Interactive Computer Graphics

CS 418 – Spring 2011

MP2 Flight Simulator and Shading

TA: Gong Chen

Email: gchen10 at illinois.edu

Office Hours

To be posted on Piazza

Agenda

- Office Hour
- About MP2
- Multiple Object Rendering
- Lighting
- Q&A for midterm

MP2 : Flight Simulator

- Due on October 16th at 3:30PM
  - Camera Control (Flight Simulator)
  - Some Features:
    - Multiple Object rendering (Model Transformation)
    - Object picking/control
    - Terrain Texturing / Lighting

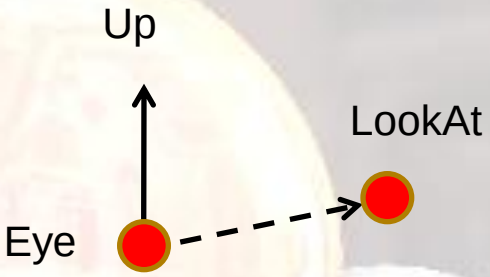
Flight Control

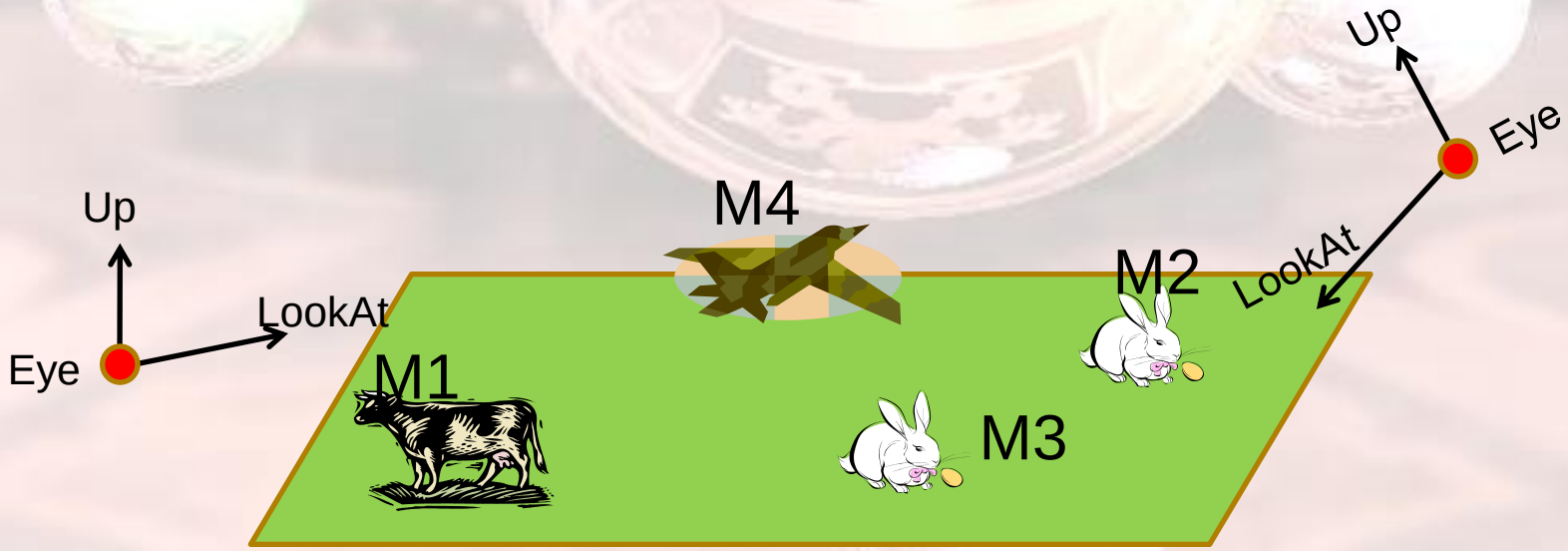
Move your camera based on user keyboard input

- Intuitive Ways(rotate camera)
  - Update gluLookAt parameters
  - keep track your own matrix transformation
    - Easier to think, but more work
    - Recommended if you don't want to mess with OpenGL transformation.
- Less intuitive Way(rotate the world)
  - Update OpenGL transformation
    - Easier to implement, but difficult to make it correct
    - Recommended if you are absolutely sure what to do.

Flight Control

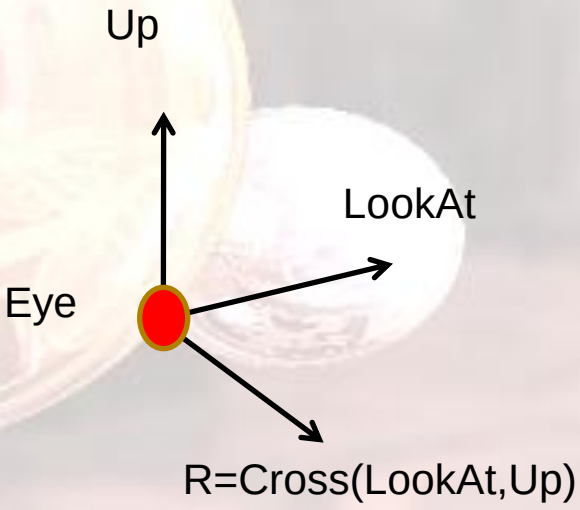
- gluLookAt :
  - Eye position
  - Look At point (direction)
  - Up vector





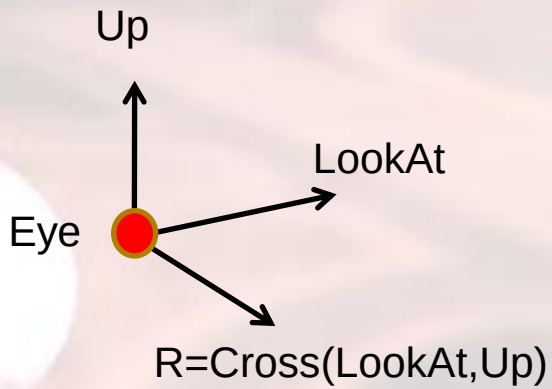
Flight Control

- Initilize Eye position and Up,LookAt, R vector.
- Move forward :
  - Offset Eye position in LookAt direction
- Tilt Up/Down
  - Rotate LookAt,Up about R axis.
- Turn Left/Right
  - Rotate UP, R about LookAt axis. Then tilt up and down



