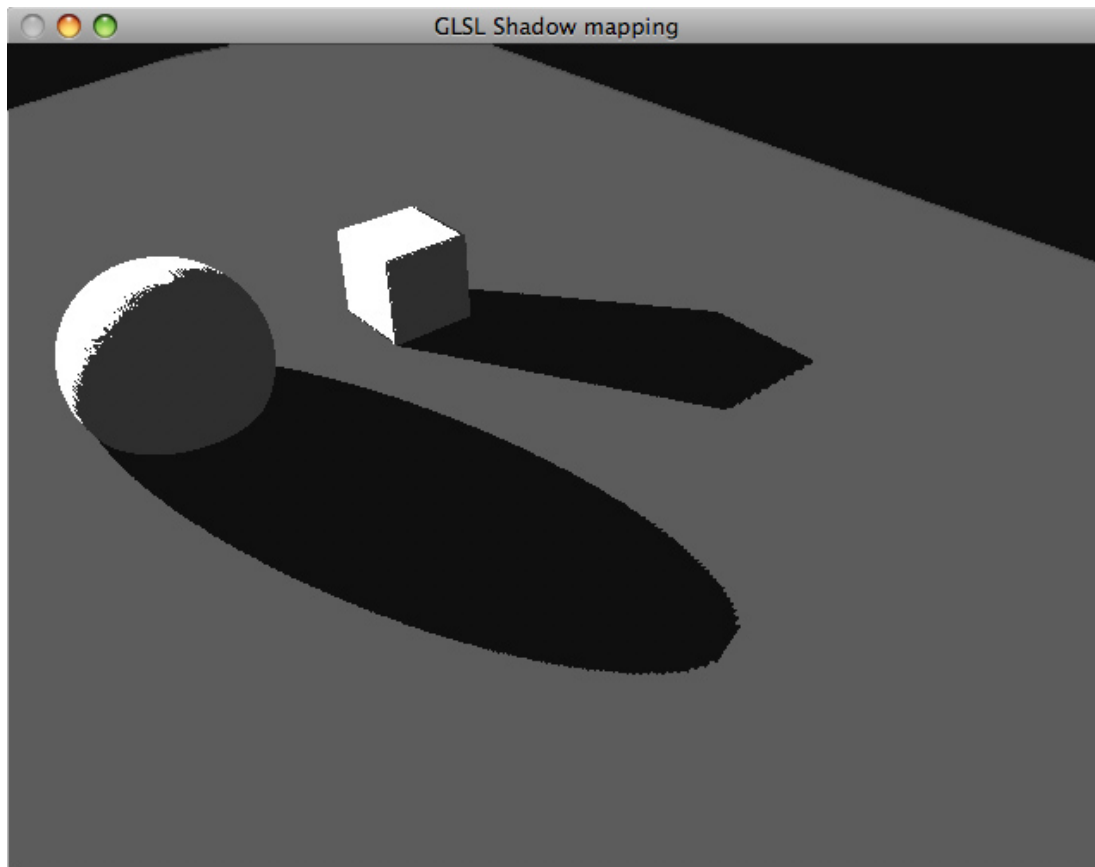


CS174A : Introduction to Computer Graphics

Kinsey 1240
MW 4-6pm

Scott Friedman, Ph.D
UCLA Institute for Digital Research and Education

Shadows



Shadows

- Shadows are integral to light as experienced in the real world.
 - They help us understand the shape and spatial relationship between objects.
 - Particularly challenging in real-time graphics.
 - Conceptually simple algorithms are too slow to implement in real-time.
 - No single technique, even today, works effectively in all situations.
 - As we have seen before, the best approach really depends on the particular constraints of the situation.

Shadows

- What is a shadow?
 - Dictionary
 - *An area that is not or only partially illuminated because of the interception of light by an opaque object between the area and the light.*
 - More specifically, from a graphics perspective
 - *A shadow is the region of space for which at least one point of a light source is occluded.*
 - Assumptions going forward
 - Only direct illumination is considered, bouncing light ignored
 - Occluders are assumed to be opaque

