Johns Hopkins
Engineering for Professionals

605.767 Applied Computer Graphics

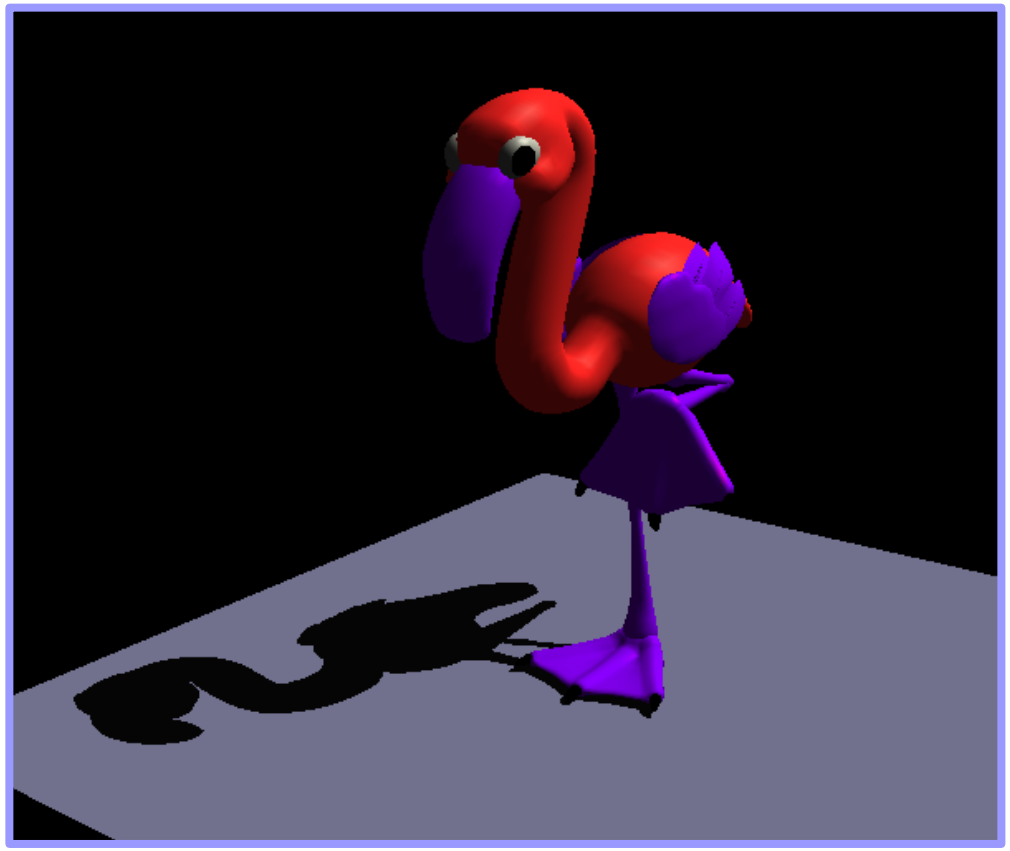
Brian Russin

Module 8C
Planar Shadows

Interactive Shadow Rendering Techniques

- Planar projected shadows
  - Shadows on a ground plane or general plane
  - Works well on flat surfaces
- Shadow volumes
  - Two stage process where volumes are created by considering each light source
    - Efficient determination of the shadow volume can be difficult
  - Volumes are accessed to determine if any object polygon is in the shadow
  - Uses stencil buffer
- Shadow Z-buffer or shadow map
  - Multi-pass, requires special support within OpenGL, DirectX
- Pre-computed shadow textures (lightmaps)
  - Not for dynamic shadows
- Projected textures
- Shadow generation based on transformation and clipping
  - Object-precision algorithm
  - Shadows are pre-computed and stored in a database

Planar Projected Shadows



Shadow is projected onto the plane of the floor
(Light from upper right)
From NVidia developer docs: GDC2K_ShadowsTransparencyFog.ppt

