CSC 240 Computer Graphics
Video 16: More Mapping

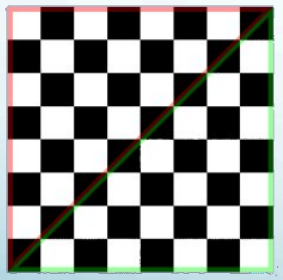
Nick Howe
Smith College

Some slides & content courtesy Sara Mathieson

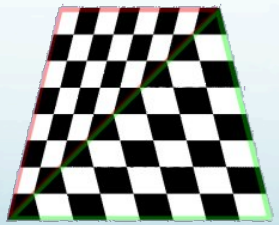
Perspective-Correct Texture Mapping

Simple interpolation of textures produces weird artifacts! Why?

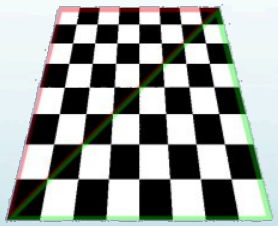
- Perspective makes x & y shrink, but not z
- Solution: Need to adjust for z coordinate as we interpolate u,v



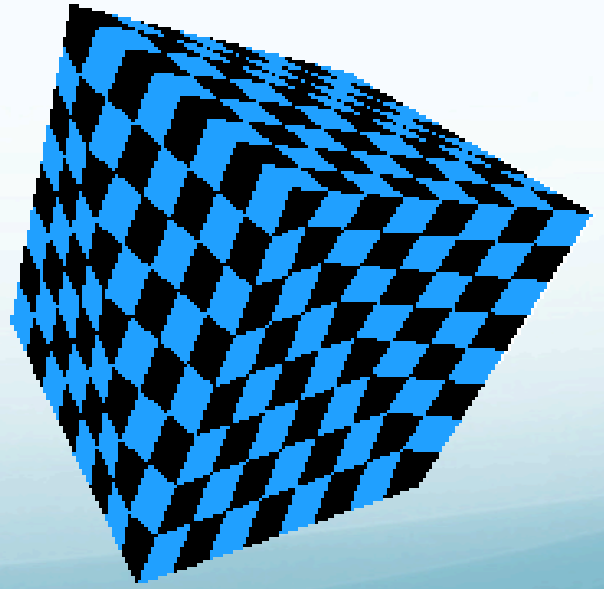
Flat



Affine



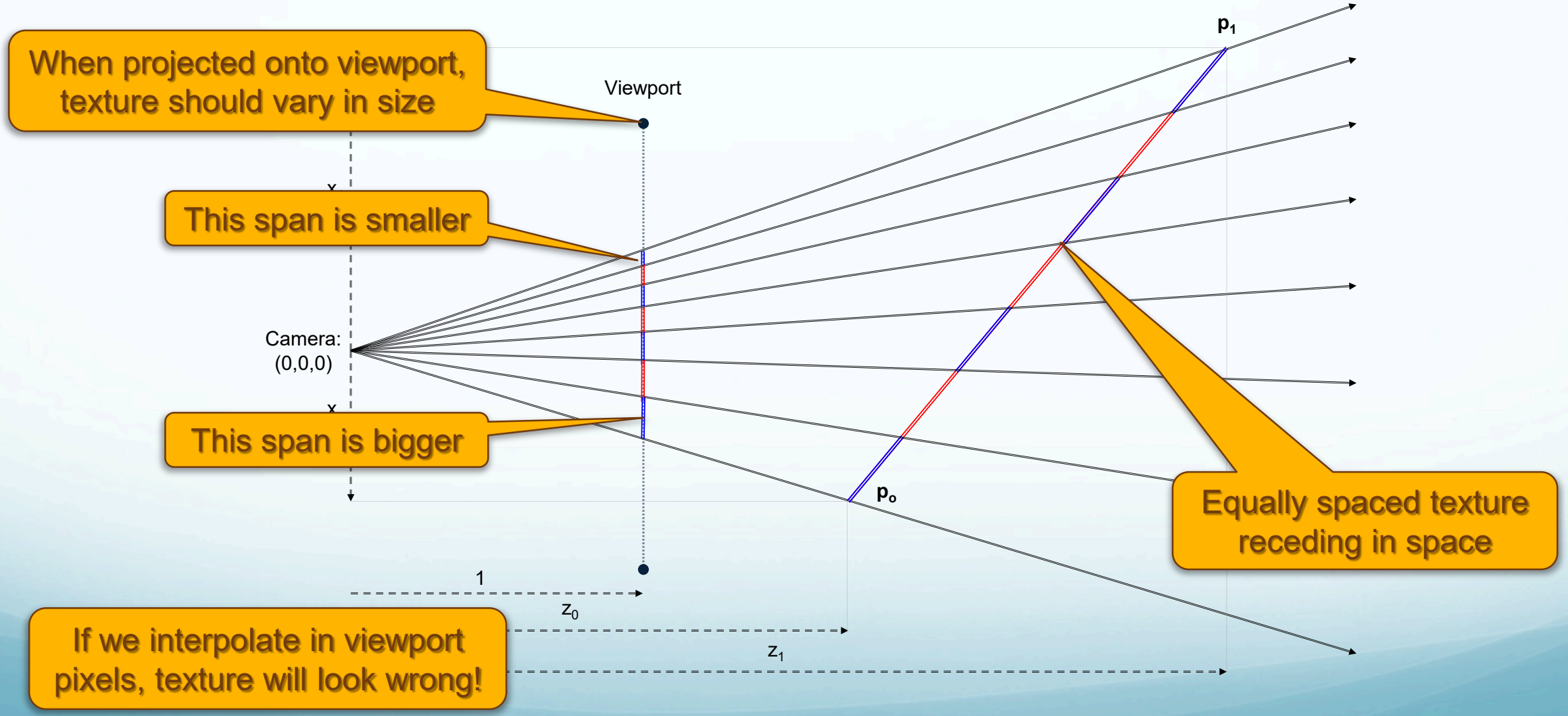
Correct



https://en.wikipedia.org/wiki/Texture_mapping

http://www.lysator.liu.se/~mikaelk/doc/perspectivetexture/

Perspective Correct Texture Mapping



Perspective Correct Texture Mapping

Values that are linear in screen coordinates: $\frac{1}{z}, \frac{u}{z}, \frac{v}{z}$

