} f_i(d) I_{lB_i} [K_{dB}(L_i \cdot N) + K_{sB}(N \cdot H_i)^{n_s}]$$

  - 利用查表将颜色转化为XYZ颜色,再转化为RGB颜色.

多边形绘制明暗处理

- 如何绘制整个几何体的真实外观?
- 如果光照计算针对每个像素绘制,将非常耗时!
- 一般先计算几何顶点处颜色,然后做明暗处理.



Flat Shading

- 一个面片的颜色直接取其中一个顶点的颜色

- Gouraud明暗处理方法，
  - 又称为亮度插值明暗处理，它通过对多边形顶点颜色进行线性插值来绘制其内部各点。

Phong明暗处理

- 又称为法矢量插值明暗处理，它对多边形顶点的法矢量进行插值以产生中间各点的法矢量。
- Phong明暗处理的基本步骤是：
  1. 计算每个多边形顶点处的平均单位法矢量。
  2. 用双线性插值方法求得多边形内部各点的法矢量。
  3. 最后按光照模型确定多边形内部各点的光强。

纲 要

- 场景绘制流水线
- 颜色成像原理
- 光照模型,灯光设置
- **D3D程序实现**

D3D中实现带光照的场景?

D3D的基本灯光设置

- 默认镜面光被关闭的；要使用镜面光你必须设置D3DRS_SPECULARENABLE渲染状态。

```
Device->SetRenderState(D3DRS_SPECULARENABLE, true);
```

- 每一种灯光都是通过D3DCOLORVALUE结构或者描述灯光颜色的D3DXCOLOR来描绘的。

```
D3DXCOLOR redAmbient(1.0f, 0.0f, 0.0f, 1.0f);
D3DXCOLOR blueDiffuse(0.0f, 0.0f, 1.0f, 1.0f);
D3DXCOLOR whiteSpecular(1.0f, 1.0f, 1.0f, 1.0f);
```

材质

- 在现实世界中我们看到的物体颜色将由物体反射回来的灯光颜色来决定。
- Direct3D通过定义的物体材质来模拟这些所有的现象。材质允许我们定义表面反射灯光的百分比。在代码中通过D3DMATERIAL9结构描述一个材质。

```
typedef struct _D3DMATERIAL9 {
    D3DCOLORVALUE Diffuse, Ambient, Specular, Emissive;
    float Power;
} D3DMATERIAL9;
```

- Diffuse—指定此表面反射的漫射光数量。
- Ambient—指定此表面反射的环境光数量。
- Specular—指定此表面反射的镜面光数量
- Emissive—这个是被用来给表面添加颜色，它使得物体看起来就象是它自己发出的光一样。
- Power—指定锐利的镜面高光,它的值是高光的锐利值。

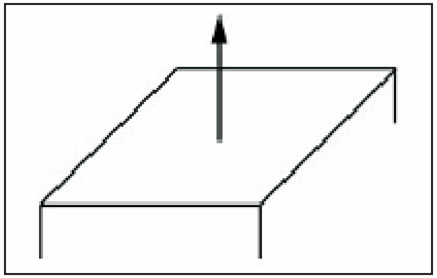
□ 举例，想得到一个红色的球。我们将定义球的材质来只反射红光吸收其他颜色的所有光：

```
D3DMATERIAL9 red;
::ZeroMemory(&red, sizeof(red));
red.Diffuse = D3DXCOLOR(1.0f, 0.0f, 0.0f, 1.0f); // red
red.Ambient = D3DXCOLOR(1.0f, 0.0f, 0.0f, 1.0f); // red
red.Specular = D3DXCOLOR(1.0f, 0.0f, 0.0f, 1.0f); // red
red.Emissive = D3DXCOLOR(0.0f, 0.0f, 0.0f, 1.0f); // no emission
red.Power = 5.0f;
```

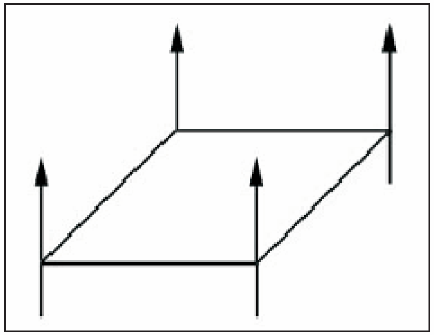
□ 设置使用IDirect3DDevice9::SetMaterial(CONST D3DMATERIAL9* pMaterial)方法。

顶点法线

- 面法线（*face normal*）是描述多边形表面方向的一个向量:



- 顶点法线（*Vertex normals*）也是基于同样的概念，但是我们与其指定每个多边形的法线，还不如为每个顶点指定:



- Direct3D需要知道顶点法线以便它能够确定灯光照射到物体表面的角度.