Flight Control

- Every time you press arrow keys, update Up,LookAt, R vector accordingly.
- Every time period (Ex : 1/30 sec), move Eye position.
- In display function, set look at function :
  - gluLookAt(Eye, Eye+LookAt, Up);



Flight Control

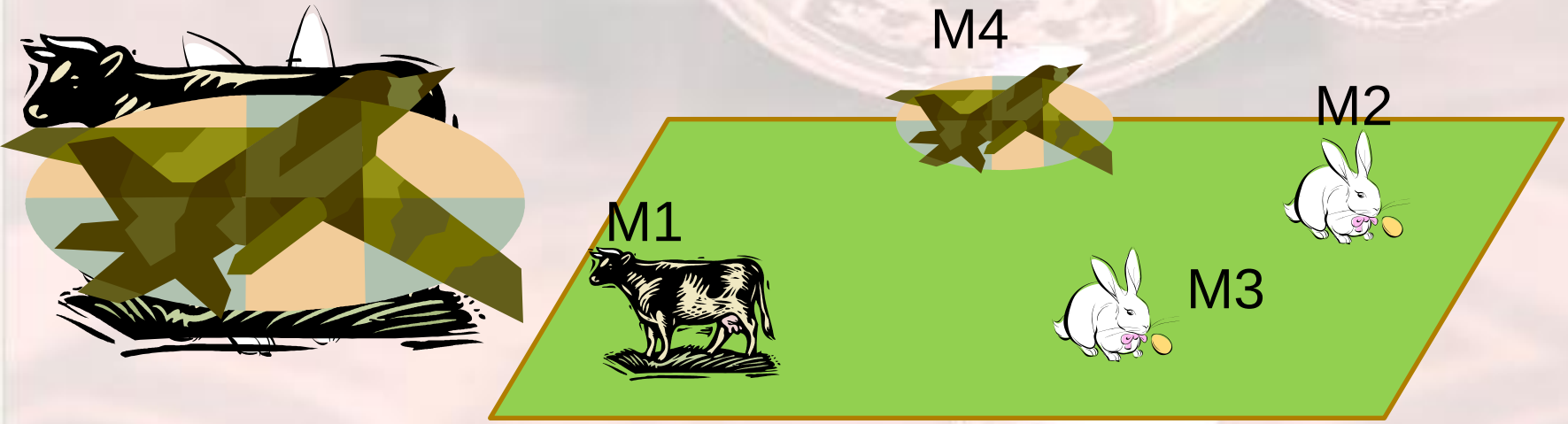
- Arrow Key Called-back function
  - glutSpecialFunc instead of glutKeyboardFunc
  - Refer to OpenGL doc. for its parameters.
- Reset OpenGL matrix before calling gluLookAt.
- You may use the formula in lecture slides to generate rotation matrix (axis-angle).

Flight Control

- Less Intuitive way
  - Moving camera is equivalent to moving every object in the world towards a stationary camera
  - Using a fixed gluLookAt, but call OpenGL transformation command instead.
  - Where should you put glTranslate/glRotate to simulate a flight simulator ?
    - Before or after gluLookAt ?
    - Pre-multiply or Post-multiply ?

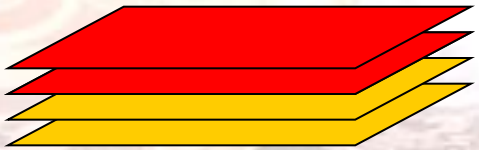
Multiple Object Rendering

- Model Transformation
  - Specify scaling, translation for each object
  - Apply different transformation on each mesh
  - Utilize push/pop matrix to backup matrix state



Push/Pop Matrix

- glPushMatrix()
  - Create a copy of top matrix & push it into stack.



- glPopMatrix()
  - Remove the top matrix from stack

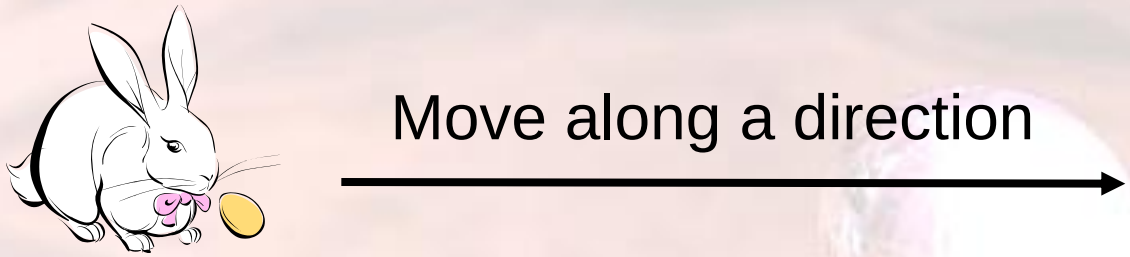
Multiple Object Rendering

Drawing each object :

```
glPushMatrix();
glTranslate()
glScale()
glBegin()
....
glEnd()
glPopMatrix();
```

Object Manipulation

- Once we select the object, we can animate that specific object.
- Object translational animation
  - Move object along a pre-defined direction.
  - Update its position periodically and redraw.
  - Change velocity based on UI.



Object Manipulation

- Animation Example

Init :
m_T = Vec3(0,0,0);
m_V = Vec3(0,0,1);
m_Speed = 1.0;

Timer :
m_T += m_V*m_Speed

Change Speed :
m_Speed ++ (or --)
Awkward for flying a plane

Rendering :

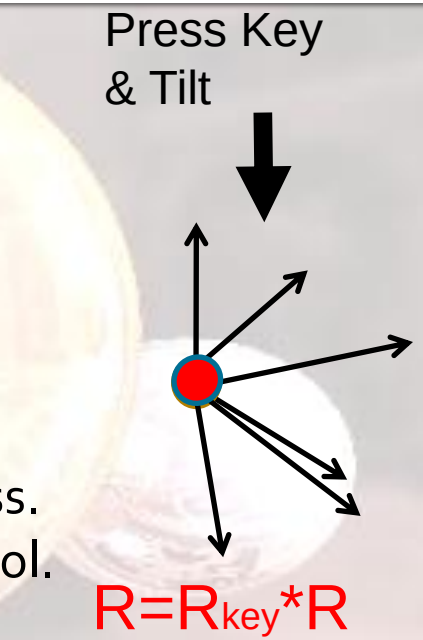
```
glPushMatrix();
glTranslate(m_T);
glBegin();
....
glEnd();
glPopMatrix();
```

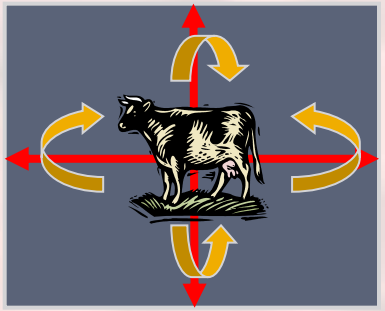
Object Manipulation

- Move object along a fixed direction is not enough.
- Rotate the object to change its moving direction.
- Problem :
  - What kind of UI to use ?
    - Keyboard ? Mouse ?
  - How to rotate its moving direction ?

