Viewing

527970
Fall 2020
11/5/2020
Kyoung Shin Park
Computer Engineering
Dankook University

Camera Movement

- To give a camera movement in OpenGL, you can apply a transformation matrix opposite to the camera movement at the beginning of the display function.
- For example, to move the camera 10 units in z-axis from the origin, you can move the world by -10 units in z-axis.

```
void display()
{
    Projection = glm::perspective(45, 1, 0.1, 1000);
    View = glm::mat4(1.0f); // Identity matrix
    World = glm::translate(glm::mat4(1.0f), glm::vec3(0, 0, -10));
    drawObjects();
}
```

Camera Movement

- In general camera movement, the camera position and orientation is applied to the world as an inverse transformation matrix.
- Example

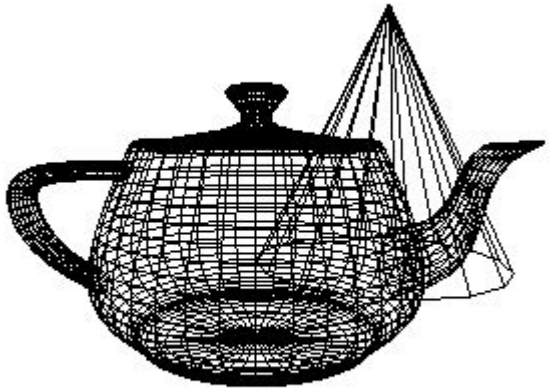
```
float cameraX, cameraY, cameraZ, cameraHeading;
void display()
{
    Projection = perspective(45, 1, 0.1, 1000);
    View = mat4(1.0f); // Identity matrix
    World = rotate(mat4(1.0f), -cameraHeading, vec3(0, 1, 0))
          * translate(mat4(1.0f), vec3(-cameraX, -cameraY, -cameraZ));
    drawObjects();
}
```

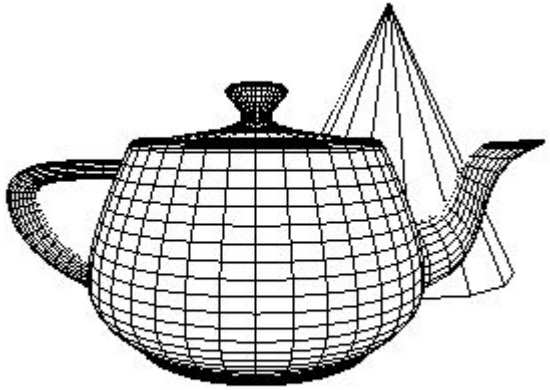
Camera Movement

- Navigation
  - Fly-through (6DOF)
    - Yaw (y), pitch (x), roll (z) orientation
    - Walk forward/backward (z), strafe right/left (x), fly up/down (y) movement
  - Walk-through (2DOF)
    - Pan (y)
    - Walk forward/backward (z)

Hidden Surface

- Hidden surfaces provides the occlusion depth cue.
- In computer graphics, the term occlusion refers to objects that are close to the viewer to occlude objects that are far from the viewer.
- In the graphics pipeline, hidden surface removal is performed before shading and rasterization with occlusion culling.