Shadows

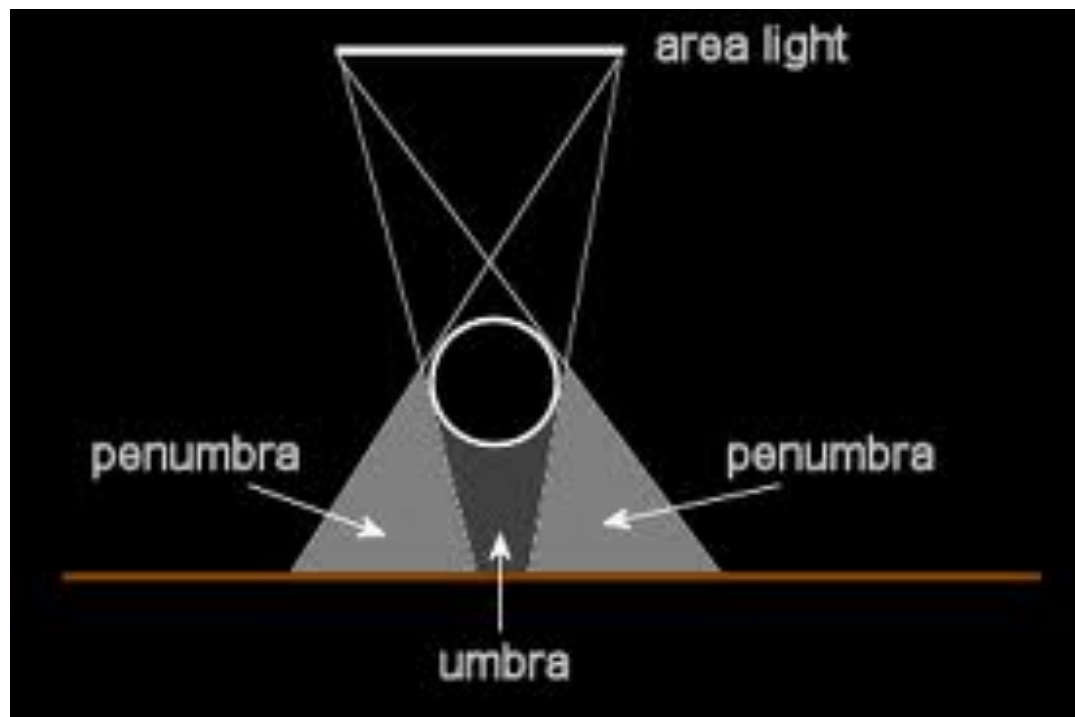
- What is a shadow?
 - Completely general algorithms that extend beyond opaque objects and direct illumination are beyond current hardware capabilities *for real-time rendering*.
 - Accurate shadows have been on every list of most important unsolved problems in graphics.
 - Given that, we will stick to a simplified definition.

Shadows

- What is *in* shadow?
 - A point p on a surface can be one of the following with respect to an *area* light source (more than a point)
 - Entire light source is blocked by the scene
 - » p is within the umbra and in shadow
 - Light source partially blocked by the scene
 - » p is within the penumbra and partially in shadow
 - Light source not blocked by the scene
 - » p is lit and not in shadow

Shadows

- What is *in* shadow?



Shadows

- What is *in* shadow?
 - Basically, we trace a ray from the light source l to a point p on a surface.
 - If there is an intersection *between* l and p , the point p is in shadow.
 - The object the ray intersects with is called the *occluder*.
 - » Also, sometimes, called the *blocker* or *shadow caster*.
 - The object the point p belongs to is called the *receiver*.

Shadows

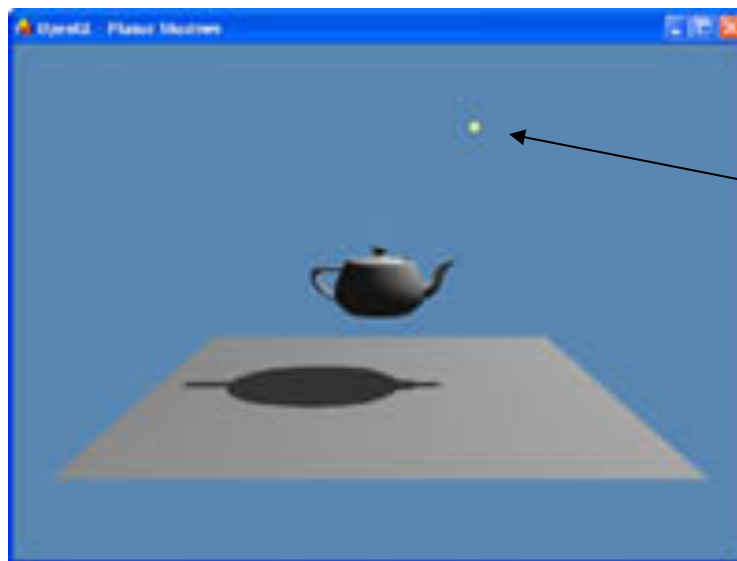
- Simplify, even more.
 - To avoid the difficult integration needed for area light sources we will restrict ourselves to point lights.
 - Considering point light sources restricts the result to hard shadows only. *Why is that do you think?*
 - Soft shadows can be achieved using approximation techniques that give the appearance of sampling an area light source (e.g. percentage closer filtering)
 - We are not going to discuss soft shadows in this class.