- 为了描述顶点的顶点法线，我们必须更新原来的顶点结构：

```
struct Vertex
{
    float _x, _y, _z;
    float _nx, _ny, _nz;
    static const DWORD FVF;
}
```

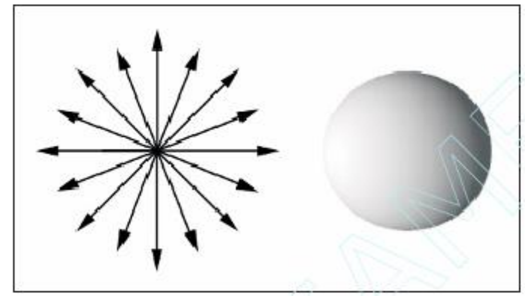
- 注意，我们

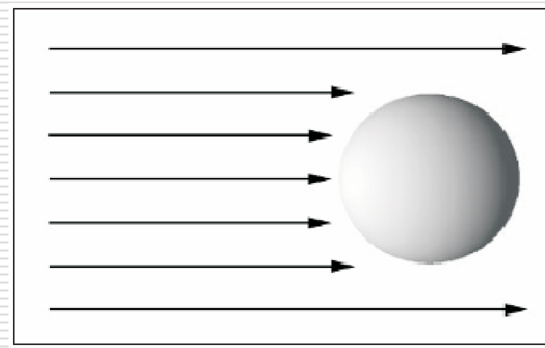
```
const DWORD Vertex::FVF = D3DFVF_XYZ | D3DFVF_NORMAL;
```

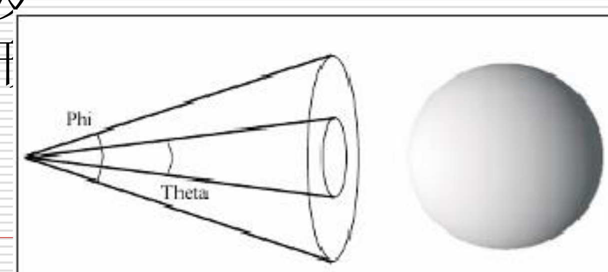
除了。这是因为我们将使用灯光来计算顶点的颜色。

□ 光源

- Direct3D支持三种类型的光源:
  - □ 点光源——这种光源在世界坐标中有一个位置且向所有方向上都照射光线。
  - □ 方向光源——这种光源没有位置但是向指定方向发出平行光线;
  - □ 聚光灯——这种类型的光源和手电筒的光类似;它有位置并且发出的光在指定方向上按照圆锥形照射。







- 光源是通过D3DLIGHT9结构来表现的。

```
typedef struct _D3DLIGHT9 {
    D3DLIGHTTYPE Type;
    D3DCOLORVALUE Diffuse;
    D3DCOLORVALUE Specular;
    D3DCOLORVALUE Ambient;
    D3DVECTOR Position;
    D3DVECTOR Direction;
    float Range;
    float Falloff;
    float Attenuation0;
    float Attenuation1;
    float Attenuation2;
    float Theta;
    float Phi;
} D3DLIGHT9;
```

- Type——定义灯光类型，我们能够使用下面三种类型之一：D3DLIGHT_POINT, D3DLIGHT_SPOT, D3DLIGHT_DIRECTIONAL
- Diffuse——此光源发出的漫射光颜色。
- Specular——此光源发出的镜面光颜色。
- Ambient——此光源发出的环境光颜色。
- Position——用一个向量来描述的光源世界坐标位置。这个值对于灯光的方向是无意义的。
- Direction——用一个向量来描述的光源世界坐标照射方向。这个值不能用在点光源上。
- Range——灯光能够传播的最大范围。这个值不能比大。且不能用于方向光源。
- Falloff——这个值只能用在聚光灯上。它定义灯光在从内圆锥到外圆锥之间的强度衰减。它的值通常设置为1.0f。
- Attenuation0, Attenuation1, Attenuation2——这些衰减变量被用来定义灯光强度的传播距离衰减。它们只被用于点光源和聚光灯上。
- Theta——只用于聚光灯；指定内圆锥的角度，单位是弧度。
- Phi——只用于聚光灯；指定外圆锥的角度，单位是弧度。