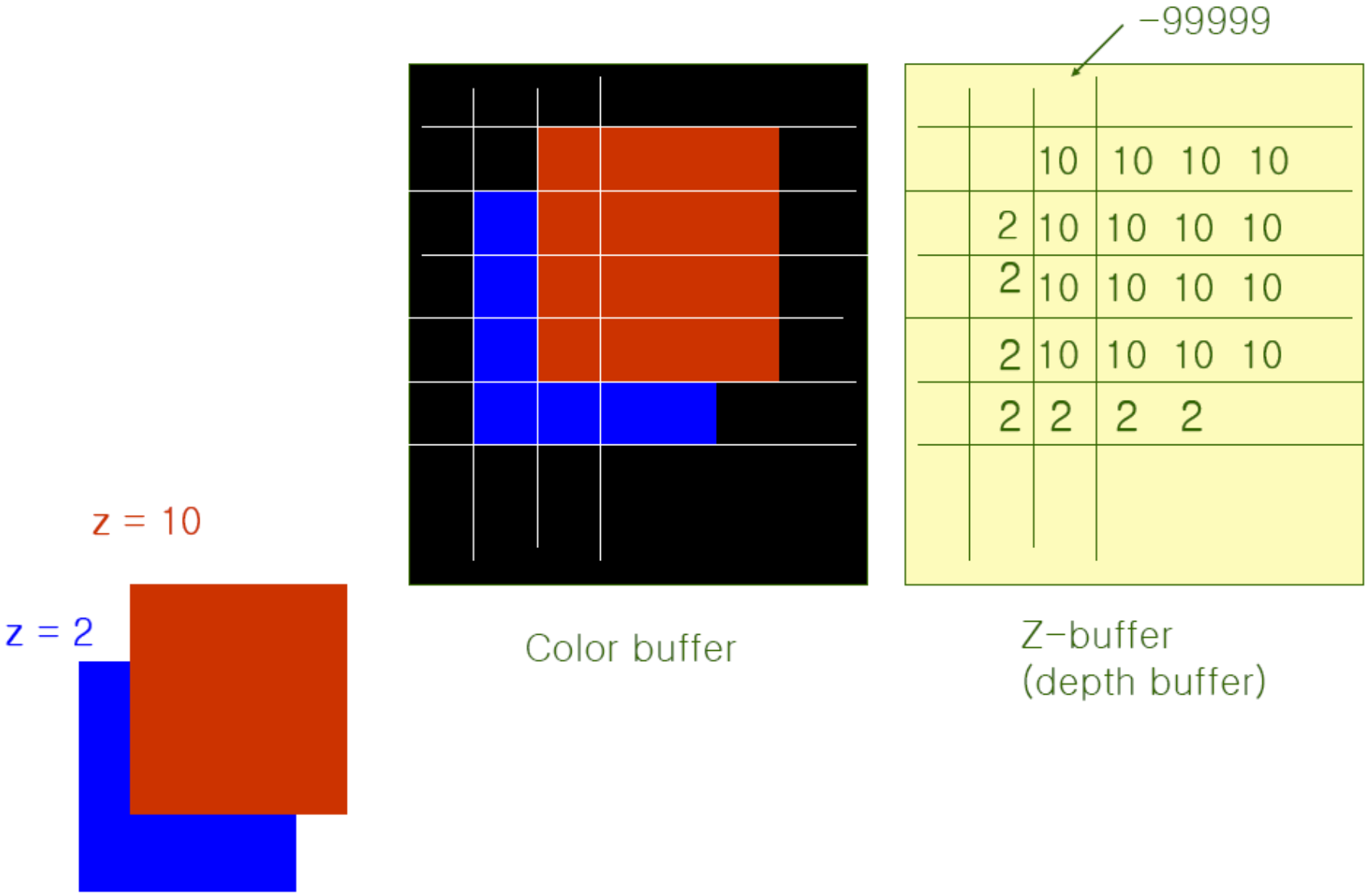
Hidden Surface Removal

- Hidden Surface Removal Algorithm
  - Object space technique – compare objects or parts of objects to determine which side and line are not visible as a whole.
    - Depth-sorting algorithm – After aligning each side of the polygon according to the depth, it is drawn from the far one to front one. Also known as Painter's algorithm.
    - Binary Space Partitioning (BSP) tree – Using BSP tree, the space is continuously partitioned by separating front and back according to the viewer direction.
  - Image space technique – act as part of the projection, and visibility is determined in units of points at the location of object pixels on each projection line.
    - Z-buffer (depth buffer) – This is the most commonly used image space technique. By examining the visibility of an object in pixels, it draws the value of the plane with the smallest z (depth) value. We need a depth buffer (z-buffer) to store the z-value.
    - Ray-casting – It projects light (ray) through each pixel on the projection surface at the viewpoint, selects the object that first meets this light and draws the pixel. It is an effective hidden surface removal algorithm for curved surface.

Hidden Surface Removal

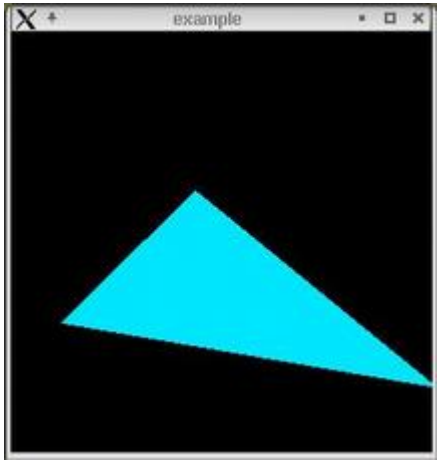
- Hidden surface removal methods
  - Depth test
    - glEnable(GL_DEPTH_TEST);
  - Cull the face/backside
    - glEnable(GL_CULL_FACE);
    - glCullFace(GL_FRONT);
    - glCullFace(GL_BACK);

