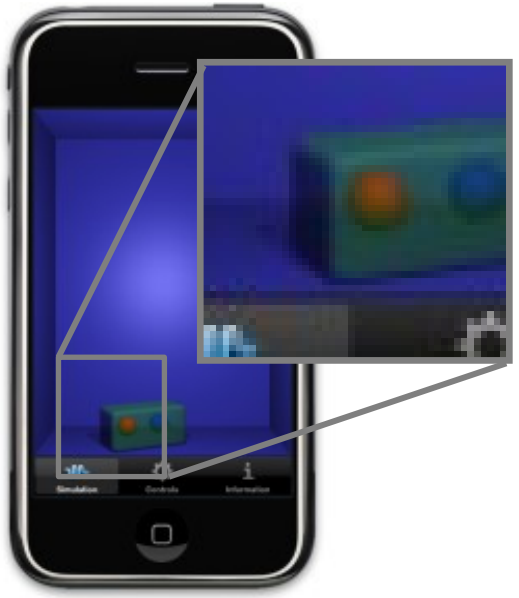
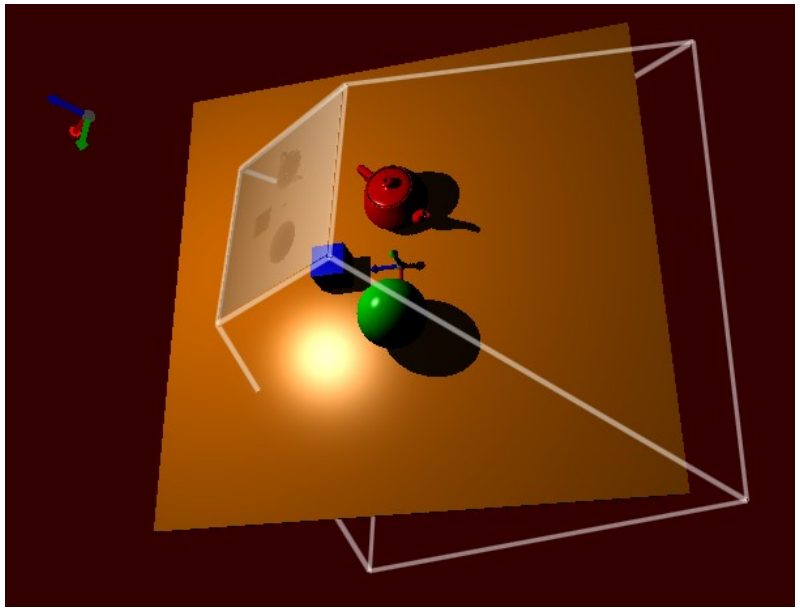
Shadows and Object Order Rendering