Shadows

- Techniques for hard shadows
 - Shadow mapping (projective & depth)
 - Shadow volumes (not going to talk about)
 - All variations are a tradeoff of efficiency and accuracy
 - The term *hard* is used because the result is binary, either a point is lit or it is not (in shadow or not).
 - It is also useful to remember that there is no such thing as a point light source in the real world.
 - It is an approximation to simplify the computation in order to achieve interactive frame rates – that is all.
 - Even a light bulb is not a *point* light source!!

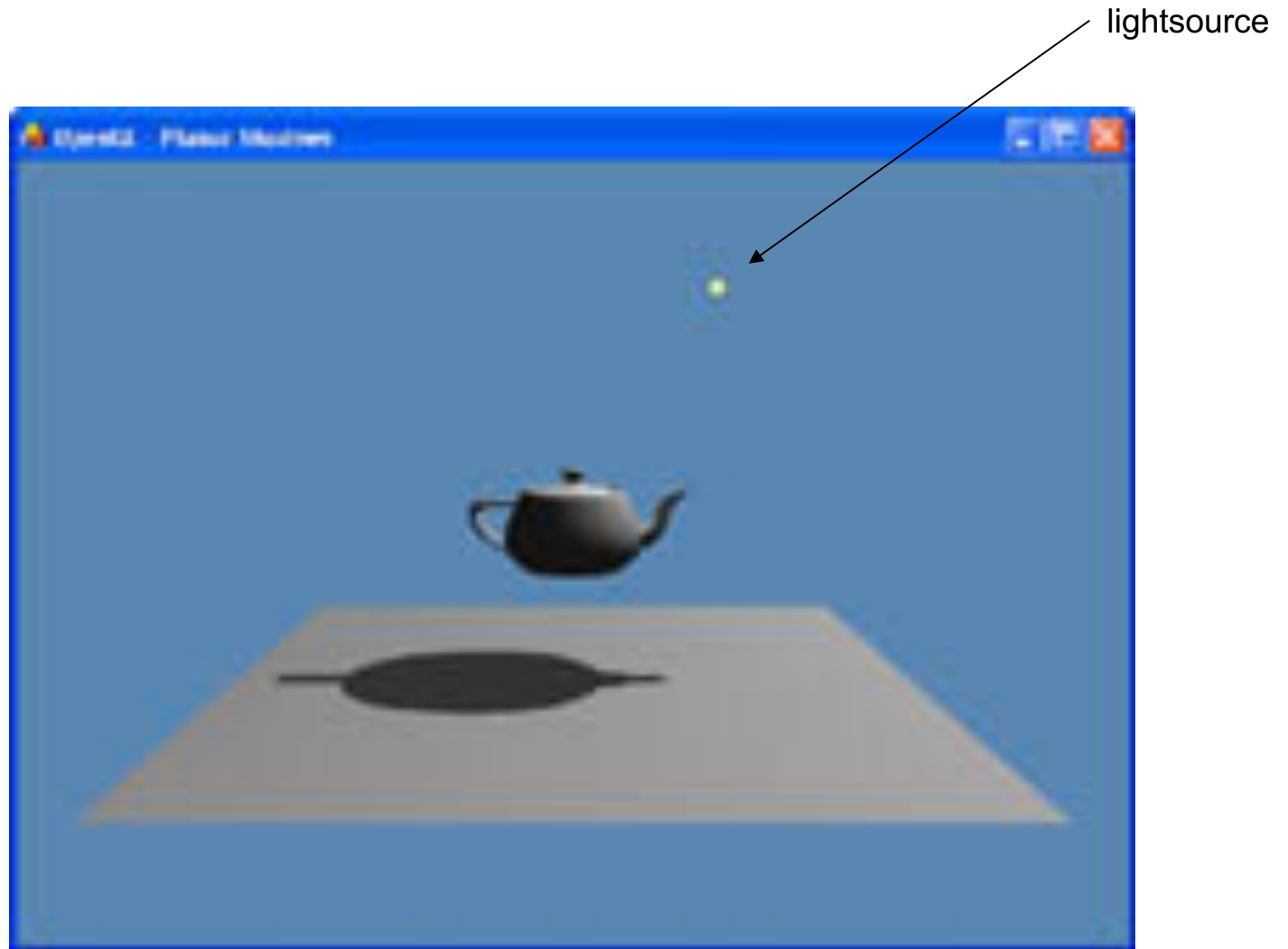
Shadows

- Planer projective shadows
 - Discussed briefly in the book
 - Projection of a shadow onto a planar (flat) surfaces only!
 - Not very useful.



