Computer Graphics Programming I

- Agenda:
  - Turn in assignment #1
  - Quiz #1!!!
  - Projections
  - Lighting
    - Lighting models
    - Lights
    - Materials
    - Shading models
  - Projected shadows

Projections

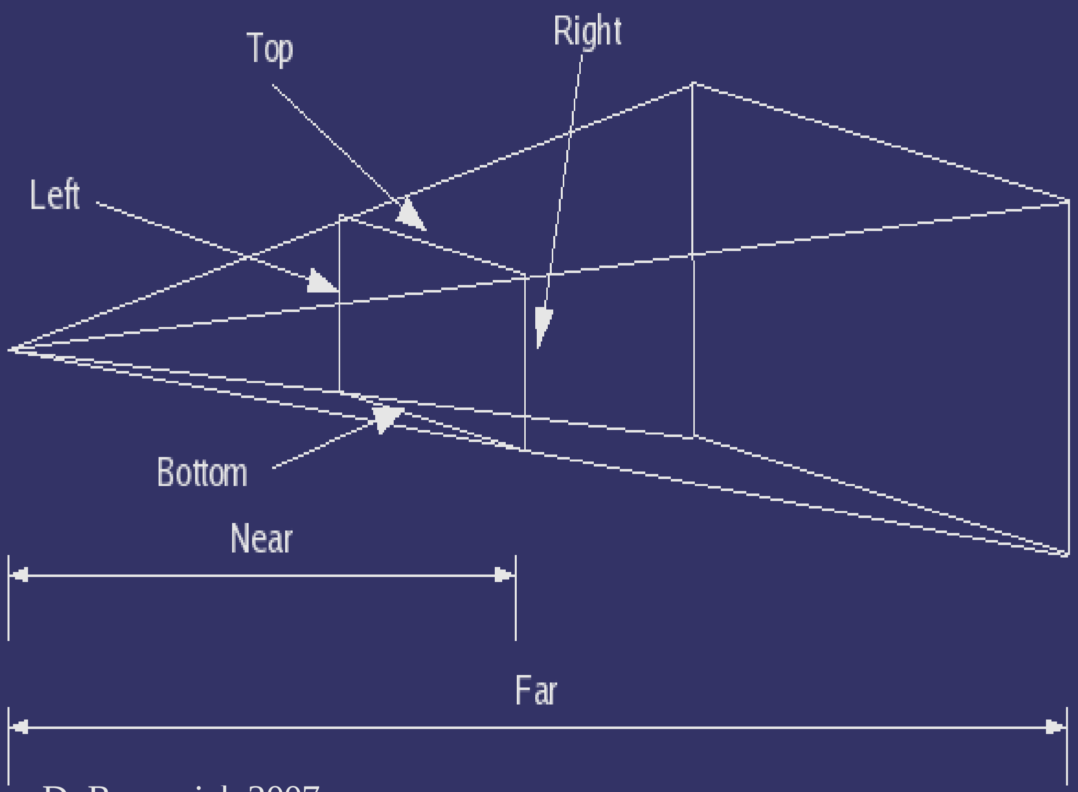
- After the modelview matrix is applied, points are still 3D.
  - The screen is 2D.
  - Camera parameters (e.g., field of view) need to be applied.
- Two steps remain.
  1) Apply the `GL_PROJECTION` matrix.
  2) Perform the perspective divide.

Types of Projection

- Perspective
  - Simulates visual foreshortening caused by the way light projects onto the back of the eye.
  - Represents the view volume with a frustum (a pyramid with the top cut off).
  - The real work is done by dividing X and Y by Z.
- Orthographic
  - Represents the view volume with a cube.
  - Also called *parallel projection* because lines that are parallel before the projection remain parallel after.

Creating Perspective Projections

- Use `glFrustum` to specify the corners of the projection plane and the distance to the near and far planes.
  - The size of the plane and the distance to the plane implicitly determine the field of view.