Projective Textures, Shadow Maps, and Cheap Shadows



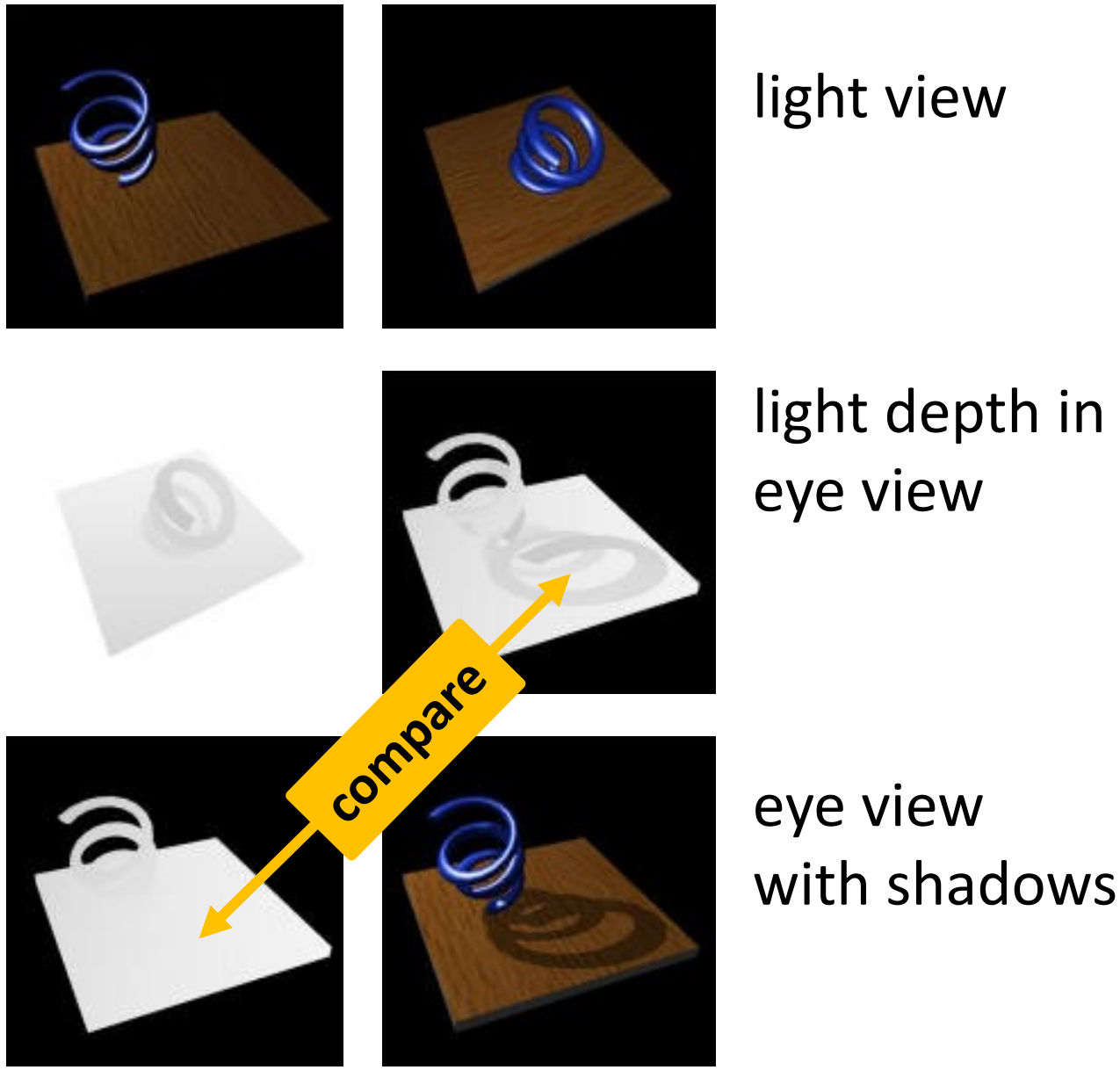


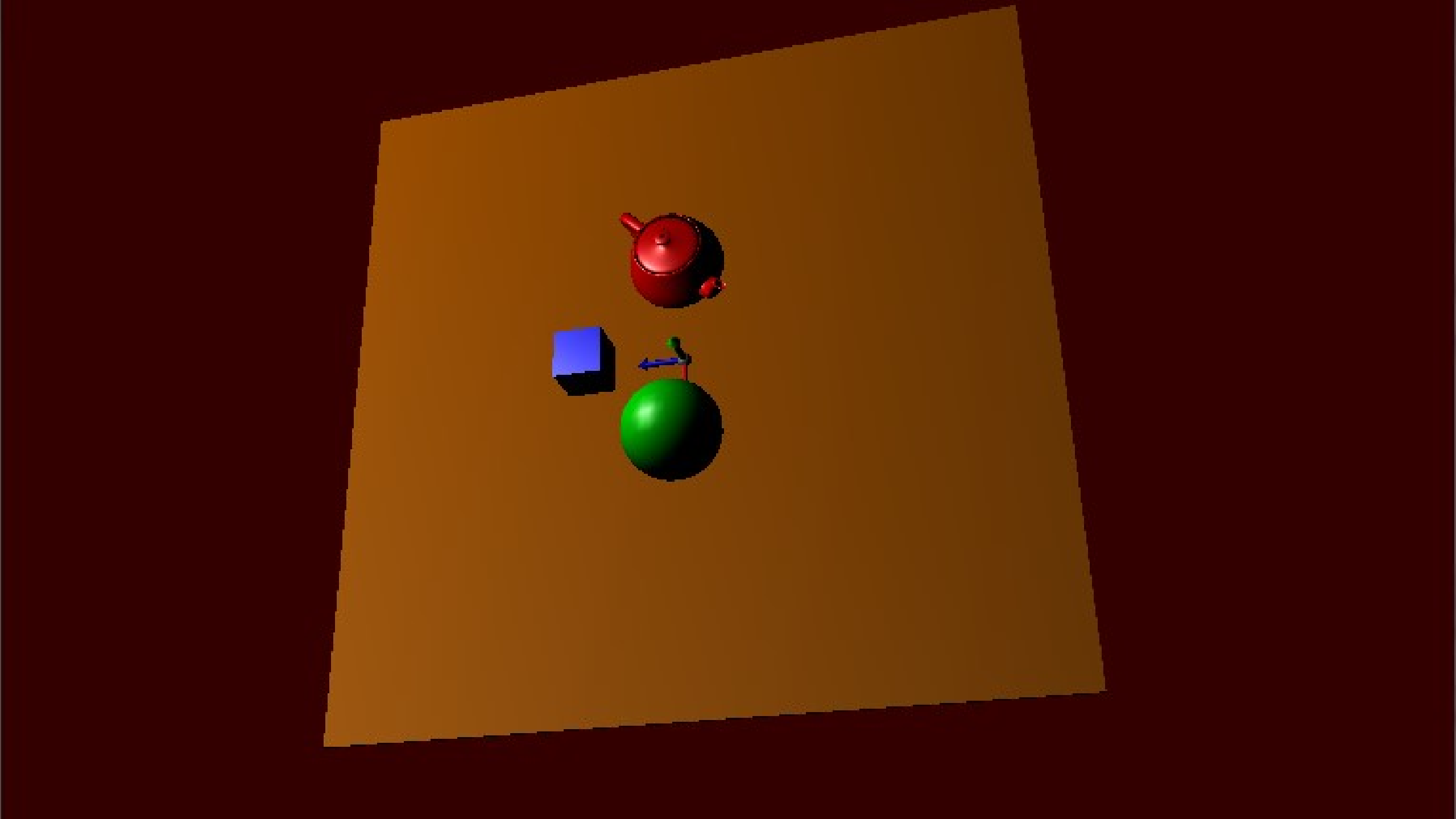
- **Cheap shadows** by projecting geometry onto planar surfaces
- **Shadow volume** is technique to get pixel accurate sharp shadows (not covered in tis course)
- **Shadow maps**, a texture of depth drawn from the light view and then compare light distance when drawing from camera