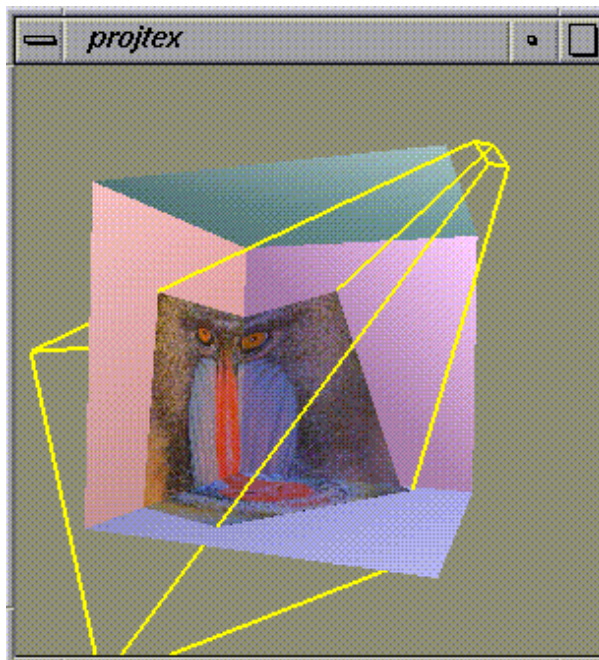
lightsource

Shadows

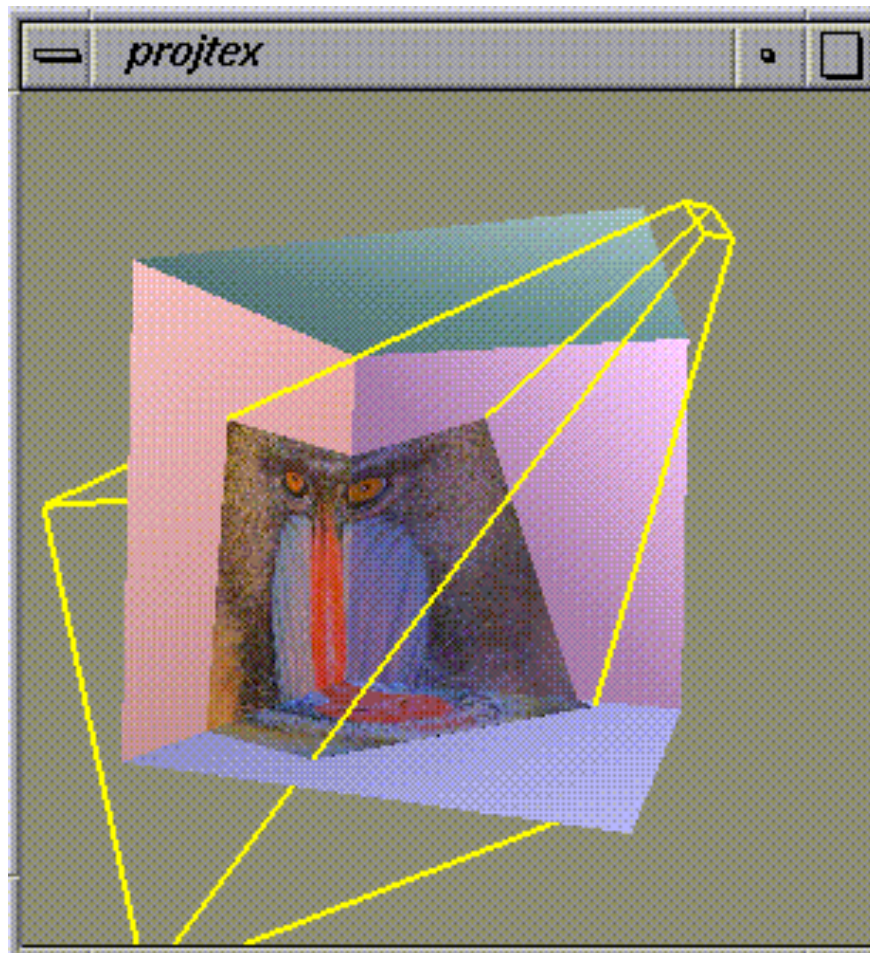


Shadows

- Projective shadow texture
 - A variation of projective textures
 - Behaves like a projector