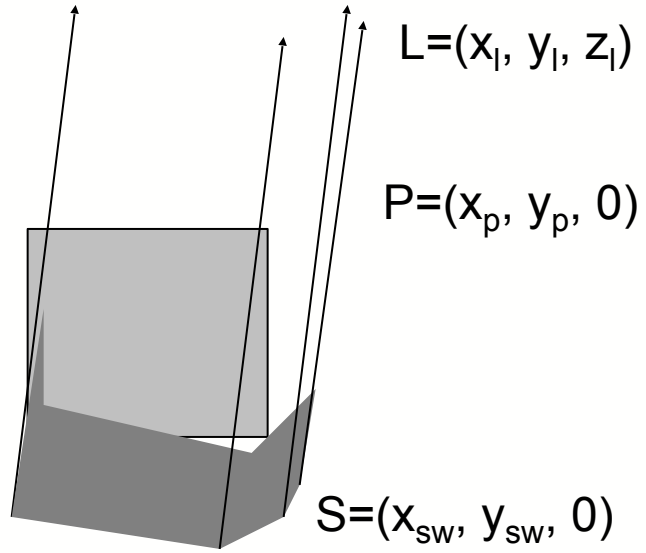
Shadows on a Ground Plane

- Simple shadow generation method developed by Blinn (1988)
  - Draws projection of an object onto a ground plane
    - Rendering the object creates shadow-like polygons in the ground plane
    - Flattens 3D model into ground plane
  - Suitable in simple scenes where objects do not cast shadows on each other
  - Projected 'shadow' polygons are drawn using a dark intensity or dark color blended with current color
  - Supports either positional or directional lights
  - Shadow effect is only applied to planes
- Some issues need to be overcome
  - Z-buffer aliasing
  - Multiple blending
  - Extending off ground region

Shadows on a Ground Plane

- Points on object P cast shadows on the ground plane at point S given a **directional** light position L
  - z = 0 (ground plane)
    - Could alternatively derive this for x or y as ground plane
    - Basically a z scale (of 0) and a shear
- $x_s = x_p - (z_p / z_l)\, x_l$, $y_s = y_p - (z_p / z_l)\, y_l$
- Derived from similar triangles
- Can be written in matrix form:

$$\begin{bmatrix} x_s & y_s & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \frac{-x_l}{z_l} & 0 \\ 0 & 1 & \frac{-y_l}{z_l} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix}$$



Rendering Projection Shadows

- Sequence to render an object that has a shadow cast from a directional light onto the x,y plane:
  - Render the scene (including shadowing object)
  - Set the ModelView matrix to the shadow projection matrix
    - Includes the scaling by 0 to “flatten” the object onto the plane
  - Perform transformations to position and orient the shadowing object
  - Set the shadow color
    - Perhaps use blending
  - Disable depth writing
  - Render the shadowing object
    - The transform flattens the shadowing object onto the ground plane

Rendering Projection Shadows (cont.)

- Shadows from point light sources can be represented by using a perspective transform
  - Not quite the same as the OpenGL perspective transform!
    - Do not want to scale to the [-1,1] box
- Constructing a projection matrix for an arbitrary plane (P) and light position (l)
  - Plane equation of form $P: n \cdot x + d = 0$
  - Ray from l through vertex v and intersecting plane P
    - Intersect point: $p = l - \frac{d + n \cdot l}{n \cdot (v - l)}(v - l)$
    - Projection matrix M that satisfies Mv=p

$$\begin{bmatrix} (n \cdot l) + d - l_x n_x & -l_x n_y & -l_x n_z & -l_x d \\ -l_y n_x & (n \cdot l) + d - l_y n_y & -l_y n_z & -l_y d \\ -l_z n_x & -l_z n_y & (n \cdot l) + d - l_z n_z & -l_z d \\ -n_x & -n_y & -n_z & (n \cdot l) \end{bmatrix}$$