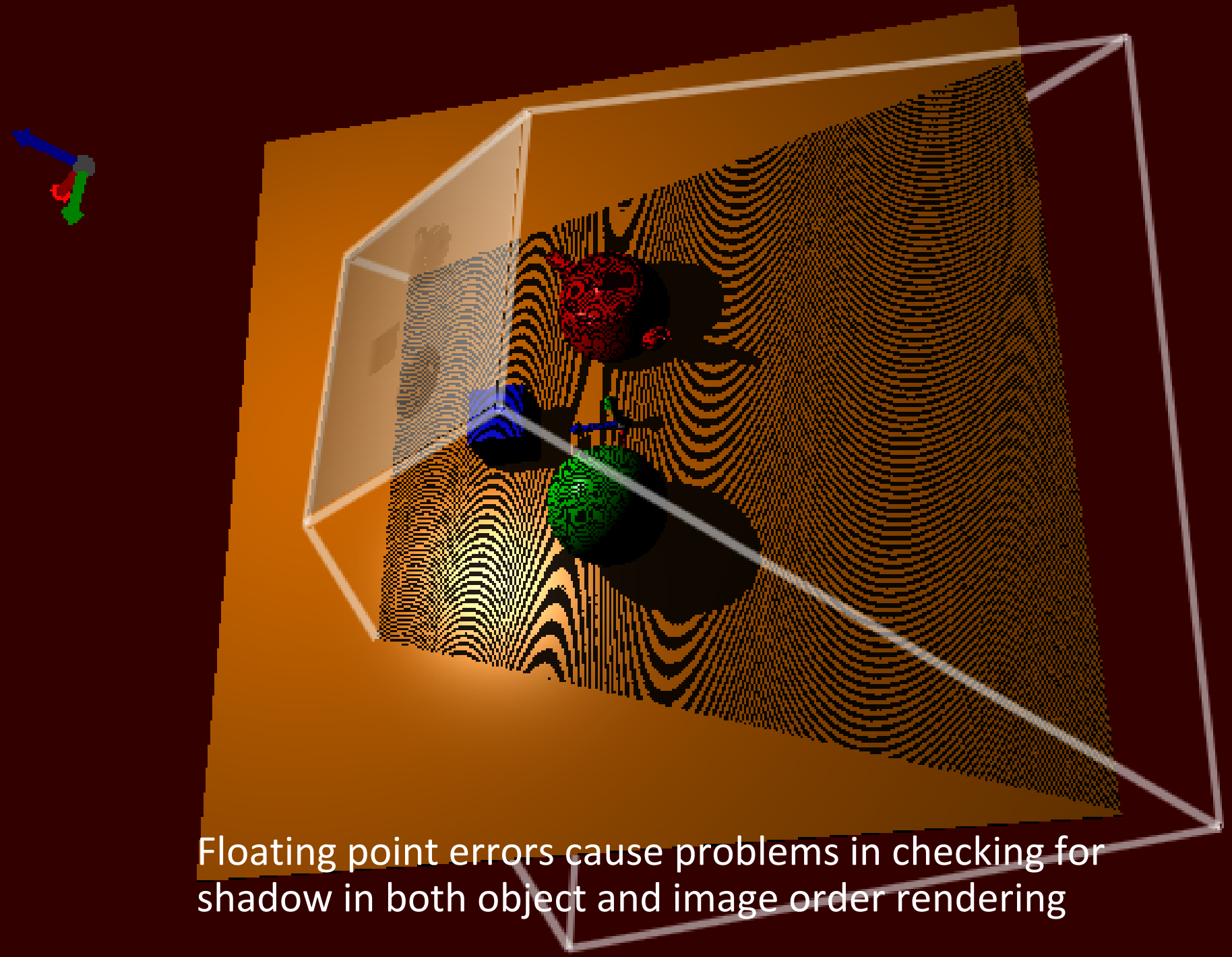
Shadows by Shadow Maps

- Surface is only illuminated if nothing blocks its view of the light
- Need to check if anything is occluding
- Object order rendering
  - ***Can draw the view from the light!***
  - Store depth information for the closest surfaces.
  - Compare this depth (distance) information when processing fragments visible in the camera view