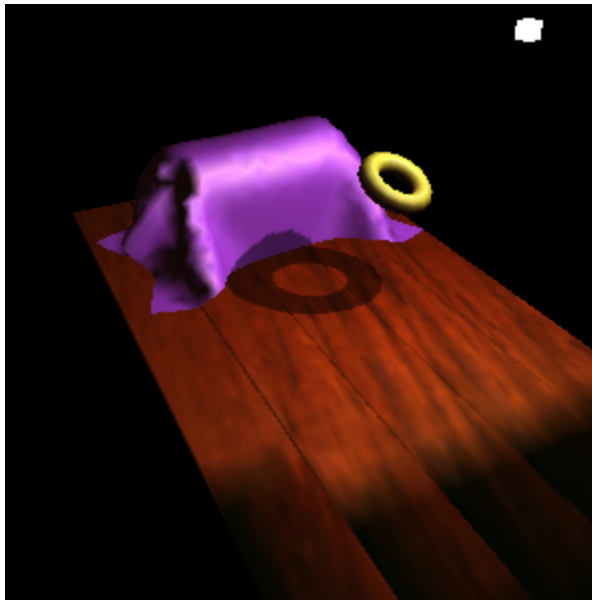


Shadows

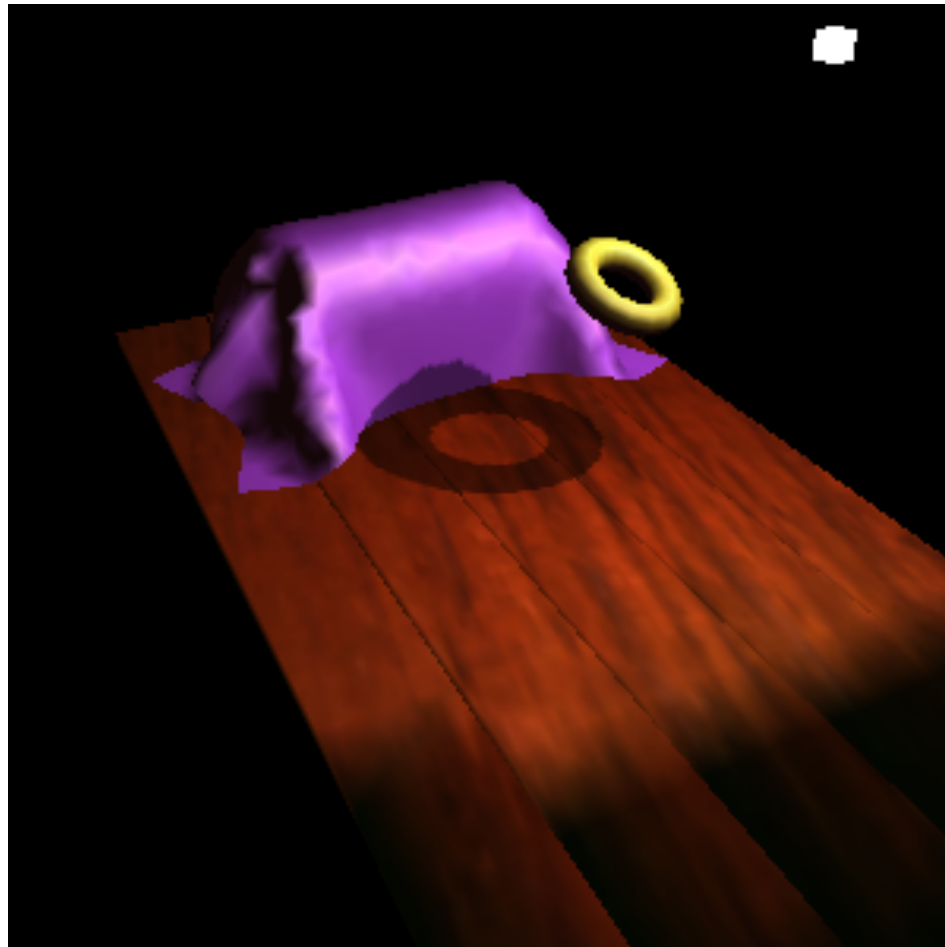


Shadows

- Projective shadow texture
 - Not limited to planar receiver surfaces.
 - Think of a projector at the light source casting an image of the shadow onto the receivers.