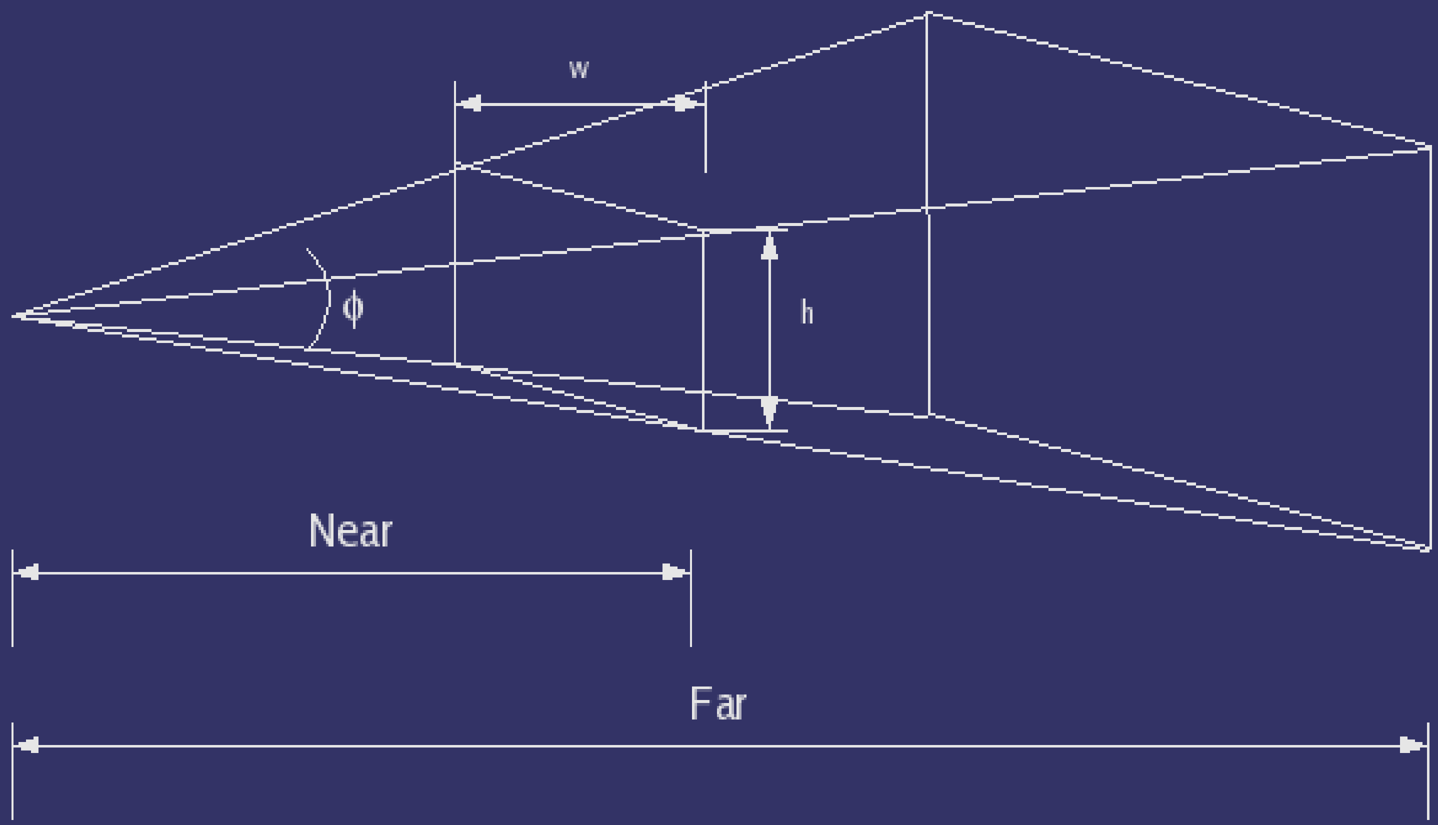


Creating Perspective Projections (cont.)

- `gluPerspective` explicitly sets the field of view.

$$aspect = \frac{w}{h}$$

$$fovy = \theta$$



References

http://en.wikipedia.org/wiki/3D_projection (esp. Third step: perspective transform).

http://en.wikipedia.org/wiki/Orthographic_projection_%28geometry

http://en.wikipedia.org/wiki/Isometric_projection

Lighting in 3D

- Three types of reflection that we usually care about.

Lighting in 3D

- Three types of reflection that we usually care about.
  - Ambient – Most “fake” of the three. Approximates scattered, omnidirectional light in the scene.

Lighting in 3D

- Three types of reflection that we usually care about.
  - Ambient – Most “fake” of the three. Approximates scattered, omnidirectional light in the scene.
  - Diffuse – Represents light scattered uniformly by tiny microfacets on the surface.

Lighting in 3D

- Three types of reflection that we usually care about.
  - Ambient – Most “fake” of the three. Approximates scattered, omnidirectional light in the scene.
  - Diffuse – Represents light scattered uniformly by tiny microfacets on the surface.
  - Specular – Perfect, mirror-like reflection from a surface.

