Lighting and Rasterization – Creating Shadows

Intended Learning Outcomes

- Apply fast techniques to generate realistic shadow on the ground plane and its programming implementation
- Extend ray casting technique for general shadow creation
- Apply shadow dow creation

Creating Shadow on Plane

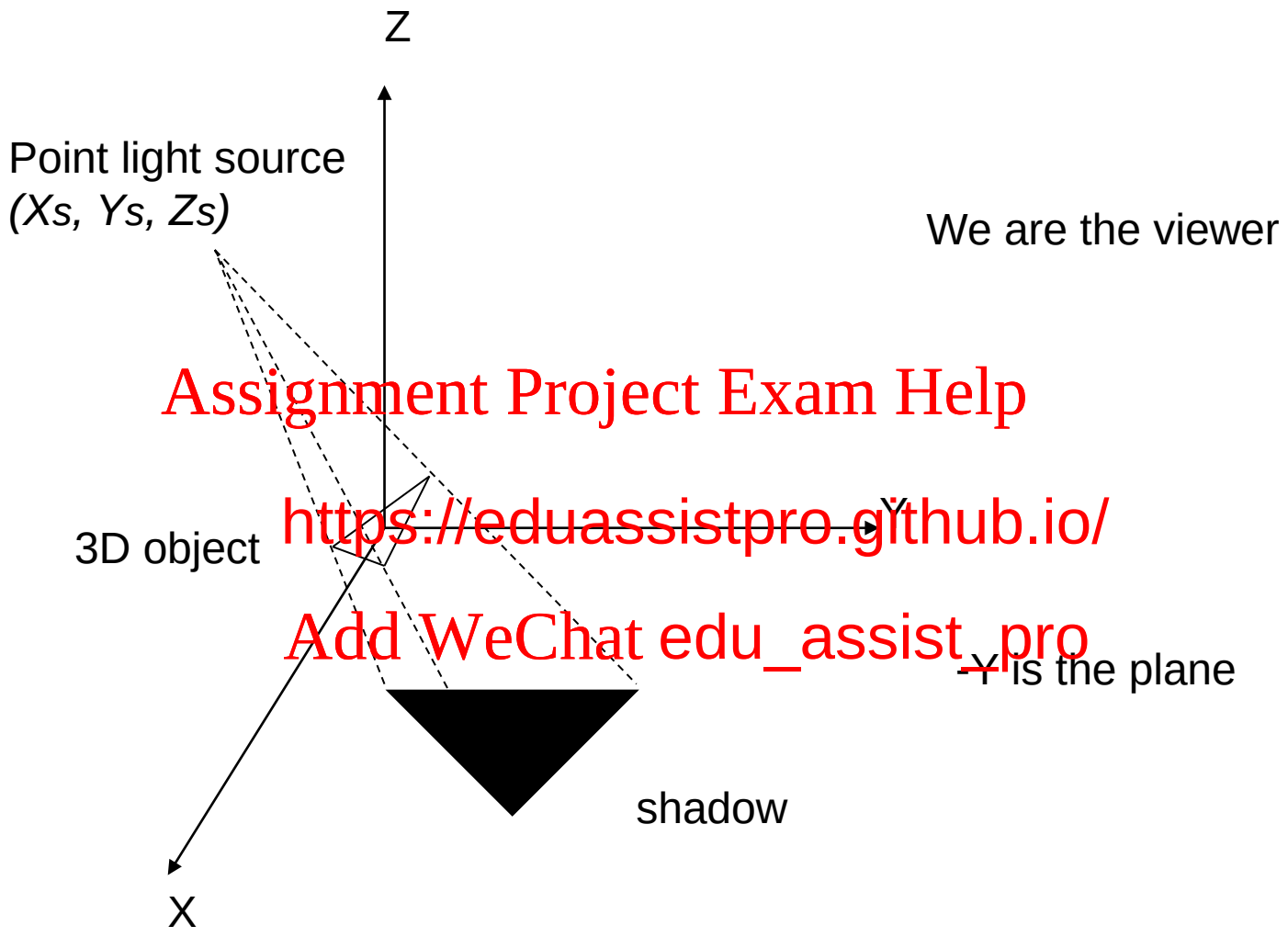
- Works only when projecting objects onto a plane and point light source

- Idea : Given a point light source s and a plane P,

1. Render the obj

2. Use a coordin that transforms s to the PRP and P to the image pla

3. Set the object colour to the shadow colour
4. Perspective project the objects onto the image plane, creating shadows.
5. Use the inverse coordinate system transformation to transform the shadows to the normal coordinate system