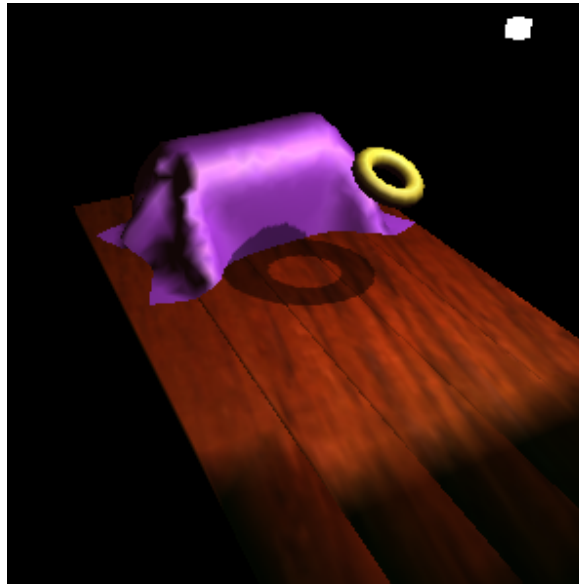


Shadows



Shadows

- Projective shadow texture
 - A buffer (texture) is cleared to white.
 - **Occluders** are rendered in black into the buffer.
 - A single texture lookup can tell us if receiver in shadow or not.



Shadows

- Projective shadow texture
 - In order to render into the texture we have to setup our camera in *light space (position of light source)*, as opposed to *camera space*.
 - Conveniently, here is actually a good use of the LookAt() function to setup the appropriate projection matrix.
 - We position the ‘camera’ at the location of the light source.
 - The camera is pointed in the direction of the shadow.
 - The result is a transformation matrix that transforms a vertex from world space to light space.

Shadows

- Projective shadow texture
 - We next need a projection matrix that will transform our points into *light clip space*.
 - Orthographic or perspective transform can be used.
 - Common to use an orthographic projection for infinite light sources, like the sun.
 - A point or directional light source would use a perspective transform.
 - d is the distance from the light source to the projection plane.

$$M_V^L = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & d & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Shadows

- Projective shadow texture
 - However, since we are in clip space we want everything to map to $[-1,1]$ so d is typically 1 which projects to $z=-1$
 - Keeping x and y in that same range requires a small change to our projection matrix adding the additional scaling values.
 - w and h are the dimensions of our shadow texture buffer.

$$M_P^L = \begin{bmatrix} w & 0 & 0 & 0 \\ 0 & h & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Shadows

- Projective shadow texture
 - Not specifically useful for this technique, but for others, is to maintain the depth information rather than projecting everything to $z=-1$
 - this results in a very familiar looking perspective projection matrix and you can use the `Perspective()` function.
 - Similar to the results you get with a “Kinect” type device.

$$M_P^L = \begin{bmatrix} \frac{1}{\alpha} \cot \frac{fovy}{2} & 0 & 0 & 0 \\ 0 & \cot \frac{fovy}{2} & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Shadows

- Projective shadow texture
 - In the *vertex* shader, when rendering the receivers, texture coordinates into the shadow texture are computed as v^s .
 - Furthermore, we need to adjust the light clip values from $[-1, 1]$ to conventional texture coordinate range of $[0, 1]$

$$M_t = \begin{bmatrix} 0.5 & 0 & 0 & 0.5 \\ 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- The entire projection can be baked into M_s