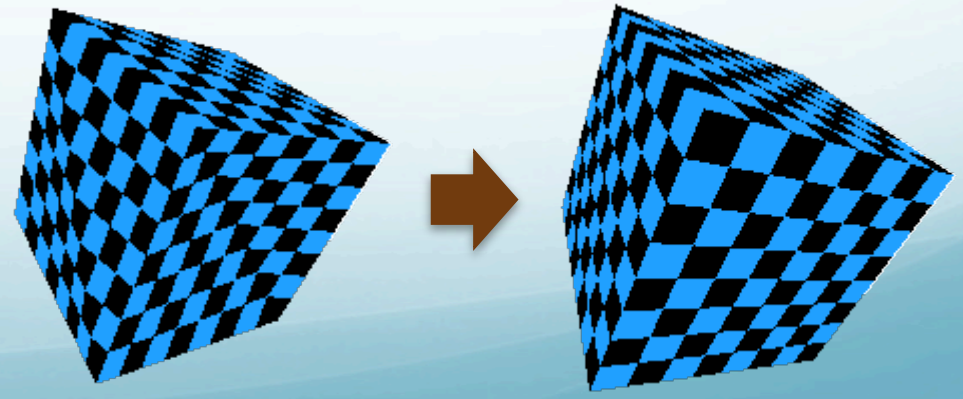
- Interpolate these, then use to compute u and v

$$u^* = \frac{\alpha\frac{u_1}{z_1}+\beta\frac{u_2}{z_2}+\gamma\frac{u_3}{z_3}}{\alpha\frac{1}{z_1}+\beta\frac{1}{z_2}+\gamma\frac{1}{z_3}},\ v^* = \frac{\alpha\frac{v_1}{z_1}+\beta\frac{v_2}{z_2}+\gamma\frac{v_3}{z_3}}{\alpha\frac{1}{z_1}+\beta\frac{1}{z_2}+\gamma\frac{1}{z_3}}$$

Barycentric interpolation of $\frac{v}{z}$

Barycentric interpolation of $\frac{1}{z}$

- This gives perspective-correct values for u, v
- Automatic in modern graphics
- Similar form for scanline interpolation



Questions

PAUSE NOW & ANSWER

1. Suppose you have a graphics system with texture mapping that doesn't correct for perspective. Under what circumstances would it still produce the correct result for a given polygon?

   If all corners of the polygon have the same depth (z value).