Object Manipulation

- Choice of UI ?
  - Key requirements : Must be able to orient object to any directions.
  - Rotation about only one fixed axis won't get full credit.
- Keyboard
  - Change the angle/axis of rotation based on key-press.
  - Analogy to flight simulator → 3rd Person view control.
  - Keep track a total accumulated rotations.
- Mouse
  - Make use of mouse movement to define a rotation.
  - Sounds familiar ? → Euler's Angle, Arcball, etc.





Object Manipulation

- How to re-orient object ?
- Maintain a model rotation matrix.
  - Change its values based on user input.
  - Object also needs to be rotated accordingly.
  - Apply the rotation for both
    - Velocity vector
    - Object model transformation

Object Manipulation

- Re-orientation Example

Init :

m_Rot = Identity

m_InitV = m_V = Vec3(0,0,1)

UI Input :

Determine fAngle,vAxis

Mat4 newR = (fAngle,vAxis);

m_Rot = newR*m_Rot;

Update Orientation

m_V = m_Rot*m_InitV;

m_T += m_V*m_Speed

Rendering :

glPushMatrix();

glTranslate(m_T);

glMultMatrix(m_Rot);

glBegin();

….

glEnd();

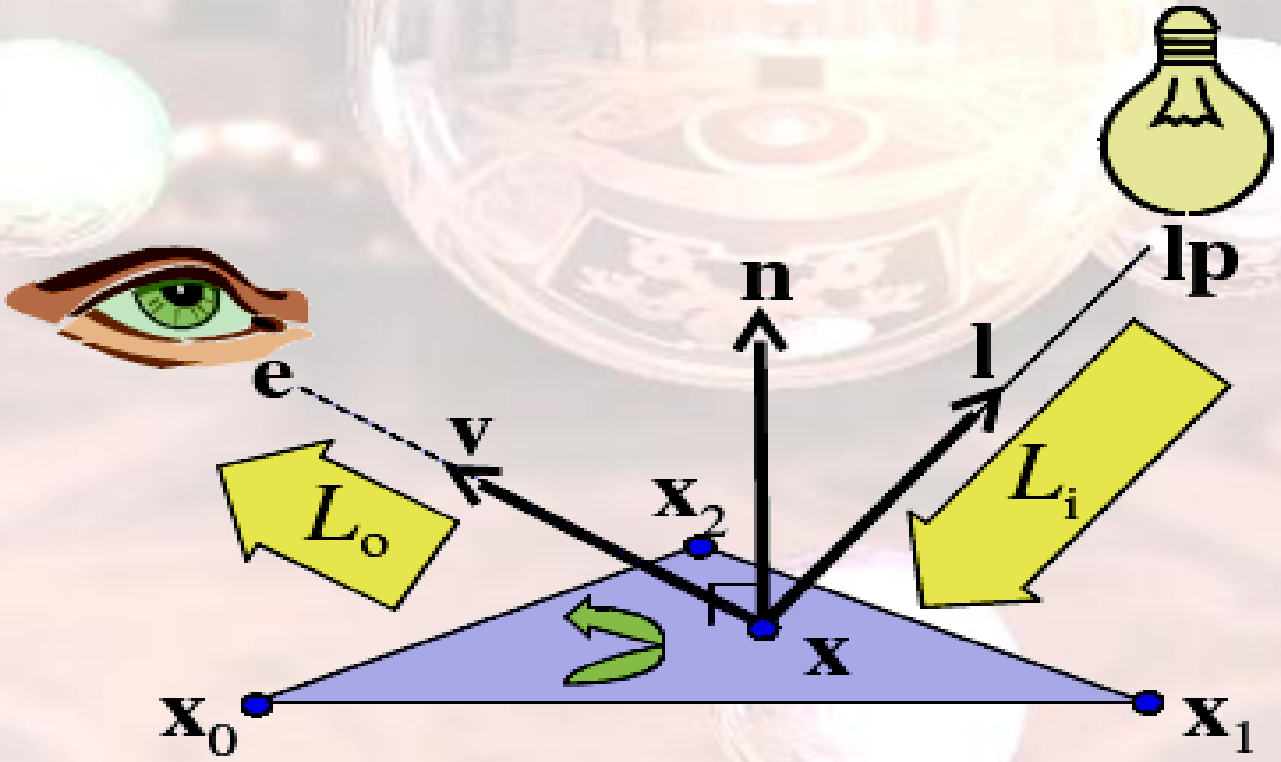
glPopMatrix();

Lighting

v – view vector: $\mathbf{v} = (\mathbf{e} - \mathbf{x})/\|\mathbf{e} - \mathbf{x}\|$