Ambient

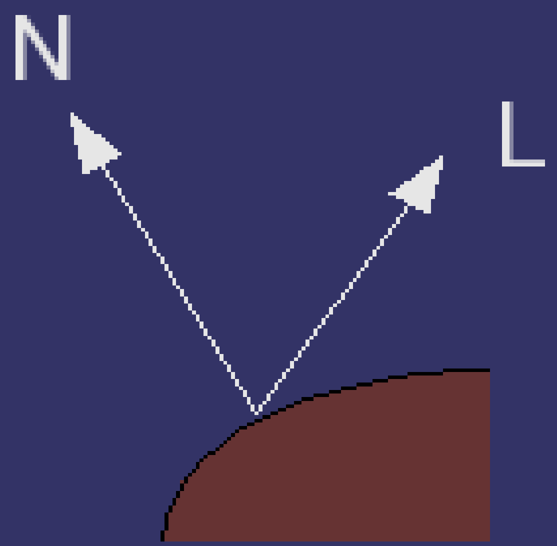
- Sets the base light level in the scene.
  - I_a is the intensity of the reflected ambient light.
  - K_a is the ambient reflection property of the surface.
  - L_a is the ambient light level in the scene.

$$I_a = K_a \times L_a$$

Diffuse

- Occurs when light hits a surface and is scattered equally in all directions.
  - Accounts for most of the lighting in the scene.
  - Also called “Lambertian reflection” because it is based on Lambert's Cosine Law.
  - Calculated for *each* light.
  - Independent of viewing direction.

$$I_d = K_d \times L_d \times max(L \cdot N, 0)$$



Lambert's Lighting Model

- What makes the equation work?

$$I_d = K_d \times L_d \times max(L \cdot N, 0)$$

Lambert's Lighting Model

- What makes the equation work?

$$I_d = K_d \times L_d \times max(L \cdot N, 0)$$

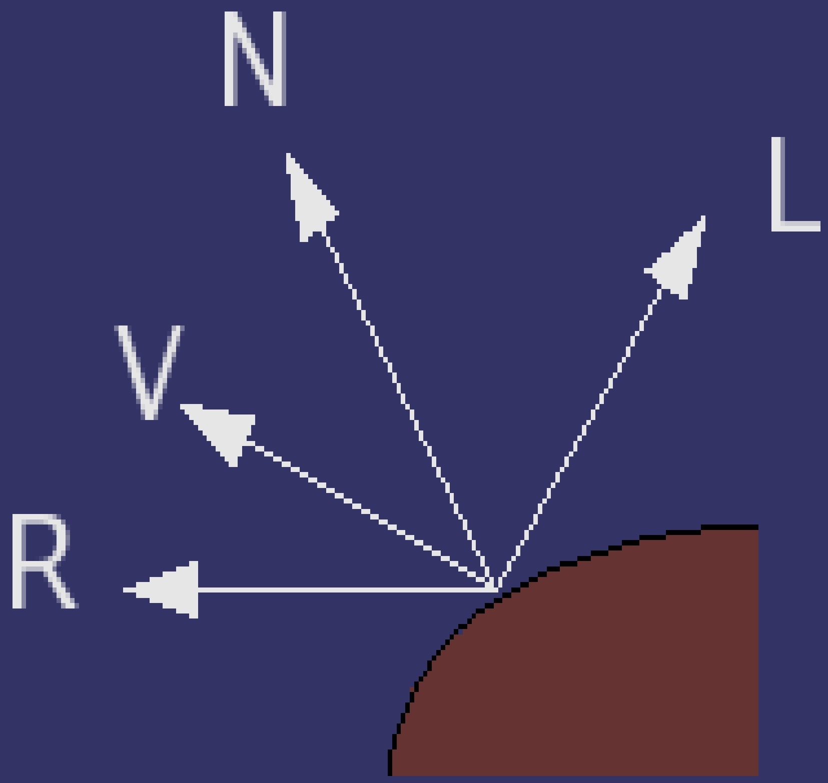
- As the angle between the normal and the light decreases, the amount of light reflected by the surface increases.
  - The more directly the surfaces faces the light, the more light hits the surface.