Z-buffer

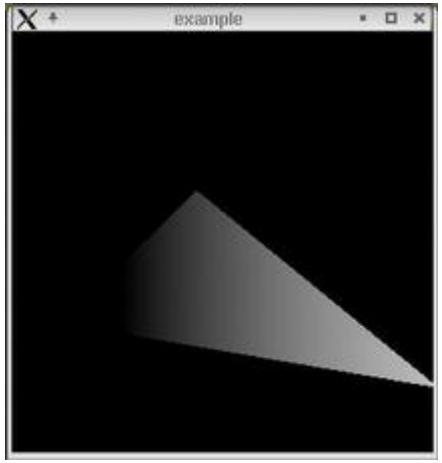

−99999
z = 10
z = 2
10 10 10 10
2 10 10 10 10
2 10 10 10 10
2 10 10 10 10
2 2 2 2
Color buffer
Z−buffer
(depth buffer)

Z-buffer

- Polygon rendering means eventually being filled with pixels.
- The color buffer contains RGB color per pixel to be drawn.
- The depth buffer (Z-buffer) has depth information per pixel to be drawn.



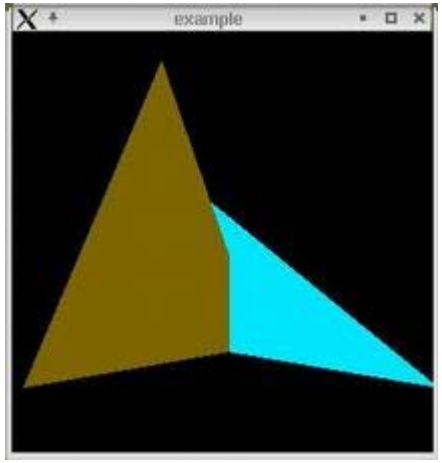
Color buffer



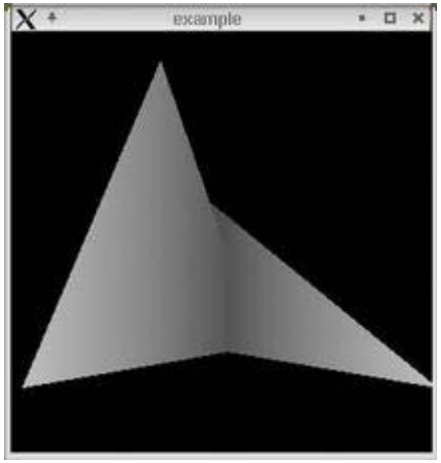
Depth buffer

Z-buffer Algorithm

- Whenever a new pixel is drawn, the Z-buffer algorithm compares the new depth information with the previous depth information in the z-buffer.
- Polygons can be drawn in any direction and can intersect.



Color buffer



Depth buffer

OpenGL Z-buffering

- To use the z-buffer in OpenGL, first initialize the depth buffer and then activate the depth test.

```
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
glEnable(GL_DEPTH_TEST);
```

- Clear the depth buffer in every frame in display function.

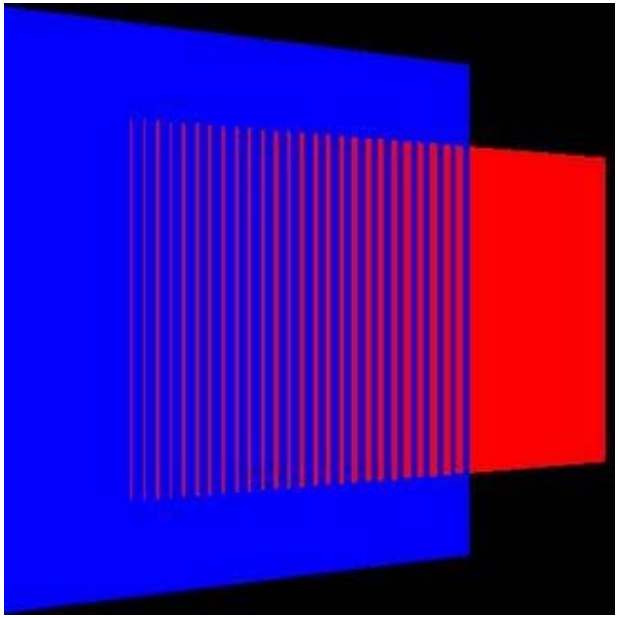
```
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
```

- Activate the culling to remove all faces facing away from the viewer.