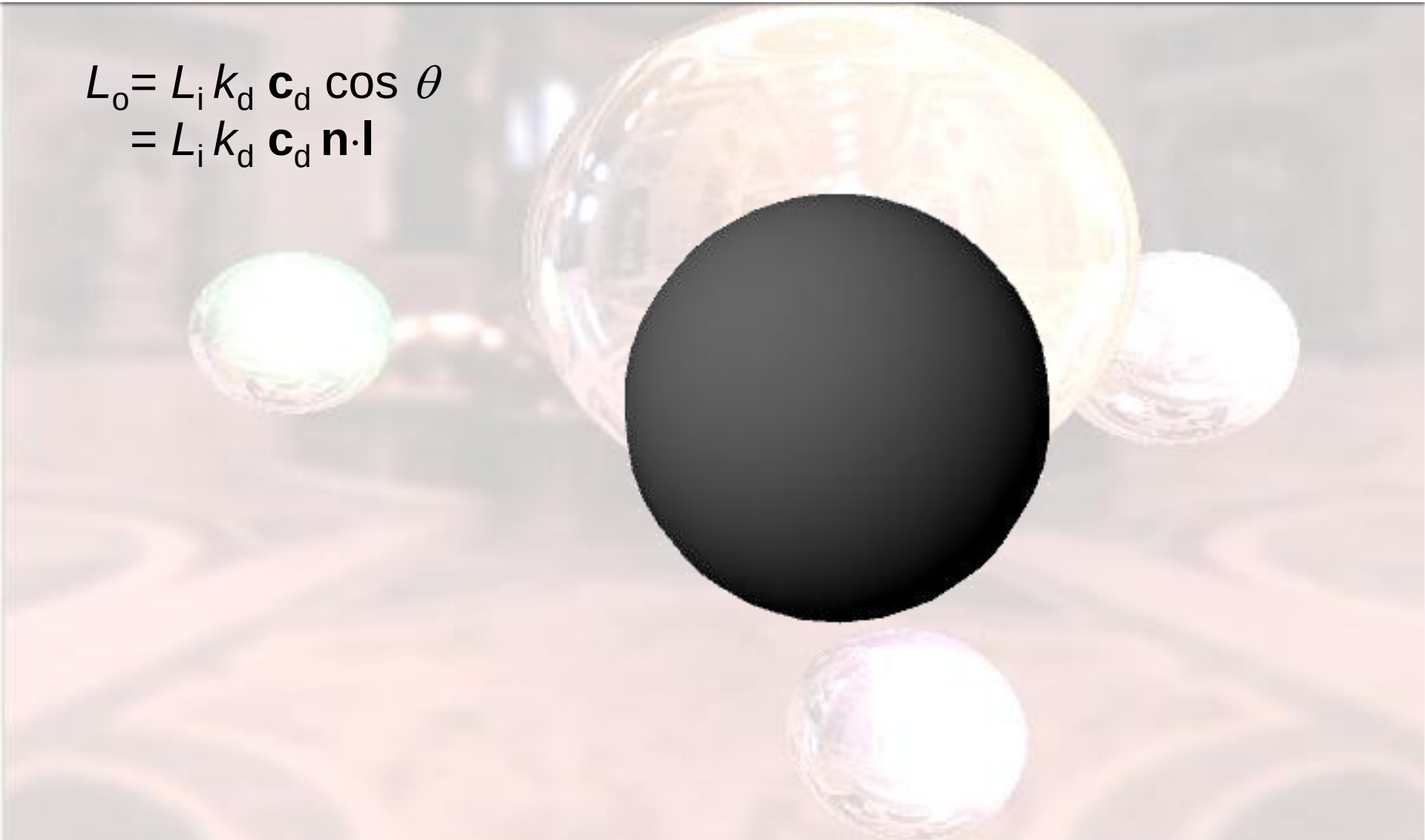
l – light vector: $\mathbf{l} = (\mathbf{lp} - \mathbf{x})/\|\mathbf{lp} - \mathbf{x}\|$

n – normal vector: $\mathbf{n} = (\mathbf{x}_1\text{-}\mathbf{x}_0)\times(\mathbf{x}_2\text{-}\mathbf{x}_0)/\|(\mathbf{x}_1\text{-}\mathbf{x}_0)\times(\mathbf{x}_2\text{-}\mathbf{x}_0)\|$



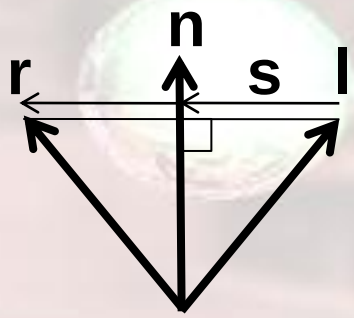
Lambertian (Diffuse)

$$L_o = L_i k_d \mathbf{c}_d \cos\theta$$
$$= L_i k_d \mathbf{c}_d \mathbf{n}\cdot\mathbf{l}$$