Specular

- Light is reflected from a surface primarily in one direction.
  - Observed light intensity depends on viewing direction.
  - Developed by Bui-Tuong Phong in 1973.
  - *R* is "ideal" reflection vector.
  - Very expensive!

$$R = 2(N \cdot L)N - L$$

$$I_s = K_s \times L_s \times max(R \cdot V, 0)^n$$



Phong's Lighting Model

- What makes the equation work?

$$R = 2(N \cdot L)N - L$$

$$I_s = K_s \times L_s \times max(R \cdot V, 0)^n$$

Phong's Lighting Model

- What makes the equation work?

$$R = 2(N \cdot L)N - L$$

$$I_s = K_s \times L_s \times max(R \cdot V, 0)^n$$

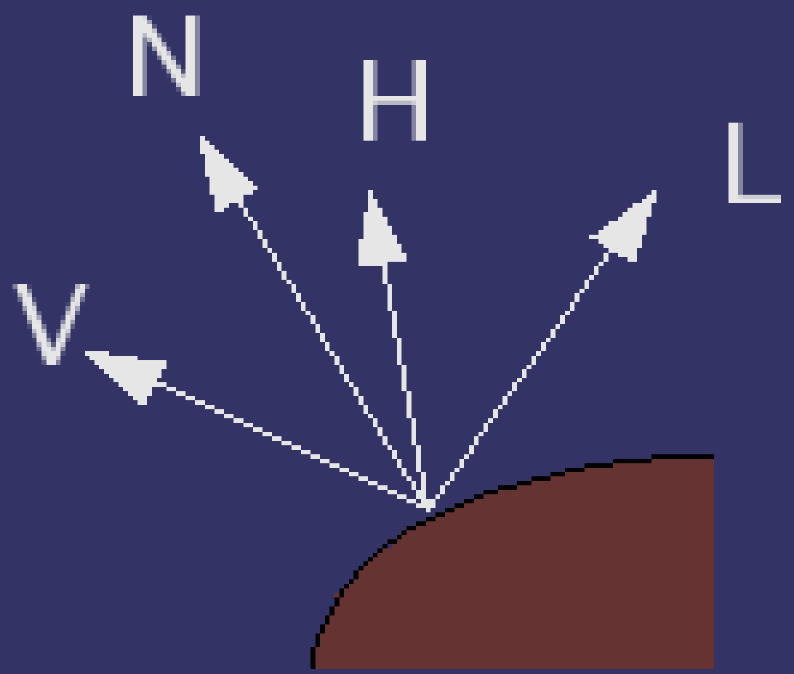
- As the angle between the ideal reflection vector, *R*, and the viewer, *V*, decreases, the light becomes more intense.

Improved Specular

- James Blinn improved Phong's model in 1977.
  - *Much* less expensive.
  - Slightly different results.
    - Both are approximations!
  - Lighting model used by OpenGL.
  - *H* is the vector half-way between the light and viewer.

$$H = \frac{(V + L)}{2}$$

$$I_s = K_s \times L_s \times max(N \cdot H, 0)^n$$



Blinn's Lighting Model

- What makes the equation work?

$$H = \frac{(V + L)}{2}$$

$$I_s = K_s \times L_s \times max(N \cdot H, 0)^n$$

Blinn's Lighting Model

- What makes the equation work?

$$H = \frac{(V + L)}{2}$$

$$I_s = K_s \times L_s \times max(N \cdot H, 0)^n$$

- Key observation is that *H* approaches *N* when *V* approaches *R* (Phong's ideal reflection vector).

Shininess

- What is the magic "n" factor in both equations?

$$R = 2(N \cdot L)N - L$$
$$I_s = K_s \times L_s \times max(R \cdot V, 0)^n$$

$$H = \frac{(V + L)}{2}$$
$$I_s = K_s \times L_s \times max(N \cdot H, 0)^n$$

Shininess

- What is the magic “n” factor in both equations?
  - Controls the “size” of the specular highlight.
  - As *n* increases, the highlight gets smaller.
    - Because the result of the dot-product factor gets smaller faster.

$$R = 2(N \cdot L)N - L$$
$$I_s = K_s \times L_s \times max(R \cdot V, 0)^n$$

$$H = \frac{(V + L)}{2}$$
$$I_s = K_s \times L_s \times max(N \cdot H, 0)^n$$

References

http://www.delphi3d.net/articles/viewarticle.php?article=phong.htm

http://en.wikipedia.org/wiki/Lambertian_reflectance

Break

Controlling Lights in OpenGL

- OpenGL lights are named `GL_LIGHT0` through `GL_LIGHT7`.
  - `GL_LIGHT0` + 3 has the same numeric value as `GL_LIGHT3`.
- Light parameters are set via `glLight`.

```
glLightf(GLenum light, GLenum param, GLfloat
  value);
glLightfv(GLenum light, GLenum param, const
  GLfloat *values);
```

  - `values` points to either 1 or 4 elements depending on `param`.