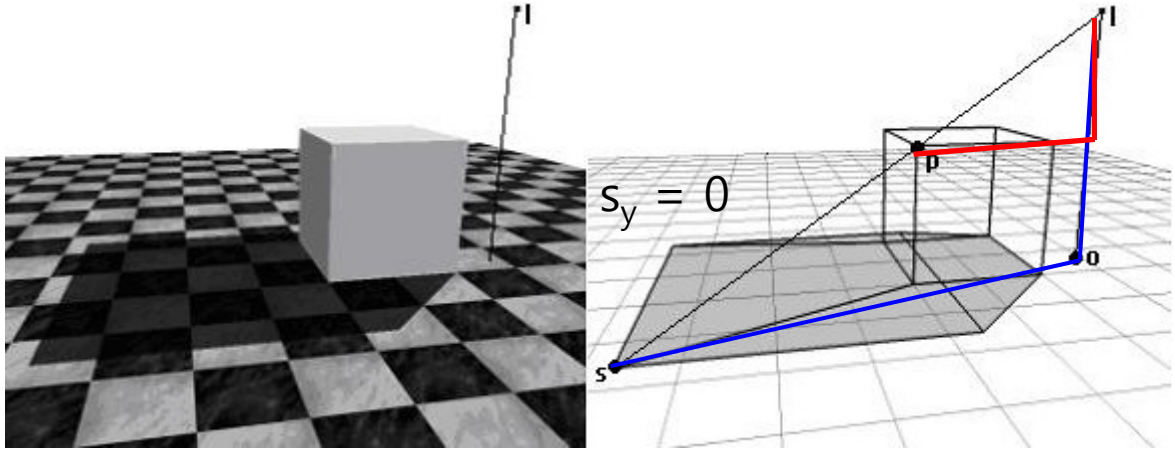
```
glEnable(GL_CULL);
```

Depth Fighting

- The depth value of the Z-buffer has a limited resolution.
- The overlap of polygons with a depth value that is very close to the depth buffer creates “depth-fighting”.
- This is a phenomenon that occurs due to “floating point round-off errors” when polygons are drawn, where random parts of polygons flight for rendering each other.



Planar Shadow [J. Blinn, 88]



$$\frac{l_y - p_y}{p_x - l_x} = \frac{l_y - 0}{\mathbf{s}_x - l_x}$$

$$\Rightarrow \frac{\mathbf{s}_x - l_x}{p_x - l_x} = \frac{l_y - 0}{l_y - p_y}$$

$$\mathbf{s}_x = \frac{l_y (p_x - l_x)}{l_y - p_y} + l_x$$

$$t = \frac{l_y}{l_y - p_y}$$

$$\begin{bmatrix} s_x \\ 0 \\ s_z \\ 1 \end{bmatrix} = \begin{bmatrix} l_y & -l_x & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -l_z & l_y & 0 \\ 0 & -1 & 0 & l_y \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}$$

$$\mathbf{s} = l + t(p - l)$$

$$\mathbf{n} \cdot \mathbf{s} + d = 0$$

$$t = \frac{l_y}{l_y - p_y}$$

Planar Shadow [J. Blinn, 88]

$$\mathbf{s} = l + \frac{l_y}{l_y - p_y}(p - l)$$

$$s_x = l_x + \frac{l_y}{l_y - p_y}(p_x - l_x)$$

$$s_y = 0$$

$$s_z = l_z + \frac{l_y}{l_y - p_y}(p_z - l_z)$$

$$s_x = l_x + \frac{l_y}{l_y - p_y}(p_x - l_x) = \frac{l_x(l_y - p_y) + l_y(p_x - l_x)}{l_y - p_y} = \frac{l_y p_x - l_x p_y}{l_y - p_y}$$

$$s_z = l_z + \frac{l_y}{l_y - p_y}(p_z - l_z) = \frac{l_z(l_y - p_y) + l_y(p_z - l_z)}{l_y - p_y} = \frac{-l_z p_y + l_y p_z}{l_y - p_y}$$

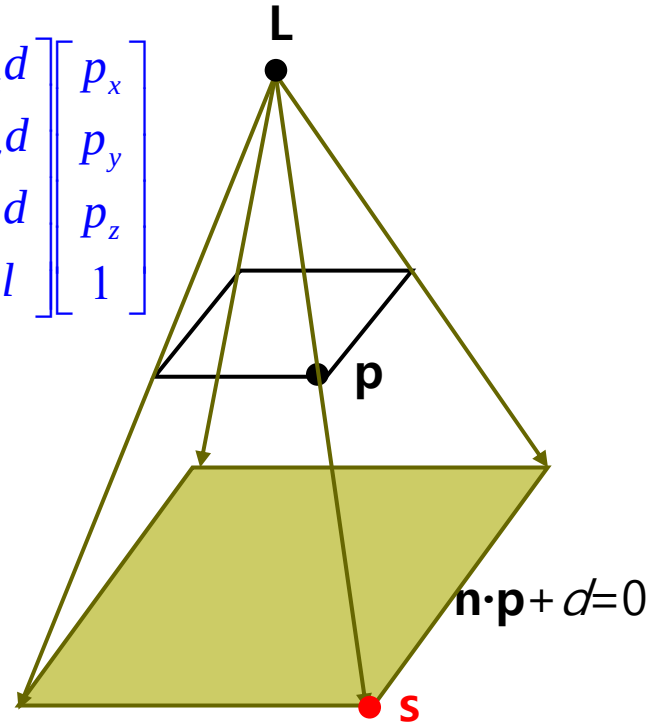
$$\begin{bmatrix} s_x \\ 0 \\ s_z \\ 1 \end{bmatrix} = \begin{bmatrix} l_y & -l_x & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -l_z & l_y & 0 \\ 0 & -1 & 0 & l_y \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}$$