2. Suppose that $\alpha = 0.5, \beta = 0.5, \gamma = 0$ for point p. Also $u_1 = 1$, $z_1 = 1$, $u_2 = 2$, $z_2 = 2$, $u_3 = 3$, and $z_3 = 3$. What is the uncorrected value of u_p?

$$u_p = \alpha u_1 + \beta u_2 + \gamma u_3 = 0.5 \cdot 1 + 0.5 \cdot 2 + 0 \cdot 3 = 1.5$$

3. What is the corrected value?

$$u^* = \frac{\alpha \frac{u_1}{z_1} + \beta \frac{u_2}{z_2} + \gamma \frac{u_3}{z_3}}{\alpha \frac{1}{z_1} + \beta \frac{1}{z_2} + \gamma \frac{1}{z_3}} = \frac{0.5\frac{1}{1} + 0.5\frac{2}{2} + 0\frac{3}{3}}{0.5\frac{1}{1} + 0.5\frac{1}{2} + 0\frac{1}{3}} = \frac{1.0}{0.75} = 1.33$$

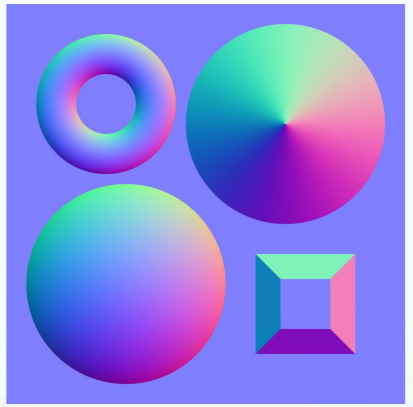
Normal Mapping

$$x = \frac{r}{127.5} - 1$$
$$y = \frac{g}{127.5} - 1$$
$$z = \frac{b}{127.5} - 1$$

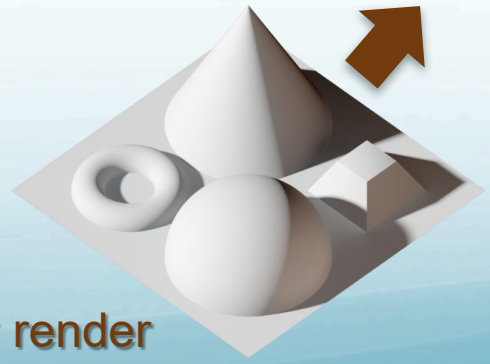
Texture (pixel color) isn't the only thing we can interpolate.

- For realistic shading of rough surfaces, use a **normal map**
  - Surface normal vectors are stored as RGB image
  - Red component, 0-255 → X coordinate, -1 to +1
  - Green component, 0-255 → Y coordinate, -1 to +1
  - Blue component, 128-255 → Z coordinate, 0 to +1

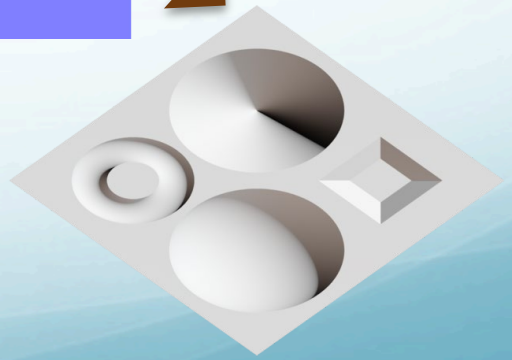
  E.g.: (128,128,255) → (0,0,1) - points towards viewer
- Combine normal vectors from map with geometry to get light-dependent shading
- Derive map from high-poly render