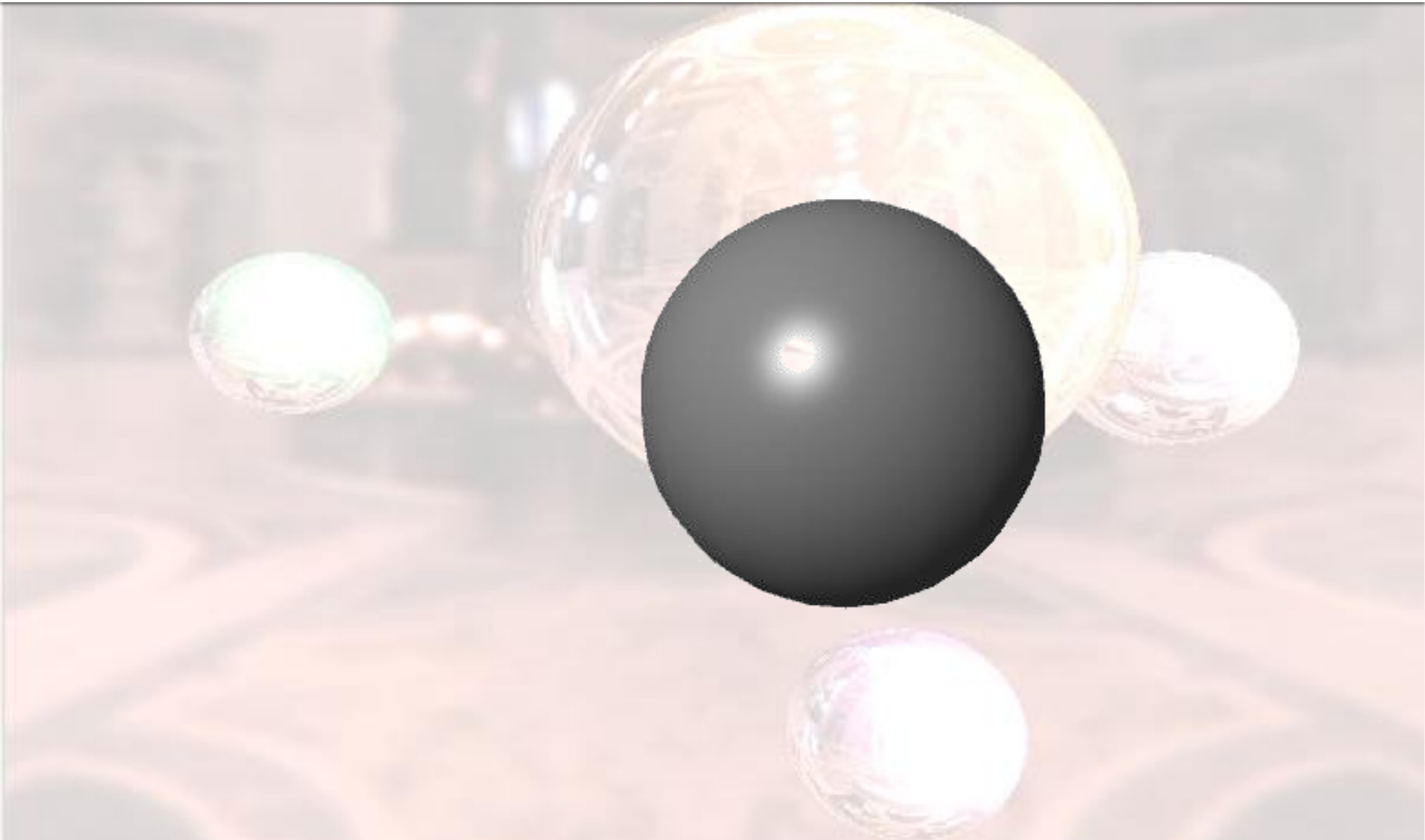
Specular Reflection

$$L_o = L_i k_s \mathbf{c}_s \cos^n \phi$$
$$= L_i k_s \mathbf{c}_s (\mathbf{v}\cdot\mathbf{r})^n$$



$$\mathbf{s} = (\mathbf{n}\cdot\mathbf{l})\mathbf{n} - \mathbf{l}$$
$$\mathbf{r} = \mathbf{l} + 2\mathbf{s}$$
$$= \mathbf{l} + 2(\mathbf{n}\cdot\mathbf{l})\mathbf{n} - 2\mathbf{l}$$
$$= 2(\mathbf{n}\cdot\mathbf{l})\mathbf{n} - \mathbf{l}$$