以InitDirectionalLight为例:

- 初始化:

```
D3DLIGHT9 d3d::InitDirectionalLight(D3DXVECTOR3* direction, D3DXCOLOR* color)
{
    D3DLIGHT9 light;
    ::ZeroMemory(&light, sizeof(light));
    light.Type = D3DLIGHT_DIRECTIONAL;
    light.Ambient = *color * 0.4f;
    light.Diffuse = *color;
    light.Specular = *color * 0.6f;
    light.Direction = *direction;
    return light;
}
```

- 然后创建一个方向光源，它沿着**x**轴正方向照射白色灯光:

```
D3DXVECTOR3 dir(1.0f, 0.0f, 0.0f);
D3DXCOLOR c = d3d::WHITE;
D3DLIGHT9 dirLight = d3d::InitDirectionalLight(&dir, &c);
```

- 在把**D3DLIGHT9**初始化好以后，设置改灯光:
  - **Device->Setlight(0, &light)**
- 打开灯光的开关:
  - **Device->LightEnable(0, true)**

Direct3D中设置光照

- 激活光源计算：

SetRenderState(D3DRS_LIGHTING,TRUE);

- 设置表面材质（反射系数等）

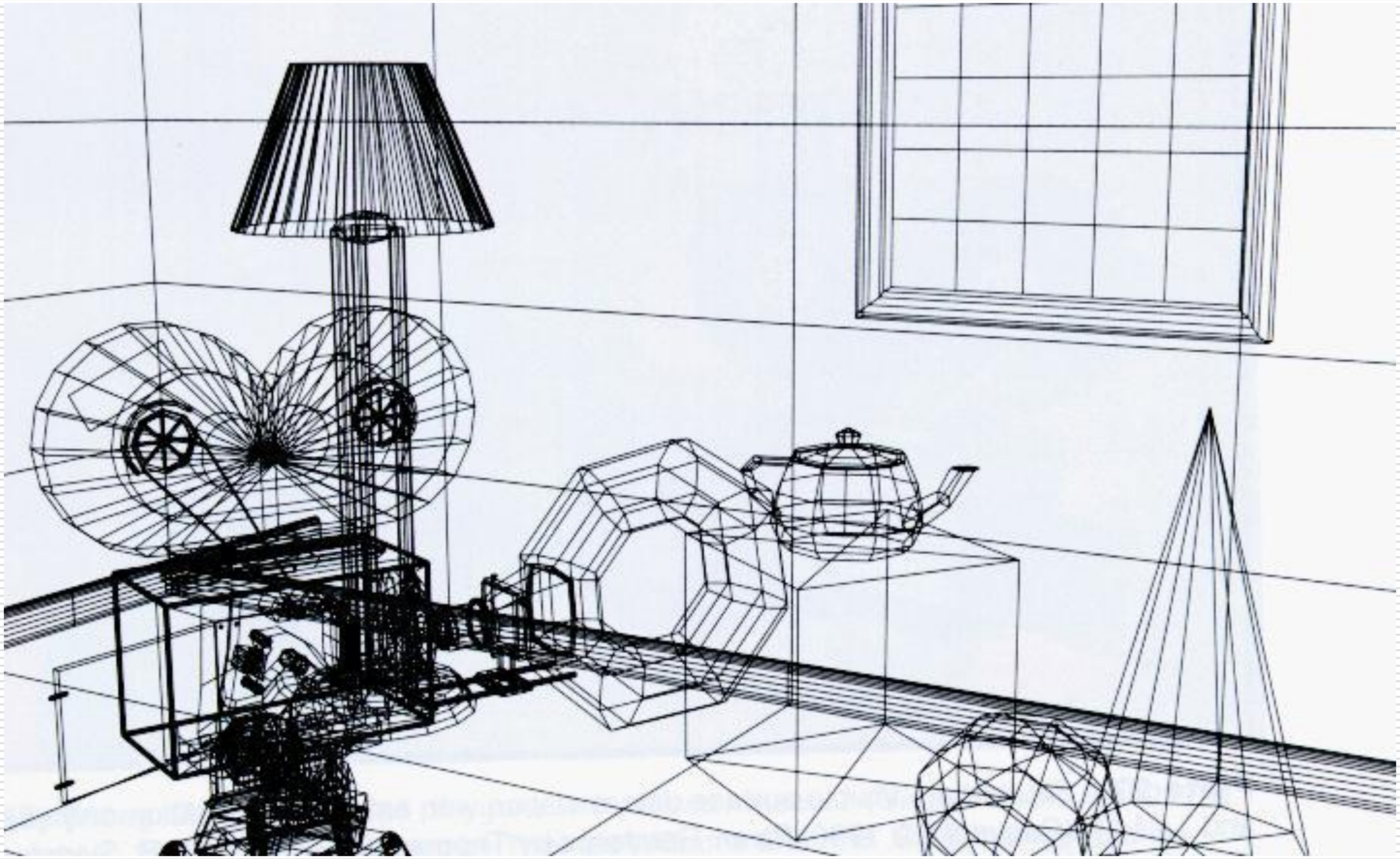
  SetMaterial()