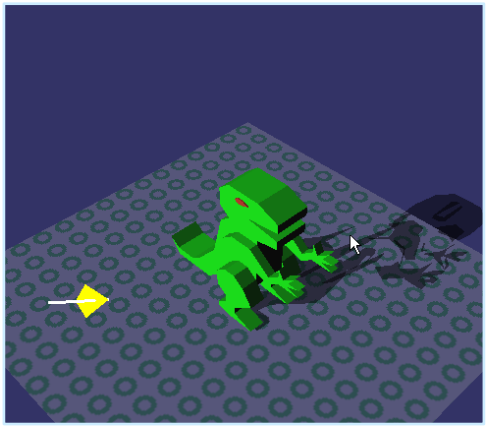
Constructing a Shadow Matrix

Following code allows either positional or directional light

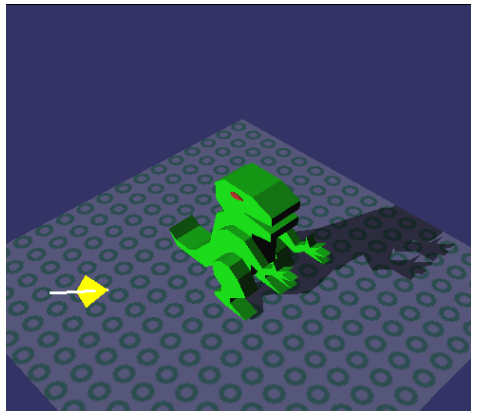
```
void shadowMatrix(GLfloat shadowMat[4][4], GLfloat groundplane[4], GLfloat lightpos[4])
{
   /* Find dot product between light position vector and ground plane normal. */
  GLfloat dot = groundplane[X] * lightpos[X] + groundplane[Y] * lightpos[Y] +
                          groundplane[Z] * lightpos[Z] + groundplane[W] * lightpos[W];
  shadowMat[0][0] = dot - lightpos[X] * groundplane[X];
  shadowMat[1][0] = 0.f - lightpos[X] * groundplane[Y];
  shadowMat[2][0] = 0.f - lightpos[X] * groundplane[Z];
  shadowMat[3][0] = 0.f - lightpos[X] * groundplane[W];
  shadowMat[0][1] = 0.f - lightpos[Y] * groundplane[X];
  shadowMat[1][1] = dot - lightpos[Y] * groundplane[Y];
  shadowMat[2][1] = 0.f - lightpos[Y] * groundplane[Z];
  shadowMat[3][1] = 0.f - lightpos[Y] * groundplane[W];
  shadowMat[0][2] = 0.f - lightpos[Z] * groundplane[X];
  shadowMat[1][2] = 0.f - lightpos[Z] * groundplane[Y];
  shadowMat[2][2] = dot - lightpos[Z] * groundplane[Z];
  shadowMat[3][2] = 0.f - lightpos[Z] * groundplane[W];
  shadowMat[0][3] = 0.f - lightpos[W] * groundplane[X];
  shadowMat[1][3] = 0.f - lightpos[W] * groundplane[Y];
  shadowMat[2][3] = 0.f - lightpos[W] * groundplane[Z];
  shadowMat[3][3] = dot - lightpos[W] * groundplane[W];
}
```

Issues with Projected Planar Shadows

- Shadow must be cast on infinite planes
  - Stencil can limit shadow to finite planar region
- “Pile of polygons” creates depth buffer artifacts
  - Can use a polygon offset to fix this
    - Or adjust the shadow projection matrix
  - Stenciling also provides a solution
- Difficult to blend shadow with ground texture
  - Stencil testing can eliminate multiple blending
  - Specular highlights can show in shadow



Shadow extends off ground region
Depth buffer artifacts
Multiple blending



Fixed!

Polygon Offset

`glPolygonOffset(GLfloat factor, GLfloat units)`

When enabled, the depth value of each fragment is added to the offset value. Offset is added before the depth test and before the depth value is written into the depth buffer. The offset value is:

Offset = m * factor + r * units.

Where m is the maximum depth slope of the polygon and r is an implementation specific constant.

- Polygon offset is useful to many shadow algorithms
  - Projected, planar shadows
    - Help reduce depth buffer aliasing when projected shadow is coplanar with plane it is projected onto
  - Shadow maps
    - Help reduce self-shadowing

Use Stencil Buffer to Prevent Multiple Blending

Drawing multiple, blended shadow polygons creates uneven shadow color. Subsequent polygons blend with other shadow polygons to create darker regions

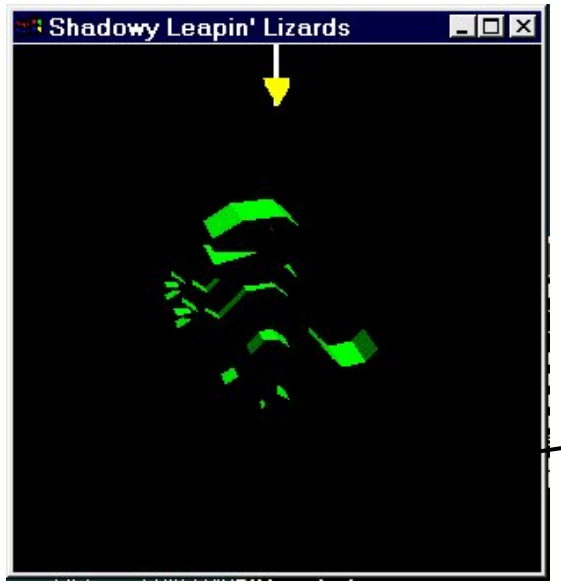


Image from: GDC Tutorial:
Advanced OpenGL Game Development,
“Reflections, Shadows,Transparency, and Fog”
by Mark Kilgard (NVidia Corp)
www.nvidia.com

Notice darks spots on the planar shadow.

Stencil buffer provides a solution:

Clear stencil to zero.

Draw floor with stencil of one.

Draw shadow if stencil is one.

If shadow’s stencil test passes, set stencil to two.

Shadow Projection onto General Plane

- General technique is to project onto general plane(s)
  - Could project each polygon in the scene onto each other polygon plane
    - Clip shadow against the polygon edges
    - Skip polygons that are back-facing relative to light source
  - **Costly!!!**

```
for each polygon (source polygon)
{
  for each polygon (destination polygon != source polygon)
  {
    if (destination polygon is not closer than the source polygon)
    {
      Project source polygon onto destination polygon plane from
          the light's viewpoint
      Create the shadow polygon by clipping the projected
          polygon against the destination polygon edges
    }
  }
}
```