Ambient

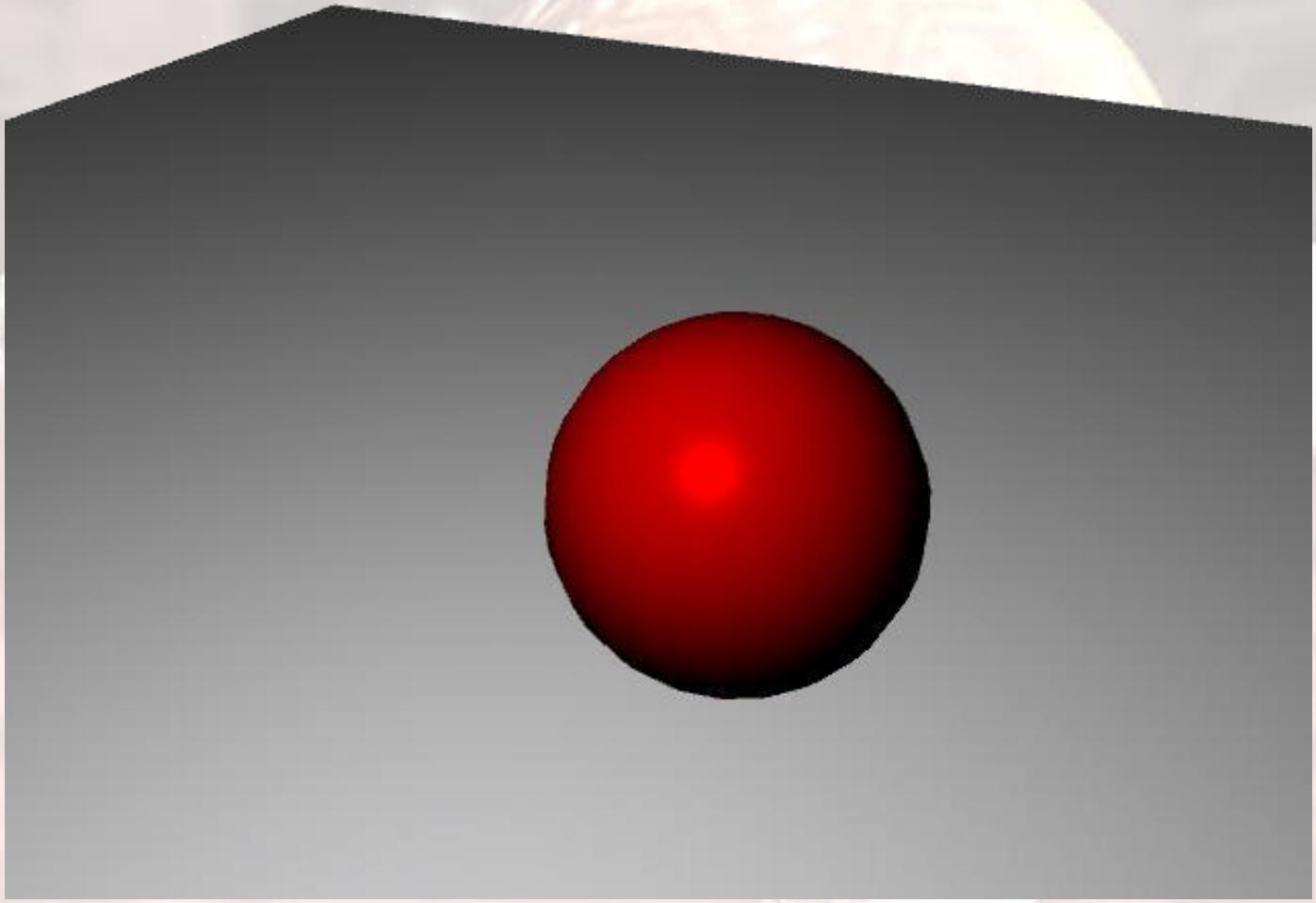


OpenGL Lighting

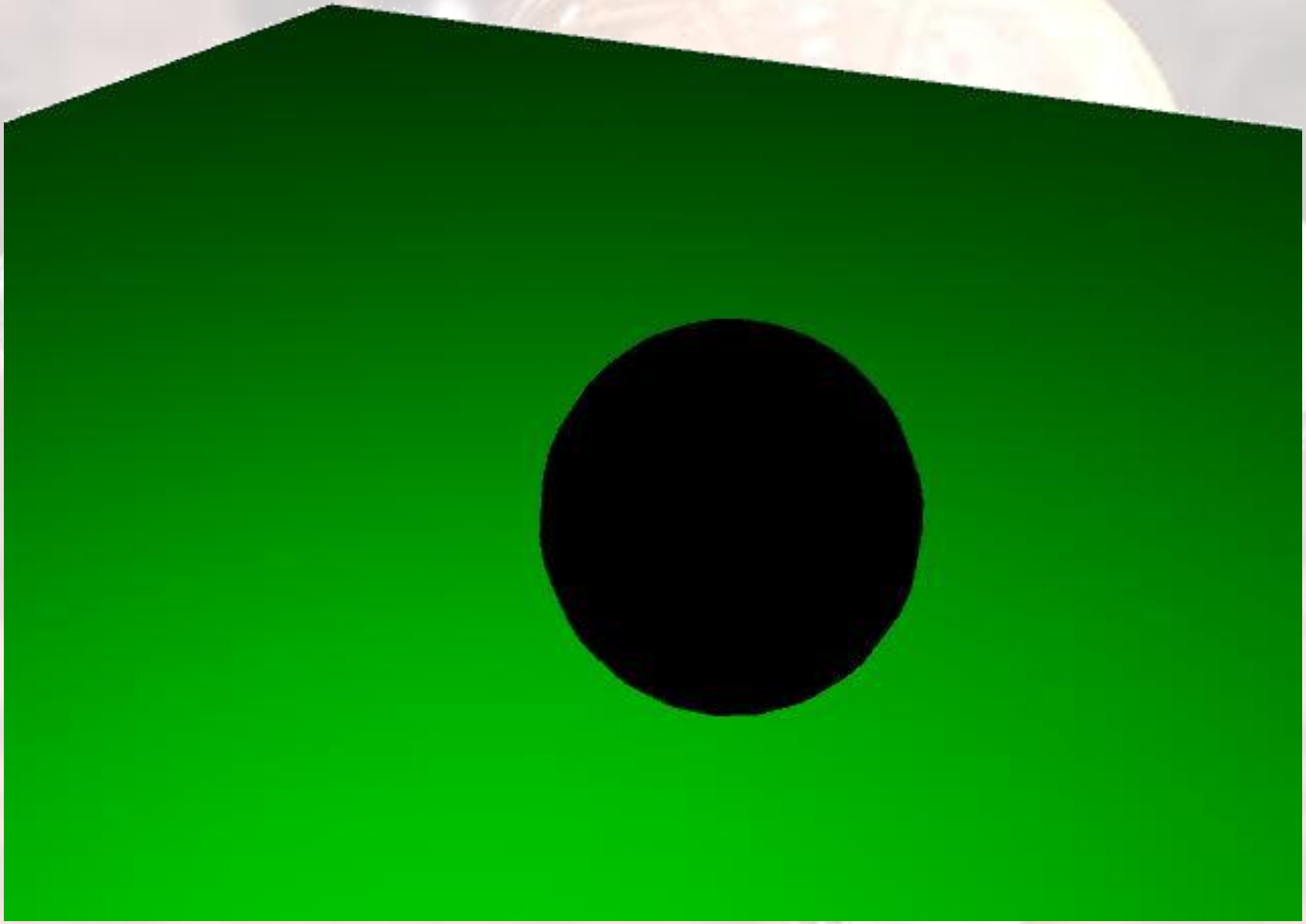
OpenGL: $$L_o = k_a L_{\#a} + L_{\#d} k_d \, \mathbf{n} \cdot \mathbf{l} + L_{\#s} k_s \, (\mathbf{v} \cdot \mathbf{r})^n$$

- Materials
  - Define the surface properties of an object (k_a, k_d, k_s)
- Lights
  - Define the properties of the emitted light (L#a, L#d, L#s)