Shadow Maps

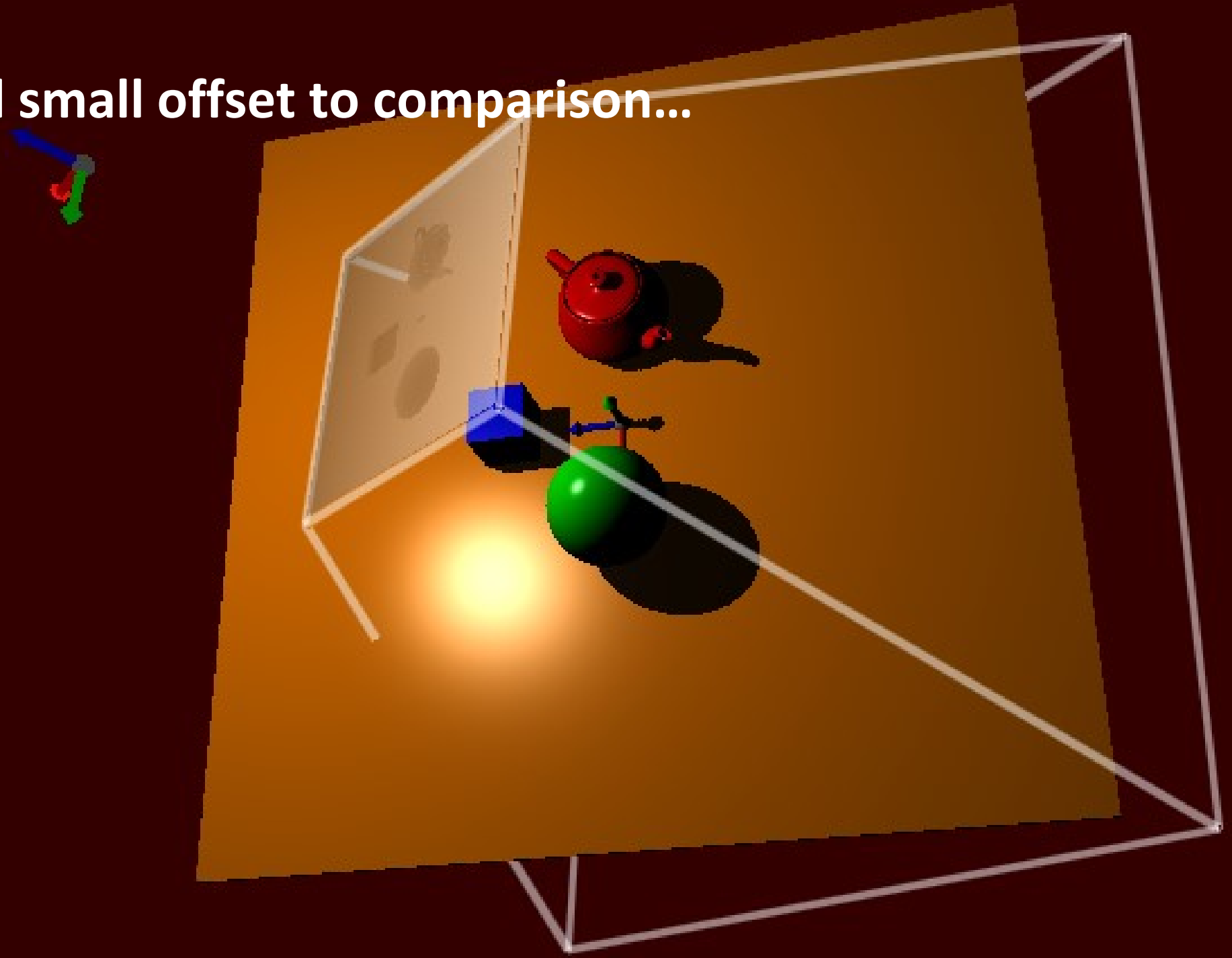

eye view
light view
light depth
light depth in
eye view
compare
eye view depth
eye view
with shadows





Floating point errors cause problems in checking for shadow in both object and image order rendering

Add small offset to comparison…



GLSL Lighting Computations

- Vertex program computes eye coordinate position of the vertex needed for lighting.
  - This is needed for point based lights, to compute the light direction
  - This is also needed for Phong and Blinn-Phong for the viewing direction (when not assuming an orthographics projection)
- In per fragment lighting, the ***vertex program*** computes the surface fragment position in camera coordinates and passes a ***out*** vec3, to be used in the ***fragment program***.

```
out vec3 v; // surface fragment location in camera
…
v = vec3( ViewMatrix * ModelMatrix * gl_Vertex );
```

- Quantities are output at the vertex stage, and interpolated quantities are available at the fragment stage
  - E.g., we also interpolate normals for Blinn-Phong

GLSL Lighting Computations

```
in vec3 v; // surface fragment location in camera coordinates
in vec3 n; // normal of fragment in camera coordinates
```

Given the fragment location in the camera, diffuse lighting can be computed with the following lines in a ***fragment program***

```
vec3 L = normalize( lightPosition.xyz - v );
vec4 Ld = material_kd * lightIntensity * max(dot(n,L), 0.0);
Ld = clamp(Ld, 0.0, 1.0);
```