Coordinate transformation such that the light source becomes the origin

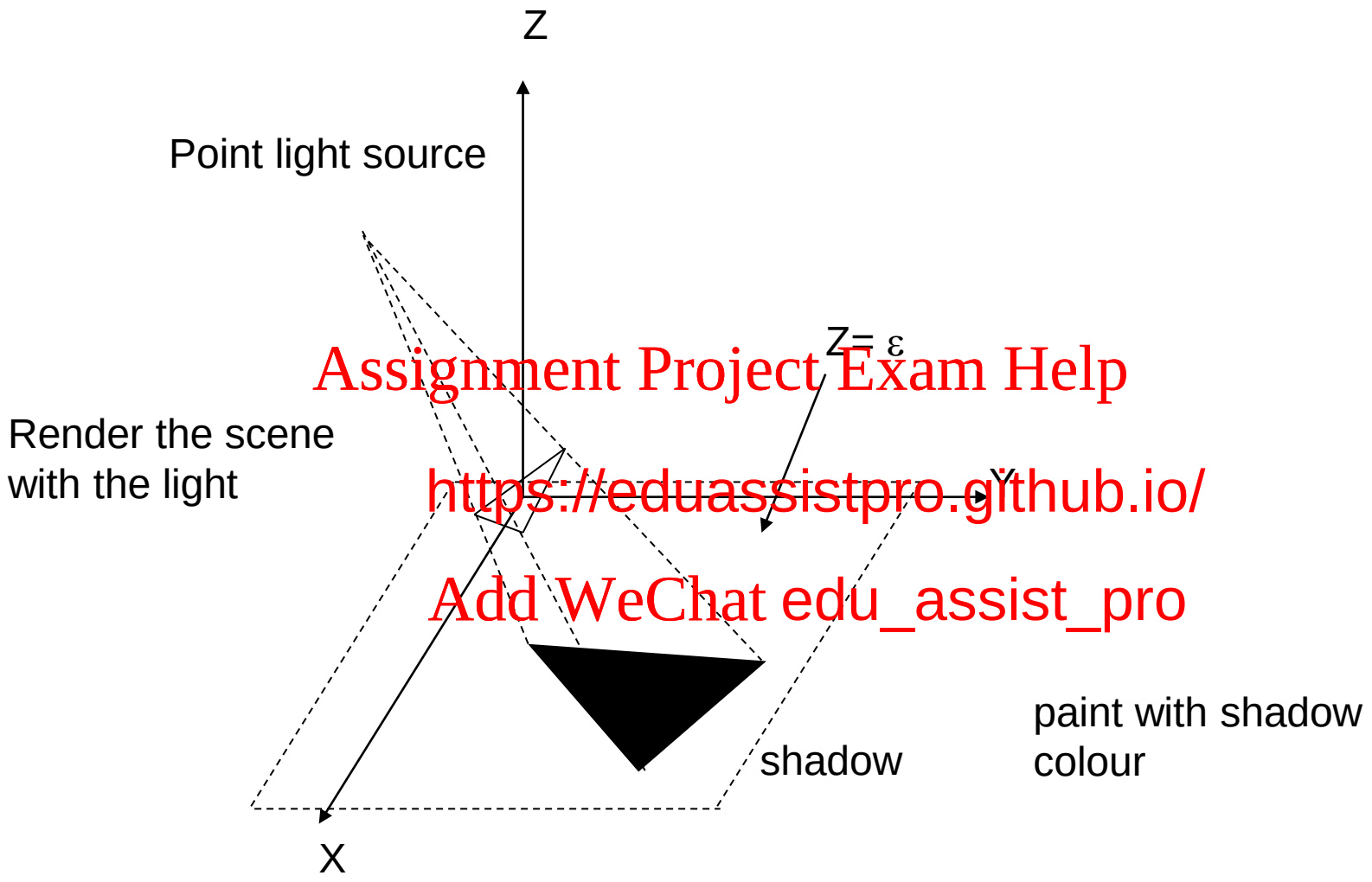
$\mathbf{M}_{s \leftarrow WC} = \mathbf{T}(-X_s, -Y_s, -Z_s)$

Perspective proj

$$\mathbf{M} = \begin{pmatrix} 1 & 0 & 0 & \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{-Z_s} & 0 \end{pmatrix}$$

OpenGL code

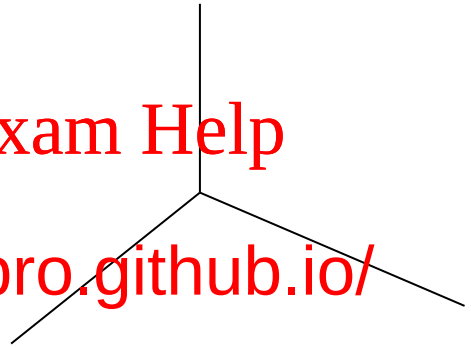
```
GLfloat  light1PosType [ ] = {Xs, Ys, Zs, 1.0};
                    :
GLfloat  M[16];                 // OpenGL is in column major format
                                //  though C is in row major format
for (i=0; i<16; i++)
    M[i]=0;
M[0]=M[5]=M[10]=1;
M[11]=-1.0/Zs;

object ( );

glPushMatrix ( );               // save state

glMatrixMode (GL_MODELVIEW);
glTranslatef (Xs, Ys, Zs);      //  Mwc←s
glMultMatrixf (M);              //   perspective project
glTranslatef (-Xs, -Ys, -Zs);   //  Ms←wc

glColor3fv (shadowcolour);      //  set k_a = k_d = k_s = 0 if you are using lighting model
object ( );

glPopMatrix ( );                //  restore state
```



Implementation notes: in actual programming, cast the shadow on a plane $Z = \varepsilon$ after changing to the light source coordinate system, where ε is a very small number (why?)