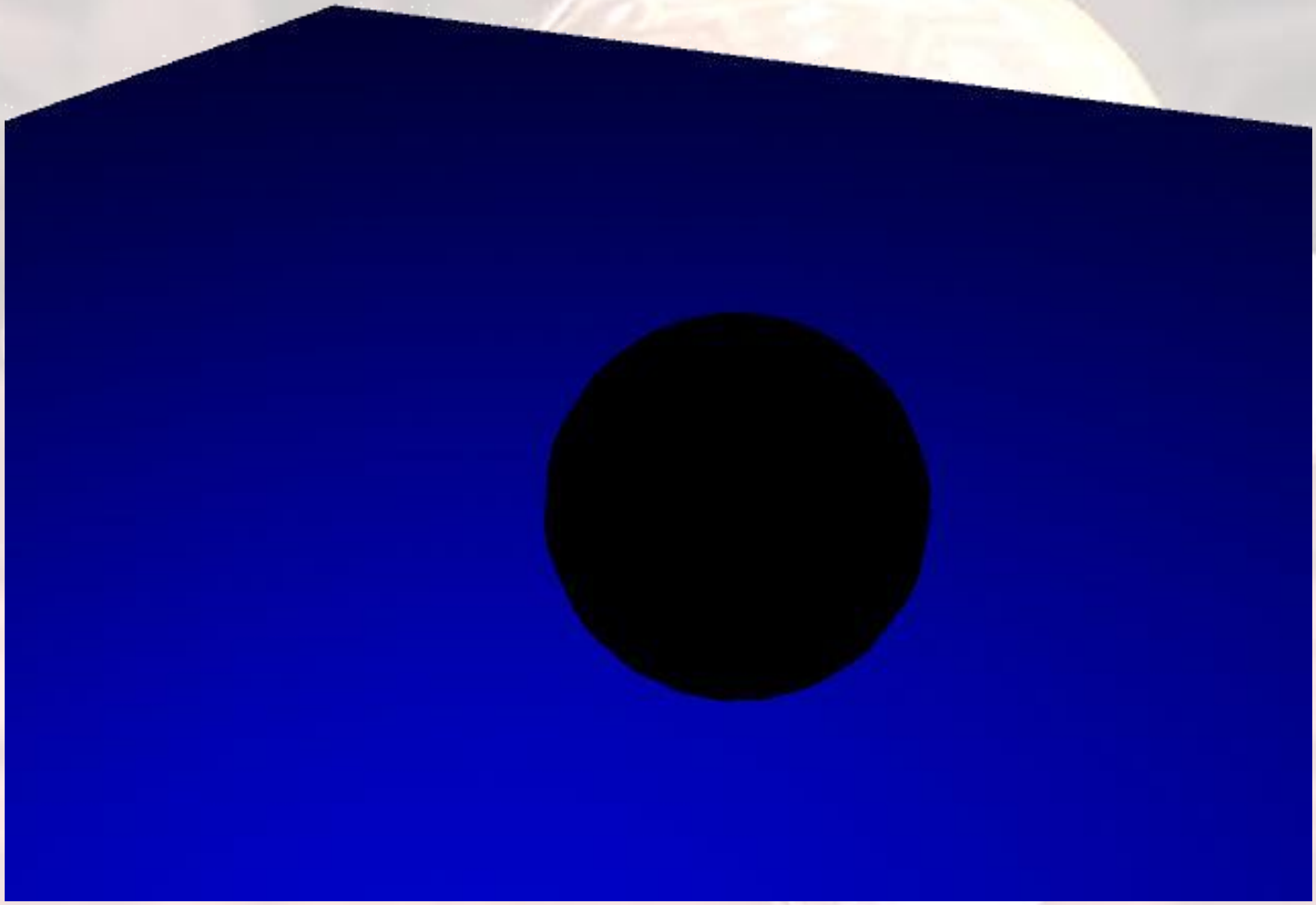
Red Ball + White Light



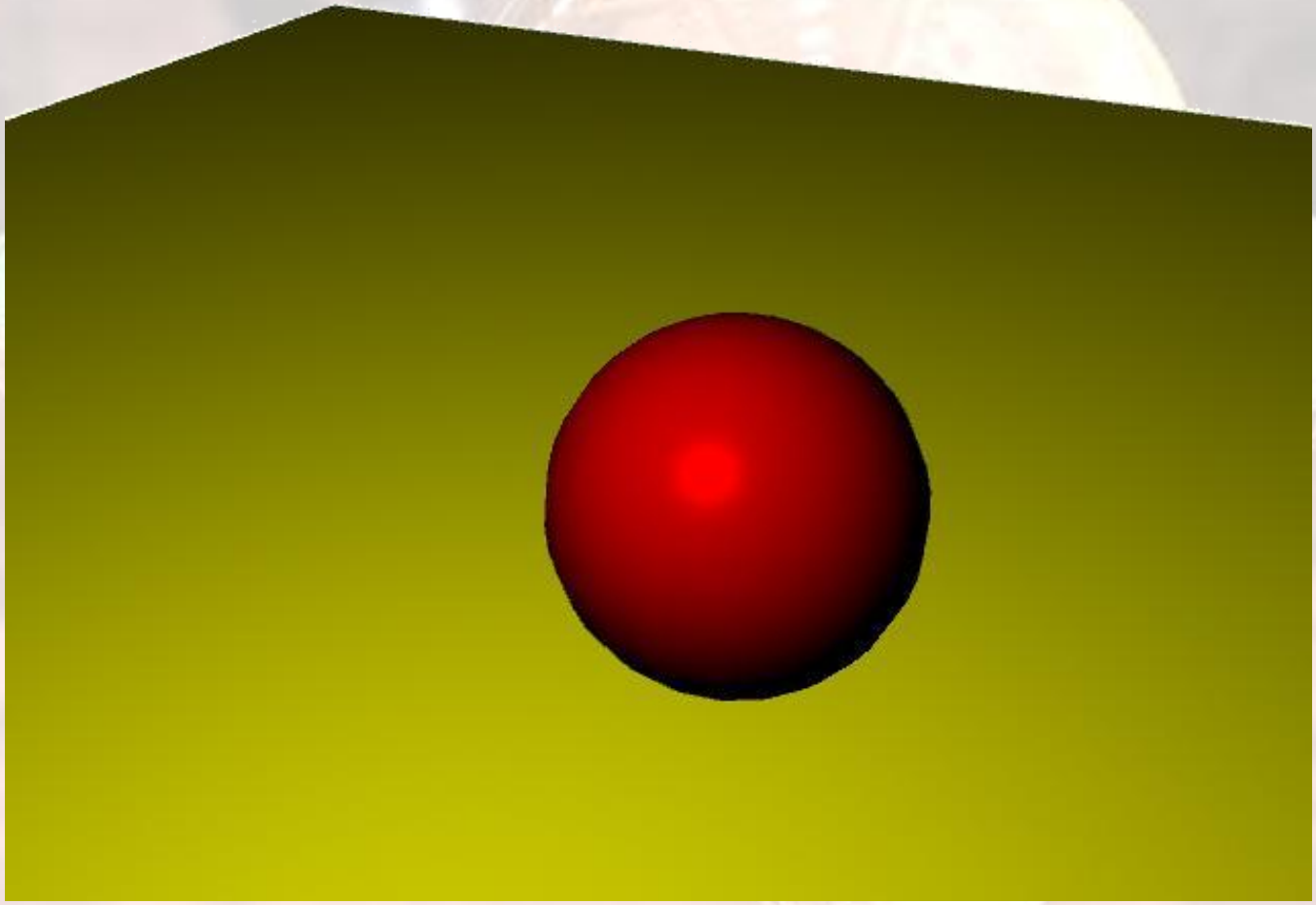
Red Ball + Green Light



Red Ball + Blue Light



Red Ball + Yellow Light



Lighting

- Define the surface properties of a primitive
  - `glMaterialfv( face, property, value );`
  - Follow Phong lighting model for parameters
  - You can define different material for front/back faces.

GL_DIFFUSE	Base color
GL_SPECULAR	Highlight Color
GL_AMBIENT	Low-light Color
GL_EMISSION	Glow Color
GL_SHININESS	Surface Smoothness

OpenGL Materials

```
GLfloat mat_ambient[] = { 0.1, 0.1, 0.1, 1.0 };
GLfloat mat_diffuse[] = { 0.8, 0.8, 0.8, 1.0 };
GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat mat_shininess[] = { 50.0 };
```

```
glMaterialfv(GL_FRONT,GL_AMBIENT, mat_ambient)
glMaterialfv(GL_FRONT,GL_DIFFUSE, mat_diffuse)
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular)
glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
```

Lighting

- Define light source property
  - OpenGL support at least 8 lights. (GL_LIGHTo ~ GL_LIGHTn)
  - `glLightfv( light, property, value );`
  - ***light*** specifies which light source
  - OpenGL light can emit different colors for each material property
  - Lighting type
    - Point light
    - Directional light
    - Spot light

GL_DIFFUSE	Base color
GL_SPECULAR	Highlight Color
GL_AMBIENT	Low-light Color

Lighting

- Change Light Type : Set related properties in
  `glLightfv( light, property, value );`
- Point Light
  - Set position to (x,y,z,1.0)
- Directional Light
  - Set direction to (x,y,z,0)
- Spot Light
  - Set spot cut-off for point light
- Be careful when setting light positions
  - Light also get transformed by ModelView matrix.