Shadow Map lookup

```
uniform sampler2D shadowMap;
```

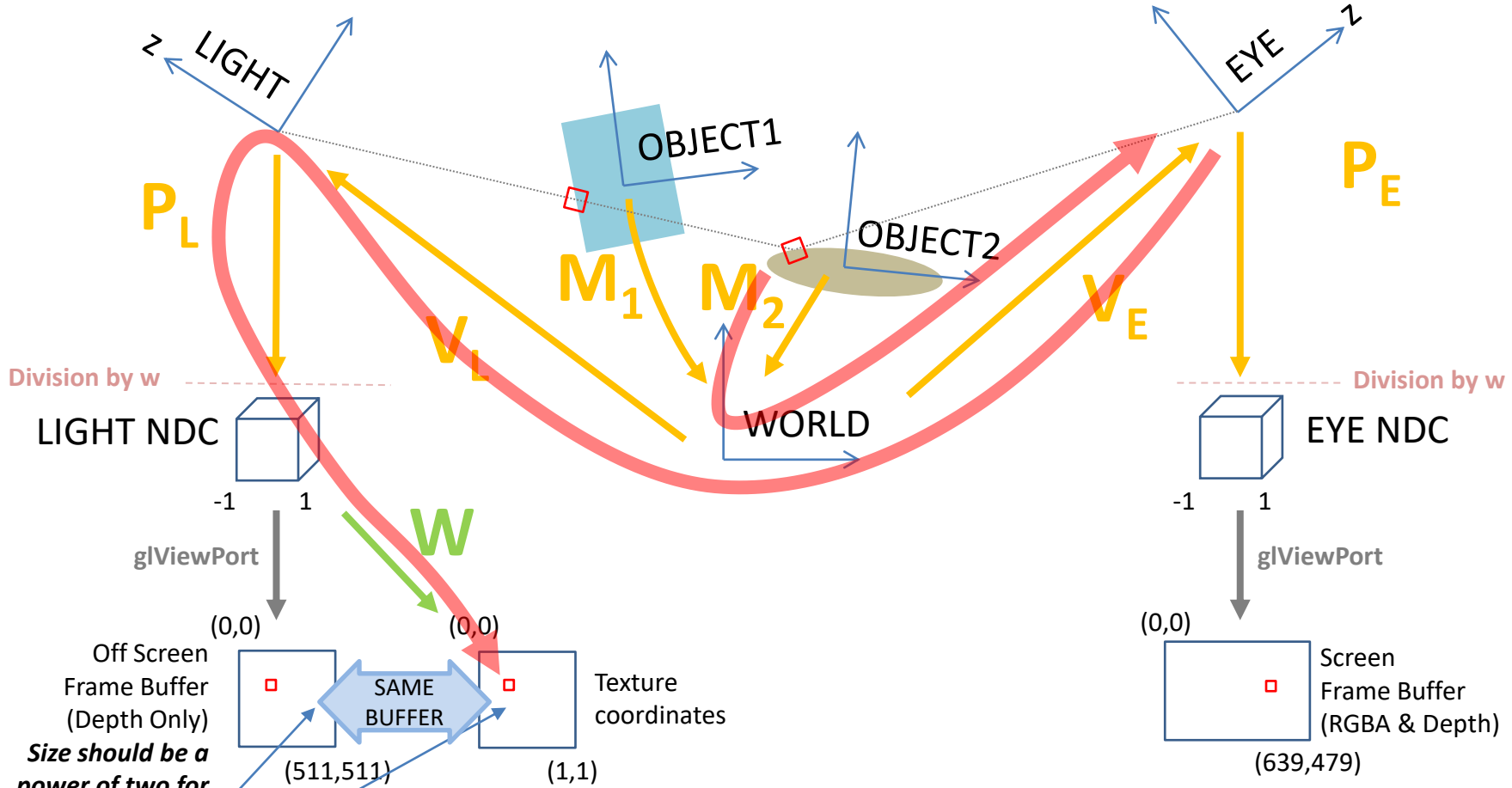
- **shadowMap** is the depth texture rendered from the light view
- Need to compute **pos** as the fragment position…
  - transformed to the light view coordinates,
  - projected into the light's normalized device coordinates NDC
  - Include divide through by w, *and* mapped to [0,1]x[0,1] through a windowing transformation as this is the range used for texture lookups, ***not the [-1,1]x[-1,1] of the NDC***.

```
float distanceFromLight = texture2D( shadowMap, pos.xy ).r
```

- Then compare **distanceFromLight** of the closest surface to the fragment you are currently shading (which is at distance **pos.z**). Note that you need to include some offset to avoid self shadowing!