$$M_s = M_t M_p^L M_v^L$$

$$v^s = M_s v$$

Shadows

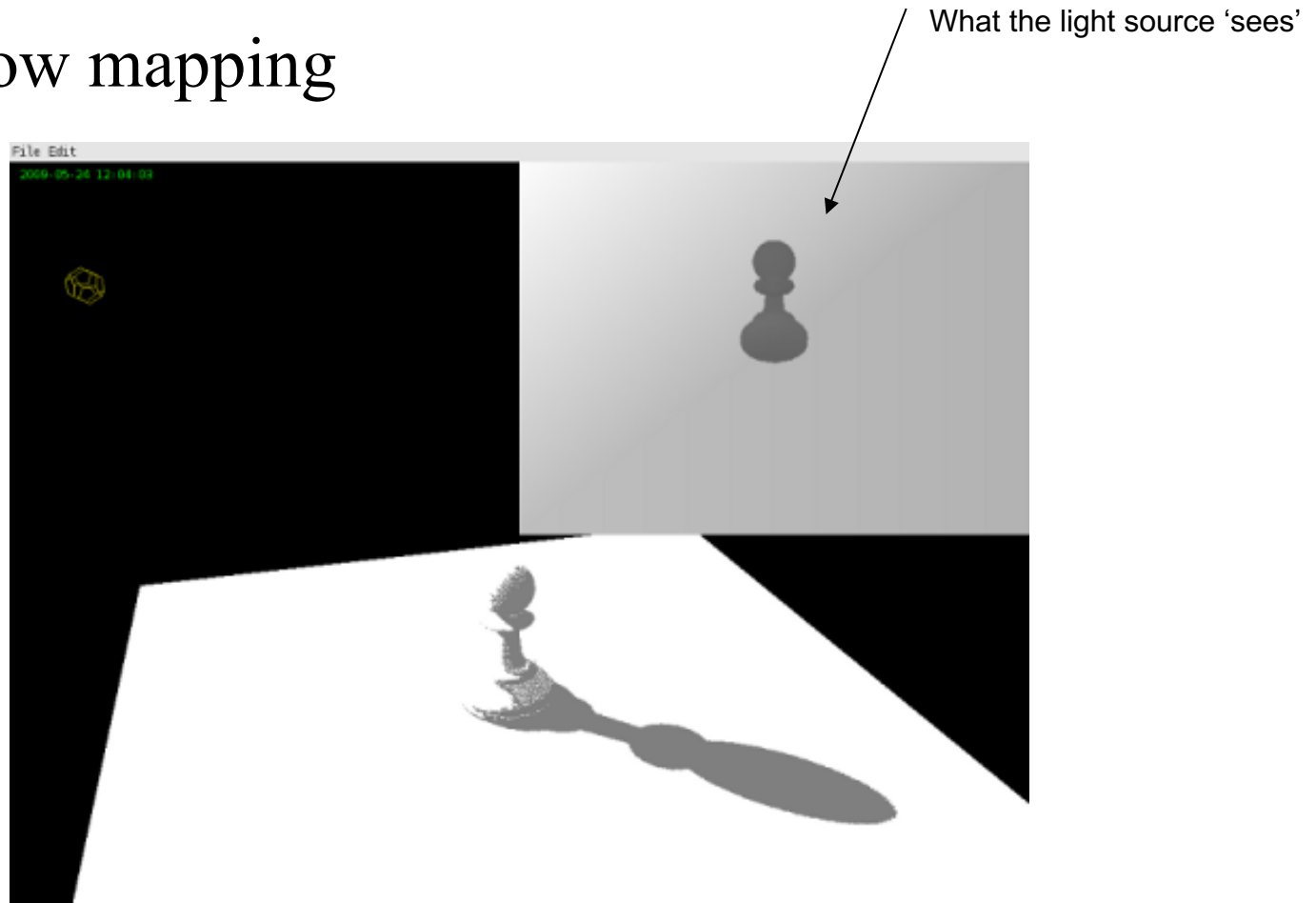
- Projective shadow texture
 - Values that fall outside of $[0,1]$ are never shadowed.
 - This can be checked for in the fragment shader.
 - Many issues, bad and good
 - Occluders and receivers must be known and separated
 - Requires shadow texture per occluder
 - No self shadowing
 - Achieve a form of soft shadow by filtering the texture map.
 - » Many techniques have been developed.
 - Often called a *light attenuation map*, which is used to bake shadows of static lights, occluders and receivers into a scene.
 - Overall a simple technique

Shadows

- **Shadow mapping**
 - More general form of projective texture shadows
 - No need to separate occluders from receivers
 - Can handle self-shadowing
 - Here we, again, render *from the position of the light source*.
 - However, the *entire* scene is rendered into the shadow map.
 - Every point that is rendered is, implicitly, lit. (what a light source can ‘see’)
 - Anything else is in shadow.
 - Determining whether a 3d position is in shadow or not becomes a matter of checking whether it is lit in the shadow map or not.

Shadows

- Shadow mapping



Shadows

- Shadow mapping
 - Simple in theory, harder to do nicely in practice.
 - Image space based so artifacts are a fact of life.
 - Shadow map resolution can result in jagged shadow edges
 - i.e. shadow map is lower resolution than display (image space)
 - We are really only interested in the depth map created by rendering from the light's position.
 - Here each fragment's position p is transformed into *light clip space* as before.
 - The x and y components index into the depth map.
 - The z value is the distance from the light source.
 - » Not forced to -1 as before

Shadows

- Shadow mapping
 - Recall that the actual x and y values need to be scaled like the projective texture was to $[0,1]$, or p^S
 - In any case, the z value is compared to the depth value of the point under consideration.
 - If the point in light clip space has a depth value greater than the point's value in the shadow map the fragment is hidden, or in shadow.
 - Otherwise it is lit and shaded normally.