Controlling Lights in OpenGL (cont.)

- Lighting calculations need to be enabled.
- Each light also needs to be enabled.
  - Both are done with `glEnable`.
  - Each can be disabled with `glDisable`.

Example

```
void setup_lights(void)
{
    glEnable(GL_LIGHTING);

    glEnable(GL_LIGHT0);
    glLightfv(GL_LIGHT0, GL_AMBIENT, ambient0);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse0);
    glLightfv(GL_LIGHT0, GL_SPECULAR, specular0);
    glLightfv(GL_LIGHT0, GL_POSITION, position0);

    glEnable(GL_LIGHT1);
    glLightfv(GL_LIGHT1, GL_AMBIENT, ambient1);
    glLightfv(GL_LIGHT1, GL_DIFFUSE, diffuse1);
    glLightfv(GL_LIGHT1, GL_SPECULAR, specular1);
    glLightfv(GL_LIGHT1, GL_POSITION, position1);
}
```

Lights are transformed too!

- The current modelview matrix *when the light's position is set* is used to transform the light.

```
glLightfv(GL_LIGHT0, GL_POSITION, position);
glRotatef(angle, 0.0, 0.0, 1.0);
glTranslatef(dist_x, dist_y, dist_z);
glLightfv(GL_LIGHT1, GL_POSITION, position);
```

  - Light 0 and light 1 will be at different positions!

Visualizing a Light

- Drawing a point at the light's position can help debug lighting problems.

```
set_light_transform();
glLightfv(light_name, GL_POSITION, light_pos);

glDisable(GL_LIGHTING);
glPointSize(5.0);
glBegin(GL_POINTS);
glColor3ub(0xff, 0xff, 0x00);
glVertex3fv(light_pos);
glEnd();

glEnable(GL_LIGHTING);
```

Surface Material Properties

- `glMaterial[fi][v]` is used to control attributes of the surface.
  - *Cannot* be called within begin / end.

```
glMaterialfv(GL_FRONT, GL_AMBIENT, Ka);
glMaterialfv(GL_FRONT, GL_DIFFUSE, Kd);
glMaterialfv(GL_FRONT, GL_SPECULAR, Ks);
glMaterialf(GL_FRONT, GL_SHININESS, n);
```

  - Can set different parameters for the front and back sides of a surface.

Scaling = Trouble

- Normals get transformed by the inverse transpose of the modelview matrix.
  - Really, this is just the upper 3x3 portion...without the translation part.
- If the modelview matrix has a scaling factor, the normals will also get scaled.
  - So?

Scaling = Trouble

- Normals get transformed by the inverse transpose of the modelview matrix.
  - Really, this is just the upper 3x3 portion...without the translation part.
- If the modelview matrix has a scaling factor, the normals will also get scaled.
  - So?
  - $N \cdot L = \cos\theta$ only works if *N* and *L* are unit length, and scaling ruins that.

Scaling = Trouble (cont.)

- OpenGL has two ways to fix this.
  - If the original normals are unit length and the scaling is uniform (i.e., $S_x = S_y = S_z$) enable `GL_RESCALE_NORMAL`.
  - In all other cases, enable `GL_NORMALIZE`.
- Neither is free, but `GL_RESCALE_NORMAL` is much less expensive.

Scaling = Trouble (cont.)

- OpenGL has two ways to fix this.
  - If the original normals are unit length and the scaling is uniform (i.e., $S_x = S_y = S_z$) enable `GL_RESCALE_NORMAL`.
  - In all other cases, enable `GL_NORMALIZE`.
- Neither is free, but `GL_RESCALE_NORMAL` is much less expensive.
  - Analyze transformation matrix and calculate the inverse scale factor once, perform 1 vector multiply per point vs. a dot-product, a square root, and a divide per point.