Shadow Map lookup

- Matrix transformations to convert vertex locations to light NDC can be stored in a ***uniform*** matrix
  - Uniform is a keyword to denote values that are changed infrequently and set as parameters to the vertex and or fragment programs before drawing many primitives.
- The interpolated (varying) fragment location in NDC can then be used in the fragment program
  - Division by w is necessary before use.



Size should be a power of two for ease of use as an OpenGL texture

Same memory!

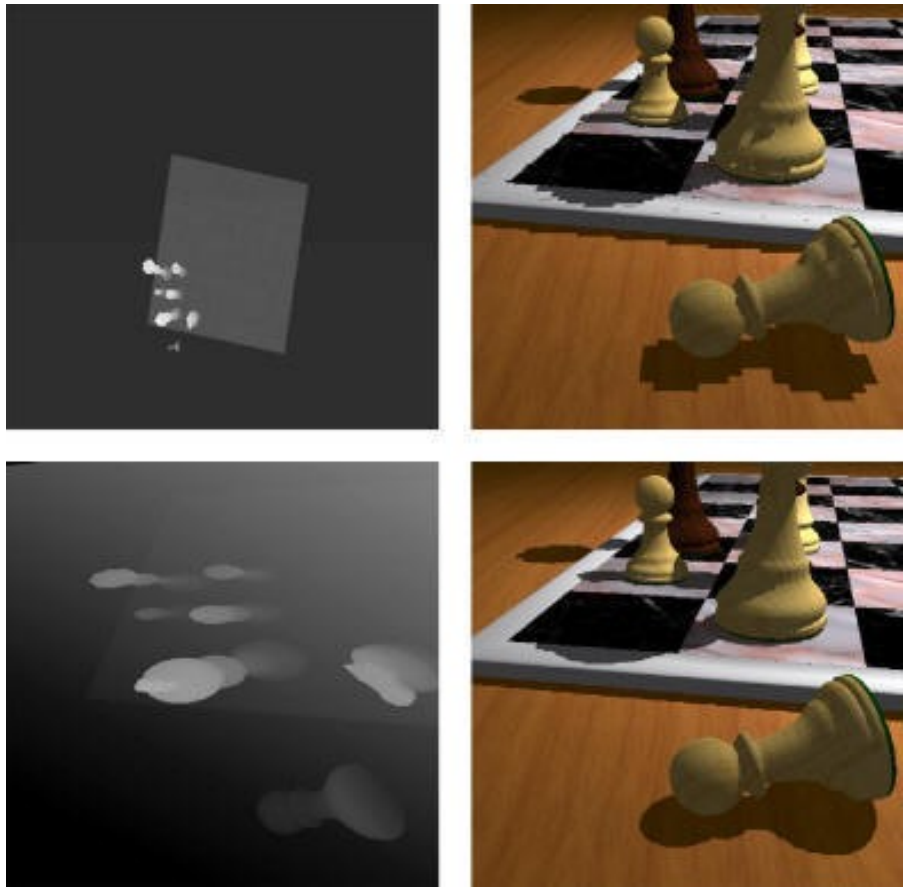
Just seen differently depending on if we are rendering to the buffer or accessing it as a texture.

Suppose you are drawing the eye view, and you have already used the trackball to prepare the eye view and eye projection on the modelview and projection matrix stacks. HOW DO YOU…

- **DRAW THE LIGHT COORDINATE FRAME?**
- **DRAW THE LIGHT FRUSTUM?**
- **CHECK THE SHADOWMAP FOR THE CURRENT FRAGEMENT?**

Shadow Maps – extras (out of scope of course)

- Need to choose appropriate near / far / field of view for light
- Can use perspective shadow maps to improve accuracy
  - See GEMS or SIGGGRAPH Paper
- Can use percentage closer filtering for soft shadows using perturbed lookups