Extension to corners

- It can be used to cast shadows on corners of the room (treat it as three planes)



- However, it cannot be used to cast shadows on general non-plane objects

General Shadow creation

Limitations:

- Up to now, shadows can only be casted on planes or corners
- No shade diffe hadows
- The shadow b

- Below we introduce two techniques: ray casting using shadow ray and shadow mapping, that overcome the first two limitations.
- One way to create soft shadows is radiosity, which is a sophisticated model of ambient reflection

Ray Casting

- retrace the light paths of the rays that arrive at the pixel
- for each pixel, send a ray from PRP that goes through the pixel
- find all intersec surfaces
- the nearest intersections is th rt of the surface for that pixel

Ray casting

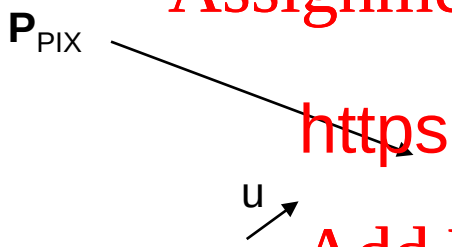
- Consider the math.

$$\boldsymbol{P} = \boldsymbol{P}_0 + s\,\boldsymbol{u} \qquad \text{(Pixel Ray Equation)}$$

$\boldsymbol{P}_0$ may eit

$\boldsymbol{P}_{pix}$ is the (X, Y, Z) coordi e pix

$$\mathbf{u} = \frac{\mathbf{P}_{PIX} - \mathbf{P}_{PRP}}{\left|\mathbf{P}_{PIX} - \mathbf{P}_{PRP}\right|}$$ is a unit vector pointing out from **PRP**



PRP

Ray – Surface Intersections

- Suppose the CG scene consists of n surfaces or polygons
- Compute the intersection point(s) of the pixel ray with each of the n s
- The surface/po on point has the smallest s is the visible surfa
- since it is the nearest

Ray – Sphere Intersection

- Sphere is the simplest surface with analytical equation

$$(X - X_C)^2 + (Y - Y_C)^2 + (Z - Z_C)^2 = r^2$$

$\mathbf{P}_C(X_C, Y_C, Z_C)$ Center

r Radi

$$\left|\mathbf{P} - \mathbf{P}_c\right|^2 - r^2 = 0$$ Vector Equation

Ray-Sphere Intersections (2)

- Sub. $\mathbf{P} = \mathbf{P}_0 + s\mathbf{u}$ gives a quadratic equation
- Solution:

$$s = \mathbf{u} \cdot \Delta\mathbf{P} \pm \sqrt{r^2 \quad \left| \qquad \right|^2} \qquad \Delta\mathbf{P} = \mathbf{P}_C - \mathbf{P}_0$$

- If discriminant < 0, does not i
- Otherwise choose the intersection with the smaller s

- Solution is more difficult and time consuming for more complicated surfaces

Shadowing

- The ray casting method above can be used to determine the visible surface
- For each visible surface point **P**, the question is how to determine whether **P** is in shadow
- Given a set of l in shadow to a subset of light hy the rest
- If in shadow wi n the light intensity due to $\mathbf{S_0}$ is set to ze

Shadow ray

- To test whether **P** is in shadow w.r.t. a point light source $\mathbf{S_o}$:
- Send a pixel ray from **P** to $\mathbf{S_o}$
- If the pixel ray i /polygon on its way, **P** is in sh
- The pixel ray i

Shadow Mapping

- Idea: A point is in shadow iff it is not visible to the light source (a visibility determination problem)
- Change the coordinate system such that the light position is the PRP. We call this the lighting coordinate system
- Perform a persp e depth buffer. The depth buffer hol rest distance to the light source
- For each 3D point to be rendere the lighting coordinate system.
- Project the point. Compare its depth to the value in the depth buffer.
- The point is in shadow if it is not the same value as that in the depth buffer.

Figure from
http://http.developer.nvidia.com/CgTutorial/cg_tutorial_chapter09.html

Figure from
http://www.codinglabs.net/tutorial_opengl_deferred_rendering_shadow_mapping.aspx

References

- Text: Ch. 16-10 for ray casting method
- Creating shadow on plane
  - E. Angel, Interactive Computer Graphics, A Top-Down Approach Using OpenGL, 3rd Ed., 2003, pp. 261-264.