Light Source Attenuation

- Real lights don't have infinite range.
  - Objects farther away receive less light energy.
- Three different attenuation modes in OpenGL:
  - GL_CONSTANT_ATTENUATION (k_c)
  - GL_LINEAR_ATTENUATION (k_l)
  - GL_QUADRATIC_ATTENUATION (k_q)

$$attenuation = \frac{1}{k_c + k_l \times d + k_q \times d^2}$$

Spot Lights

- Most real lights have a direction and a “field of view”.
  - Objects outside the field of view receive no light.
  - Objects far from the direction receive less light.
    - Works like diffuse lighting, but instead of $N \cdot L$, we use $-L \cdot D$ (*D* the direction the light is pointing).
- Controlled by 3 parameters:
  - `GL_SPOT_DIRECTION`
  - `GL_SPOT_CUTOFF` – 180° is a point light
  - `GL_SPOT_EXPONENT` – works like *n* for specular

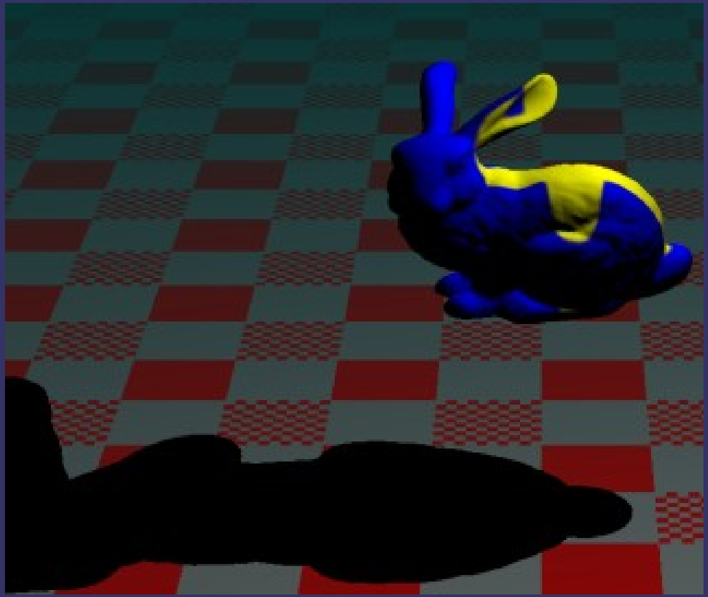
Shading Models

- Three common shading models:
  - Flat – each polygon gets a single color value.
  - Gouraud – each vertex gets a color, and those colors are interpolated across the polygon.
  - Phong – Vertex properties (i.e., normals) are interpolated across the polygon and lighting is calculated per fragment.
- The first two can be selected in OpenGL by via `glShadingModel`.
  - `GL_FLAT` for flat, and `GL_SMOOTH` for Gouraud.

Break

Planar Shadows

- Simplest shadows are those projected onto a flat plane
  - As the description implies, this can be done using a projection matrix



Plane equation

- Give a point on a plane, *p*, and the normal of that plane, *n*, calculate the plane equation:

$$d = -(n \cdot p)$$

$$n \cdot p_i + d = 0$$

Projection onto a plane

- Given a plane, defined by *n* and *d,* and a projection point, *p*, create a matrix that projects an arbitrary point onto that plane.
  - Like the projection of the view plane and the eye point.

$$M = \begin{bmatrix} n\cdot p + d - p_x n_x & -p_x n_y & -p_x n_z & -p_x d \\ -p_y n_x & n\cdot p + d - p_y n_y & -p_y n_z & -p_y d \\ -p_z n_x & -p_z n_y & n\cdot p + d - p_z n_z & -p_z d \\ -n_x & -n_y & -n_z & n\cdot p \end{bmatrix}$$

Planar shadows

- If the plane is the ground plane, and the projection point is the light, *M* is a matrix that projects the shadow of world-space geometry onto the ground.
- But where do we insert *M* into the transformation stack?

Planar shadows

- If the plane is the ground plane, and the projection point is the light, *M* is a matrix that projects the shadow of world-space geometry onto the ground.
- But where do we insert *M* into the transformation stack?
  - After the object-to-world space transformations, but before the world-to-eye space transformation.

Next week...

- Using color-materials.
- Introduction to texture mapping
  - Loading texture data
  - Getting a simple texture on a polygon
- Assignment #2 due.
- Assignment #3 assigned.

Legal Statement

- This work represents the view of the authors and does not necessarily represent the view of IBM or the Art Institute of Portland.
- OpenGL is a trademark of Silicon Graphics, Inc. in the United States, other countries, or both.
- Khronos and OpenGL ES are trademarks of the Khronos Group.
- Other company, product, and service names may be trademarks or service marks of others.