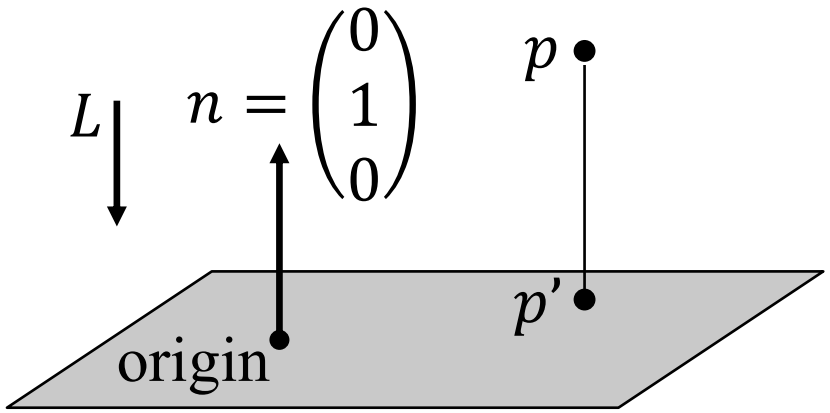


Cheap Shadow Projection

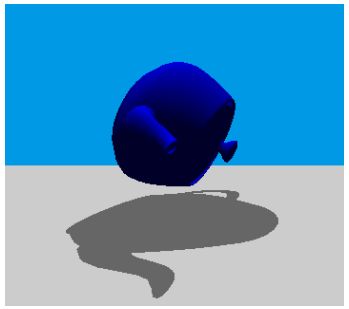
- Redraw geometry projected onto floor plane using a dark solid colour (i.e., *draw the shadows*)
  - Only good for planar shadows in simple environments.
  - No self shadowing, but often very compelling.

Cheapest Shadow Projection

- Assume directional light from above, (0,-1,0)
- Assume plane with y-axis normal containing the origin
- Projection simply sets the y value of points to zero



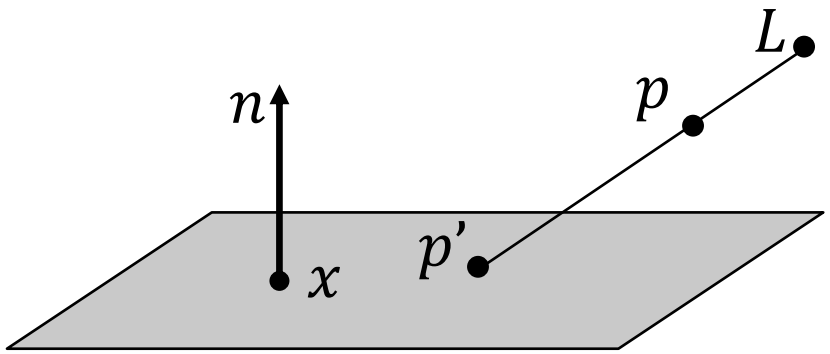
$$\begin{pmatrix} p'_x \\ p'_y \\ p'_z \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}$$

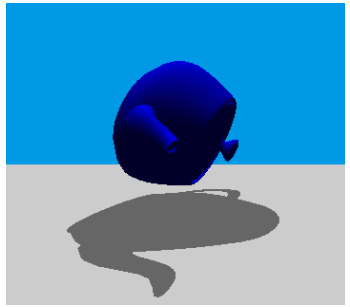


Note: this picture is showing a point light projection as opposed to the simple orthographic projection we describe here. How would you set up the projection for a point light and a general plane?

Cheap Shadow Projection

- Derive a homogeneous transformation matrix that projects point P onto ground plane point P'
  - Plane defined with normal, n, and point on plane, x





From the plane equation, let

$D = -n^T x$ i.e., $n^T p' + D = 0$.

Write the parameterized ray from the light through p as

$$p' = L + t(p - L)$$

and solve for t such that p' is on the plane,

$$n^T(L + t(p - L)) + D = 0.$$

Solving for t gives $t = \frac{-(D+n^T L)}{n^T(p-L)}$.

Thus,

$$p' = L + \frac{-(D + n^T L)(p - L)}{n^T(p - L)}$$

We can tease this apart into a linear part, translation, and perspective divide…

Shadow Volumes

- Pixel-perfect shadows in screen space
- Use a special buffer, the ***stencil buffer***, and draw polygons associated with all silhouette edges
- Count number of times you step into or out of shadow

[Kuttuva]
[Everitt, Kilgard]