- 设置光源属性：

  SetLight()

- 在多边形顶点信息中增加法线向量：
  - normal

Example – Wireframe

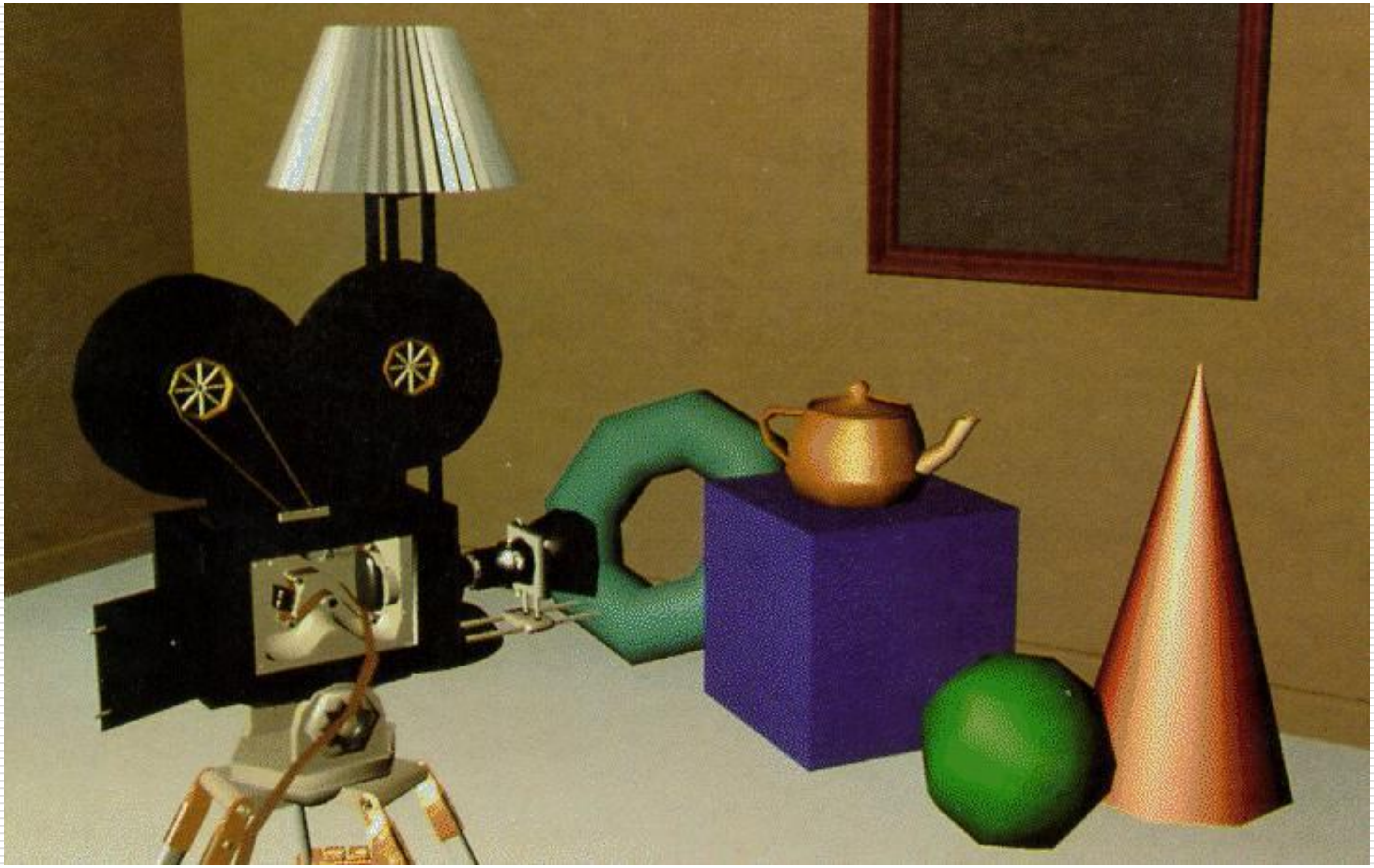


Example – Flat / Ambient

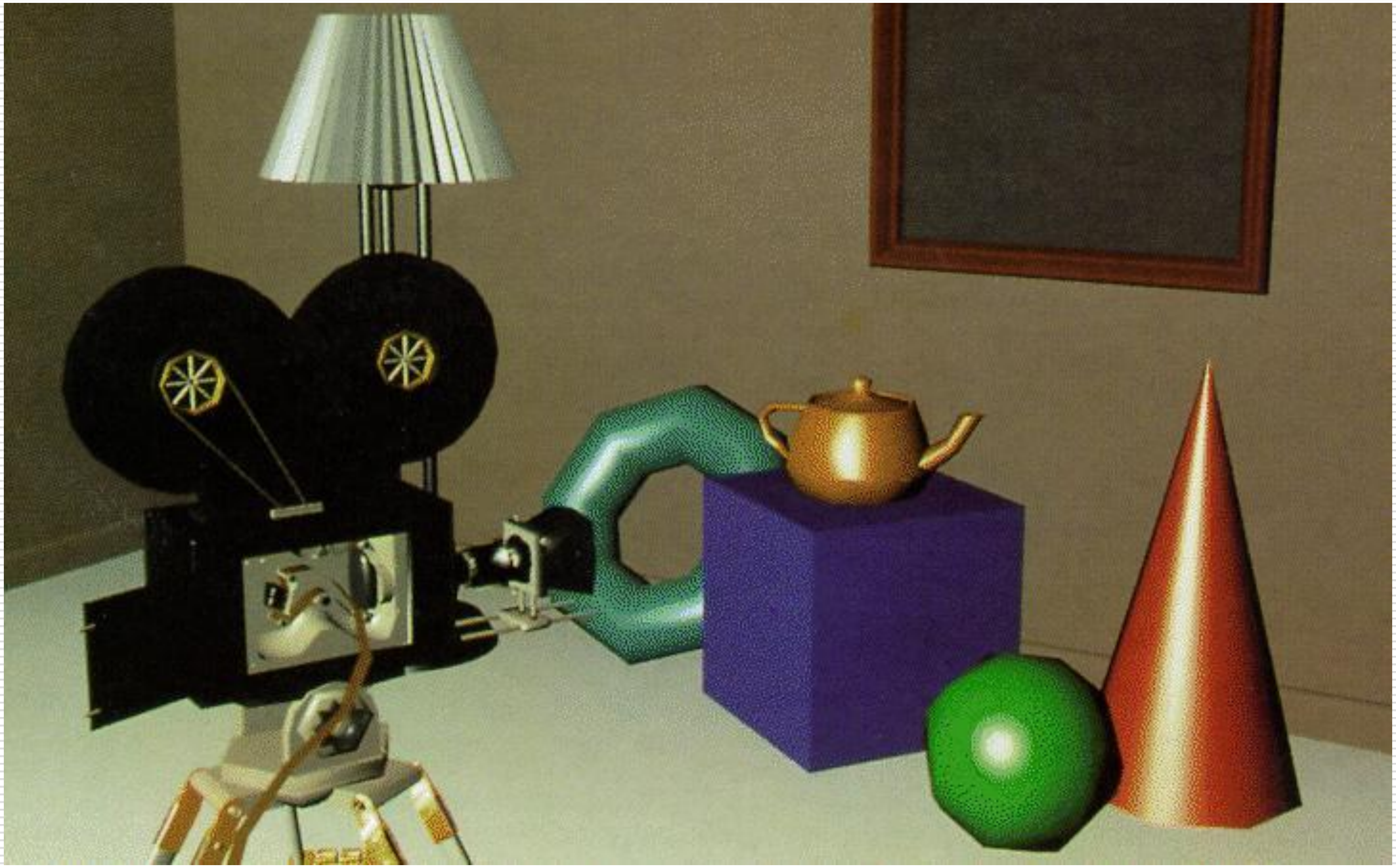
Example – Flat / Amb. + Diff.

Example – Gouraud / Amb. + Diff.

Example – Gouraud / Amb. + Diff. + Spec.



Example – **Phong / Amb. + Diff. + Spec.**



练习

- 熟悉D3D的例子Tut04_Lights程序, 画一个红色的立方体,并在场景中加入一个黄色的平行光和一个白色的点光源.