$s_x = l_y p_x - l_x p_y + 0 p_z$
$s_y = 0$
$s_z = 0 p_x - l_z p_y + l_y p_z$
$w = 0 p_x - p_y + 0 p_z + l_y$

Projection Shadow

$$\begin{bmatrix} s_x \\ s_y \\ s_z \\ 1 \end{bmatrix} = \begin{bmatrix} n\cdot l + d - l_x n_x & -l_x n_y & -l_x n_z & -l_x d \\ -l_y n_x & n\cdot l + d - l_y n_y & -l_y n_z & -l_y d \\ -l_z n_x & -l_z n_y & n\cdot l + d - l_z n_z & -l_z d \\ -n_x & -n_y & -n_z & n\cdot l \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}$$

Point light source (at Point L)



$$\mathbf{s} = l + t(p - l)$$

$$\mathbf{n}\cdot\mathbf{s} + d = 0$$

$$\mathbf{n}\cdot\big(l + t(p-l)\big) + d = 0$$

$$\mathbf{n}\cdot l + t\big(\mathbf{n}\cdot(p-l)\big) = -d$$

$$t(\mathbf{n}\cdot p - \mathbf{n}\cdot l) = -d - \mathbf{n}\cdot l$$

$$t = \frac{-d - \mathbf{n}\cdot l}{\mathbf{n}\cdot p - \mathbf{n}\cdot l} \qquad \therefore\ \mathbf{s} = l + \left[\frac{\mathbf{n}\cdot l + d}{\mathbf{n}\cdot l - \mathbf{n}\cdot p}\right](\mathbf{p} - l)$$

Projection Shadow

$$\mathbf{s} = l + \frac{n\cdot l + d}{n\cdot l - n\cdot p}(p - l)$$

$$s_x = l_x + \frac{n\cdot l + d}{n\cdot l - n\cdot p}(p_x - l_x)$$

$$s_x = l_x + \frac{n\cdot l + d}{n\cdot l - n\cdot p}(p_x - l_x) = \frac{l_x(n\cdot l - n\cdot p) + (n\cdot l + d)p_x - (n\cdot l + d)l_x}{n\cdot l - n\cdot p}$$

$$s_y = l_y + \frac{n\cdot l + d}{n\cdot l - n\cdot p}(p_y - l_y) = \frac{l_x n\cdot l - l_x n_x p_x - l_x n_y p_y - l_x n_z p_z + (n\cdot l + d)p_x - l_x n\cdot l - l_x d}{-n_x p_x - n_y p_y - n_z p_z + n\cdot l}$$

$$s_z = l_z + \frac{n\cdot l + d}{n\cdot l - n\cdot p}(p_z - l_z) = \frac{(n\cdot l + d - l_x n_x)p_x - l_x n_y p_y - l_x n_z p_z - l_x d}{-n_x p_x - n_y p_y - n_z p_z + n\cdot l}$$

$$\begin{bmatrix} s_x \\ s_y \\ s_z \\ 1 \end{bmatrix} = \begin{bmatrix} n\cdot l + d - l_x n_x & -l_x n_y & -l_x n_z & -l_x d \\ -l_y n_x & n\cdot l + d - l_y n_y & -l_y n_z & -l_y d \\ -l_z n_x & -l_z n_y & n\cdot l + d - l_z n_z & -l_z d \\ -n_x & -n_y & -n_z & n\cdot l \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}$$

Projection Shadow Matrix

$$\begin{bmatrix} s_x \\ s_y \\ s_z \\ 1 \end{bmatrix} = \begin{bmatrix} n\cdot l - l_x n_x & -l_x n_y & -l_x n_z & -l_x n_w \\ -l_y n_x & n\cdot l - l_y n_y & -l_y n_z & -l_y n_w \\ -l_z n_x & -l_z n_y & n\cdot l - l_z n_z & -l_z n_w \\ -l_w n_x & -l_w n_y & -l_w n_z & n\cdot l - l_w n_w \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}$$