OpenGL Lights

```
GLfloat light_ambient[] = { 0.0, 0.0, 0.0, 1.0 };
GLfloat light_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light_specular[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };

glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);}
```

Lighting

- Turn on the Light in OpenGL after setting up each light source.
- Flip each light's switch

```
glEnable( GL_LIGHTn );
```

- Turn on the power

```
glEnable( GL_LIGHTING );
```

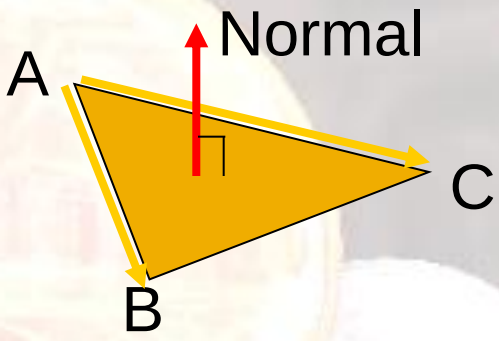
OpenGL Lighting

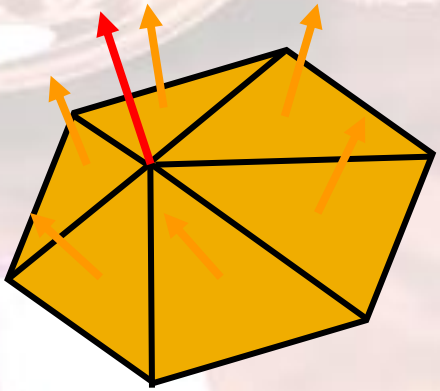
```
glShadeModel (GL_SMOOTH);
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);

glBegin(GL_POLYGON);
        glNormal3f(nx0,ny0,nz0)
        glVertex3f(x0,y0,z0);

        glNormal3f(nx1,ny1,nz1)
        glVertex3f(x1,y1,z1);

        glNormal3f(nx2,ny2,nz2)
         glVertex3f(x2,y2,z2);
glEnd();
```

Normal Computation

- Normals define how a surface reflects light

  `glNormal3f( x, y, z )`
- Face Normal :
  - Cross-product of edge vectors
- Vertex Normal :
  - Average of adjacent face normals





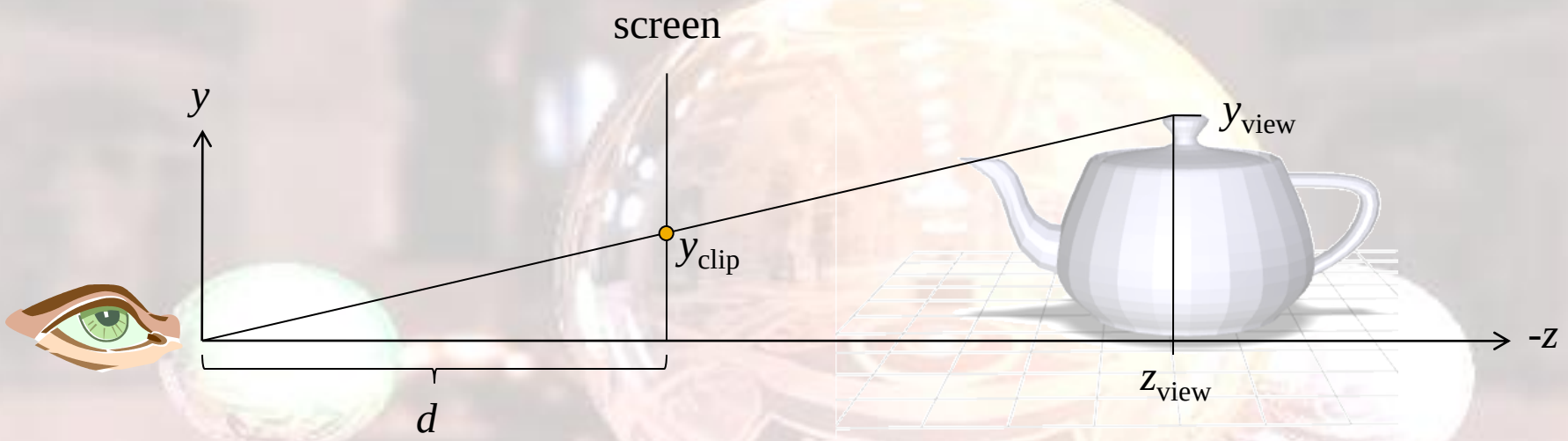
Phong Lighting Model

- Default OpenGL lighting.
- Simple model with no physical meaning.
- Usually result in a “plastic” look material.
- Cook-Torrence Demo
- Common Shading Algorithms

Exam

- Know your transformations(most points)
  - Rotation\Scale\Translate
  - Matrix Mutiplication
  - Viewing\Perspective…..
- Changing between coordinate systems
- Lighting(Applying Lighting Formulas)
- Basic vector math
  - Find normal/cross product
  - Find unit vector

Linear Perspective



$$\frac{y_{clip}}{d} = \frac{y_{view}}{-z_{view}}$$

$$y_{clip} = \frac{y_{view}}{-z_{view}/d}$$

$$\Rightarrow \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & -1/d & 0 \end{bmatrix} \begin{bmatrix} x_{view} \\ y_{view} \\ z_{view} \\ 1 \end{bmatrix} = \begin{bmatrix} x_{view} \\ y_{view} \\ z_{view} \\ -z_{view}/d \end{bmatrix} \equiv \begin{bmatrix} \frac{x_{view}}{-z_{view}/d} \\ \frac{y_{view}}{-z_{view}/d} \\ -d \\ 1 \end{bmatrix}$$