Shadows

- Shadow mapping
 - The fact that we have been producing these maps via projection and rendering may make you wonder...
 - Does this work only for directional lights?
 - No, point light sources involve rendering six frustums around the light source – just like a cube map.
 - Again, spherical maps are more efficient in terms of pixels and rendering geometry but cube maps are simpler.
 - Others have suggested using tetrahedrons as a compromise.

Shadows

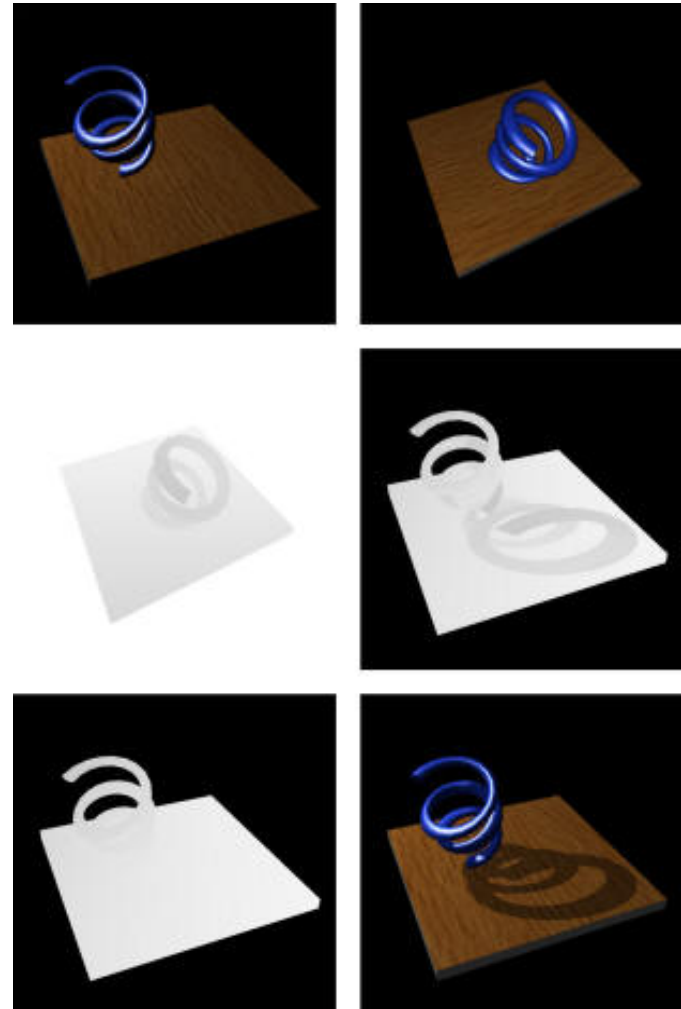
- Shadow mapping
 - Another more direct issue is related to sampling the shadow map buffer during depth comparison.
 - The shadow map has a limited precision to represent depth values.
 - The shadow map has sampled the scene at a different resolution than the camera, typically. (texture map vs screen resolution)
 - Rarely does the point in eye space that we have projected into light space correspond to an exact depth map sample.
 - This can lead to light “leaks” due to *z-fighting*.
 - This occurs, particularly, when the receiver is tilted and the discretization of the depth values result in an incorrect comparison.

Shadows

- Shadow mapping
 - The “solution” to this problem is to introduce a *bias*.
 - This bias value pushes the depth values away from the light source slightly.
 - The problem is that there is no rule of thumb for deciding what the bias value should be – experimentation is required.
 - One approach is to use the slope of the receiving surface as a guide
 - » The smaller the angle between light and surface normal – the larger the bias.
 - Another is to render the scene into the shadow map using back faces only (the opposite of what you usually do). However this only really has an effect when all models have some ‘thickness’
 - Basically the same problem when trying to render coplaner geometry – outlining geometry like assignment 1.

Shadows

- Shadow mapping
 - example
 - View from camera (tl)
 - View from light (tr)
 - Shadow map from light (ml)
 - Shadow map from camera (mr)
 - Scene depth (bl)
 - Final image after compare (br)



Shadows

- Shadow mapping
 - You may have noticed that all this assumes we can render to a texture.
 - In the depth map case we are rendering depth values to a texture.
 - This is accomplished using `gl.DEPTH_COMPONENT` when defining the texture in WebGL

Shadows

- Shadow mapping
 - <http://blog.tojicode.com/2012/07/using-webgldepthtexture.html>