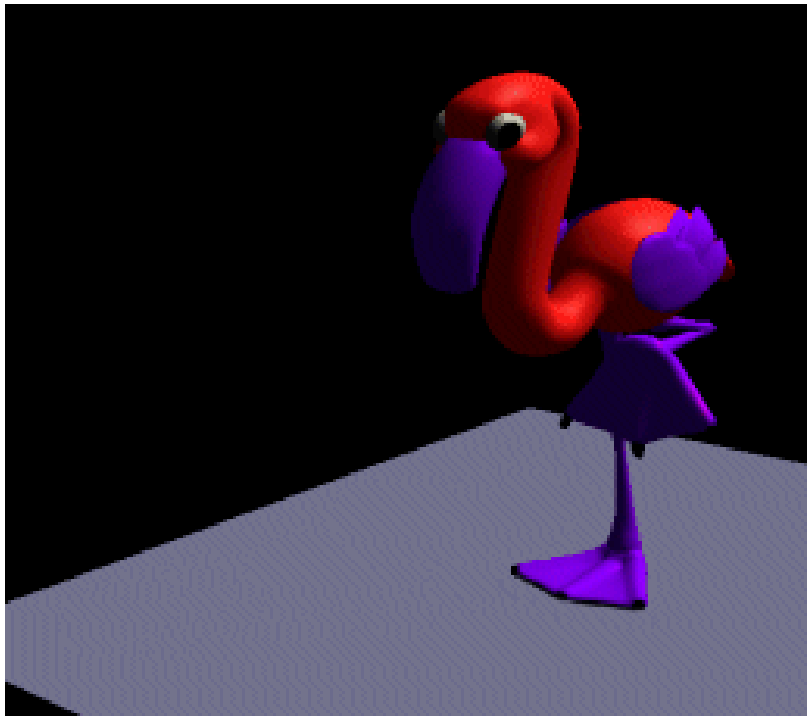
투영평면 $n = [n_x, n_y, n_z, n_w]$

광원 $l = [l_x, l_y, l_z, l_w]$ *where* $l =$ 평행광원이면 $l_w = 0,\ l =$ 점광원이면 $l_w = 1$

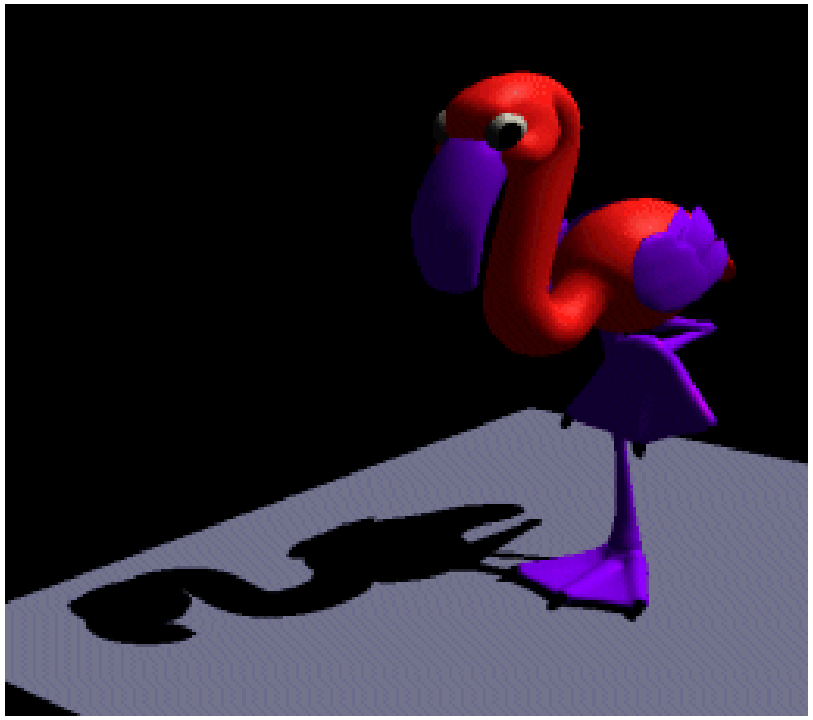
Projection Shadow Matrix

```
// create a shadow matrix that will project the desired shadow
void ShadowMatrix(GLfloat shadowMat[16], GLfloat plane[4], GLfloat lightpos[4])
{
    GLfloat dot; // dot product of light position and ground plane normal
    dot = plane[0] * lightpos[0] + plane[1] * lightpos[1] + plane[2] * lightpos[2]
          + plane[3] * lightpos[3];
    shadowMat[0] = dot - lightpos[0] * plane[0];
    shadowMat[1] = 0.f - lightpos[1] * plane[0];
    shadowMat[2] = 0.f - lightpos[2] * plane[0];
    shadowMat[3] = 0.f - lightpos[3] * plane[0];
    shadowMat[4] = 0.f - lightpos[0] * plane[1];
    shadowMat[5] = dot - lightpos[1] * plane[1];
    shadowMat[6] = 0.f - lightpos[2] * plane[1];
    shadowMat[7] = 0.f - lightpos[3] * plane[1];
    shadowMat[8] = 0.f - lightpos[0] * plane[2];
    shadowMat[9] = 0.f - lightpos[1] * plane[2];
    shadowMat[10] = dot - lightpos[2] * plane[2];
    shadowMat[11] = 0.f - lightpos[3] * plane[2];
    shadowMat[12] = 0.f - lightpos[0] * plane[3];
    shadowMat[13] = 0.f - lightpos[1] * plane[3];
    shadowMat[14] = 0.f - lightpos[2] * plane[3];
    shadowMat[15] = dot - lightpos[3] * plane[3];
}
```

Shadow

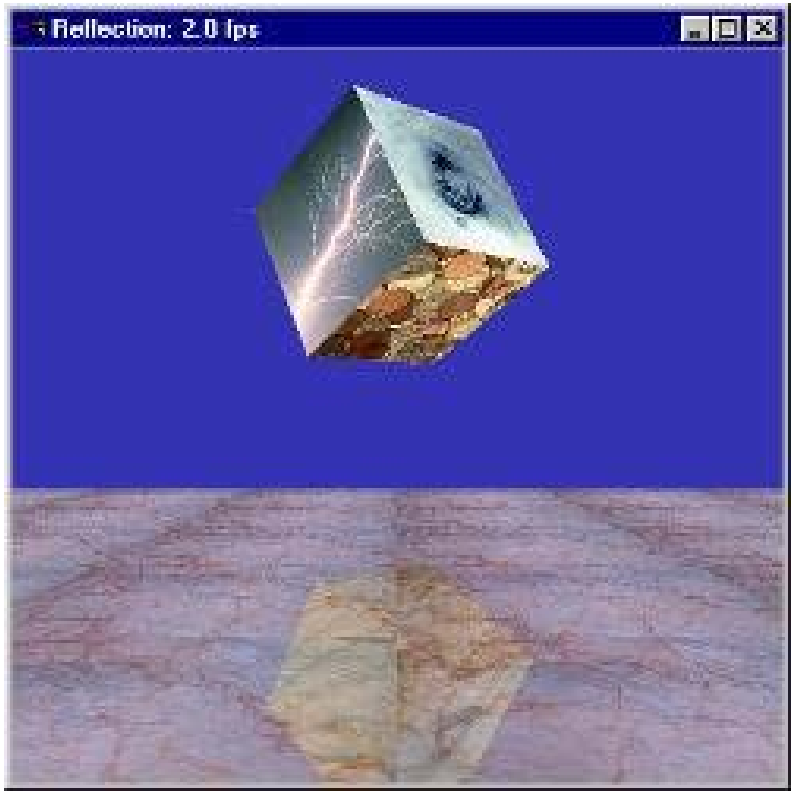


Render without shadow



Render with shadow

Reflection



http://www.gamasutra.com/features/19990723/opengl_texture_objects_02.htm

Planar Reflection

- How to calculate the reflection point, $q'=(x_0', y_0', z_0')$ of the point, $q=(x_0, y_0, z_0)$ for the mirror plane (n, d)

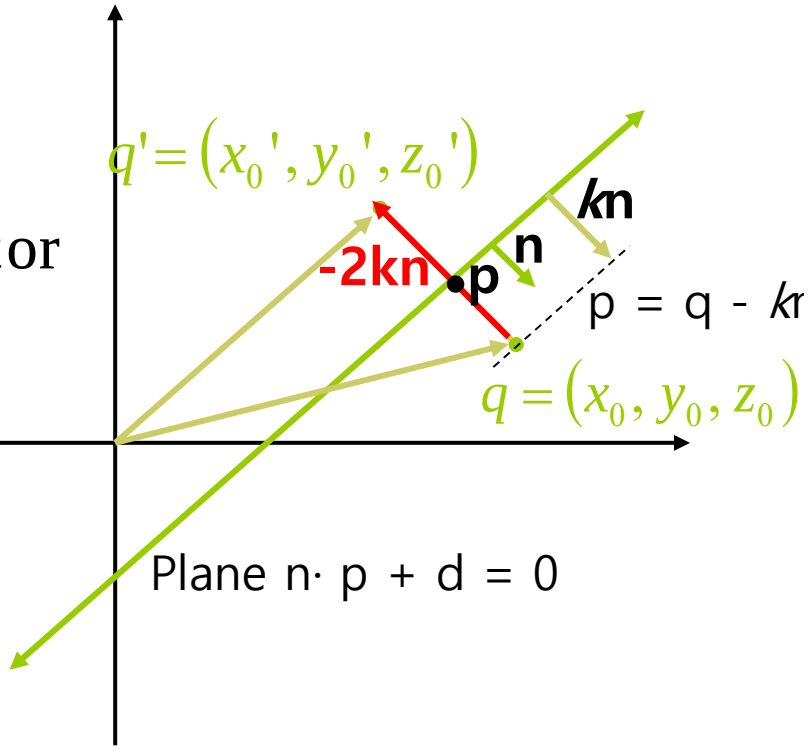
$$q' = q - 2k\mathrm{n}$$

$$= q - 2\frac{ax_0 + by_0 + cz_0 + d}{\sqrt{a^2 + b^2 + c^2}}\mathrm{n}$$

$$= q - 2(n \cdot q + d)\mathrm{n} \text{ when n is unit vector}$$

$$q' = Rq$$

$$\mathbf{R} = \begin{bmatrix} 1-2a^2 & -2ab & -2ac & -2ad \\ -2ab & 1-2b^2 & -2bc & -2bd \\ -2ac & -2bc & 1-2c^2 & -2cd \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



The signed shortest distance from point q to plane is k = n•q + d (if n is a unit vector)

Planar Reflection

$$\begin{bmatrix} x_0' \\ y_0' \\ z_0' \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} - 2(ax_0 + by_0 + cz_0 + d)\begin{bmatrix} a \\ b \\ c \end{bmatrix} \quad when\ n\ is\ unit\ vector\ (a^2 + b^2 + c^2 = 1)$$

$$\begin{aligned} x_0' &= x_0 - 2(ax_0 + by_0 + cz_0 + d)a \\ &= (1 - 2a^2)x_0 - 2aby_0 - 2acz_0 - 2ad \\ y_0' &= y_0 - 2(ax_0 + by_0 + cz_0 + d)b \\ &= 2abx_0 + (1 - 2b^2)y_0 - 2bcz_0 - 2bd \\ z_0' &= z_0 - 2(ax_0 + by_0 + cz_0 + d)c \\ &= -2acx_0 - 2bcy_0 + (1 - 2c^2)z_0 - 2cd \end{aligned}$$

$$\begin{bmatrix} x_0' \\ y_0' \\ z_0' \end{bmatrix} = R\begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}$$

$$\mathbf{R} = \begin{bmatrix} 1-2a^2 & -2ab & -2ac & -2ad \\ -2ab & 1-2b^2 & -2bc & -2bd \\ -2ac & -2bc & 1-2c^2 & -2cd \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Planar Reflection

- Reflection transformation matrix for the plane (*yz-*, *xz-*, *xy-plane*)

yz-plane Plane(1,0,0,0)

$$\mathbf{R}_{yz} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

xz-plane Plane(0,1,0,0)

$$\mathbf{R}_{xz} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

xy-plane Plane(0,0,1,0)

$$\mathbf{R}_{xy} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Planar Reflection Matrix

```
void ReflectionMatrix(GLfloat reflectionMat[16], GLfloat plane[4]) // create a reflection matrix
{
    reflectionMat[0] = 1 - 2 * plane[0] * plane[0];
    reflectionMat[1] =    - 2 * plane[0] * plane[1];
    reflectionMat[2] =    - 2 * plane[0] * plane[2];
    reflectionMat[3] =    - 2 * plane[0] * plane[3];

    reflectionMat[4] =    - 2 * plane[1] * plane[0];
    reflectionMat[5] = 1 - 2 * plane[1] * plane[1];
    reflectionMat[6] =    - 2 * plane[1] * plane[2];
    reflectionMat[7] =    - 2 * plane[1] * plane[3];

    reflectionMat[8] =      - 2 * plane[2] * plane[0];
    reflectionMat[9] =      - 2 * plane[2] * plane[1];
    reflectionMat[10] = 1 - 2 * plane[2] * plane[2];
    reflectionMat[11] =    - 2 * plane[2] * plane[3];

    reflectionMat[12] = 0.0;
    reflectionMat[13] = 0.0;
    reflectionMat[14] = 0.0;
    reflectionMat[15] = 1.0;
}
```