Normal map



High-poly render



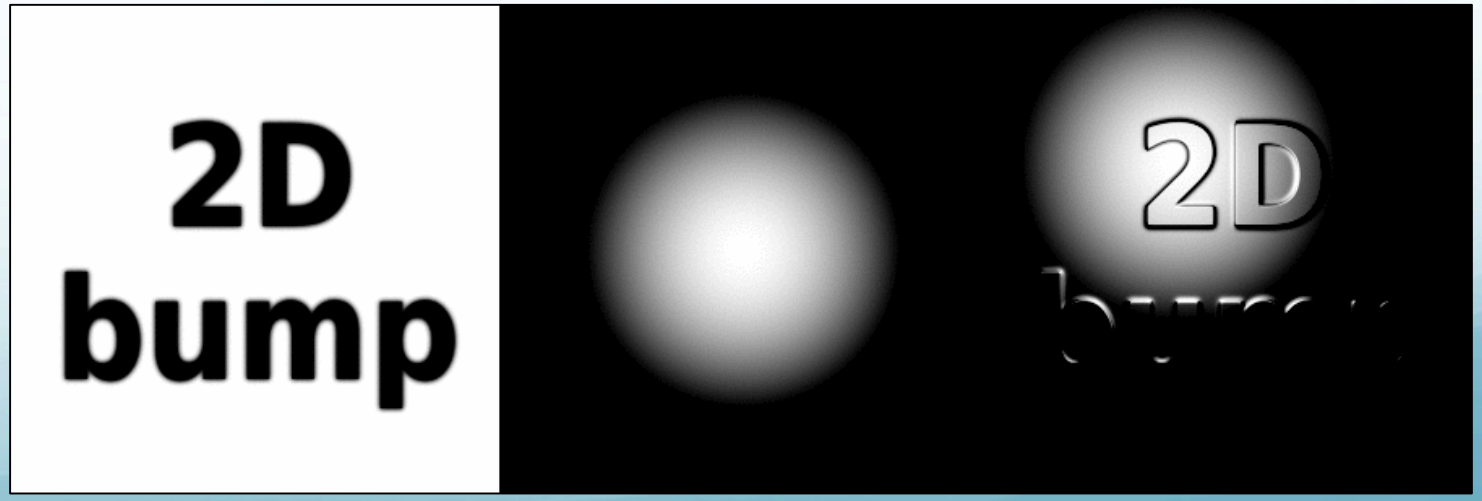
Rendered surface

wikipedia

Bump Mapping

Bump map is an alternative to a normal map.

- Single value at each point: texture height (stored as grayscale image)
- Normal vectors can be inferred form height changes between pixels
- 1/3 the storage of RGB normal map
- Somewhat less flexible
- Pick one or the other – they do similar things

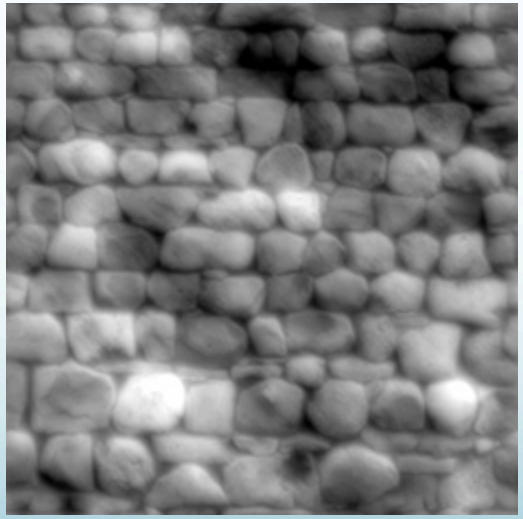
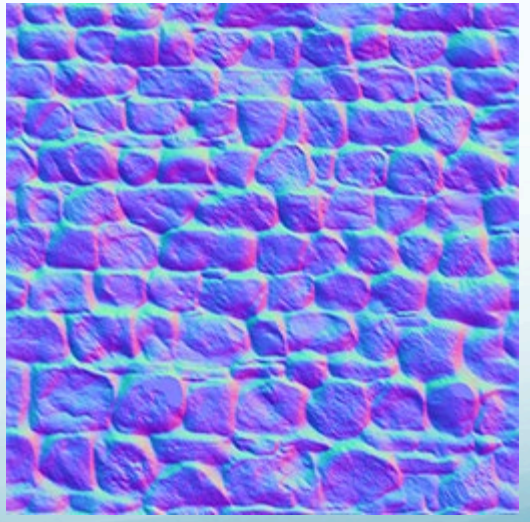


wikipedia

Normal Mapping Demo

Easy to add a texture map in Three.js:

```
cubeNormMaterial.map = loader.load("StoneWallTexture.png");
cubeNormMaterial.bumpMap = loader.load("StoneWallBump.png");
cubeNormMaterial.normalMap = loader.load("StoneWallNormals.png");
//…
bumpCube.material.bumpScale = 0.5;
```

Questions

PAUSE NOW & ANSWER

1. Does using a bump map change the location where a pixel is drawn, based on the height stored in the bump map?

   No, the pixel is still drawn in the same place. Only the shading changes.

2. Which of the following can alter the shading of a pixel?
   a. Changing the polygon vertex coordinates (geometry)
   b. Changing the light position
   c. Changing the normal map

   All of the above

3. What is the normal vector represented by the RGB triplet (174,66,230)?

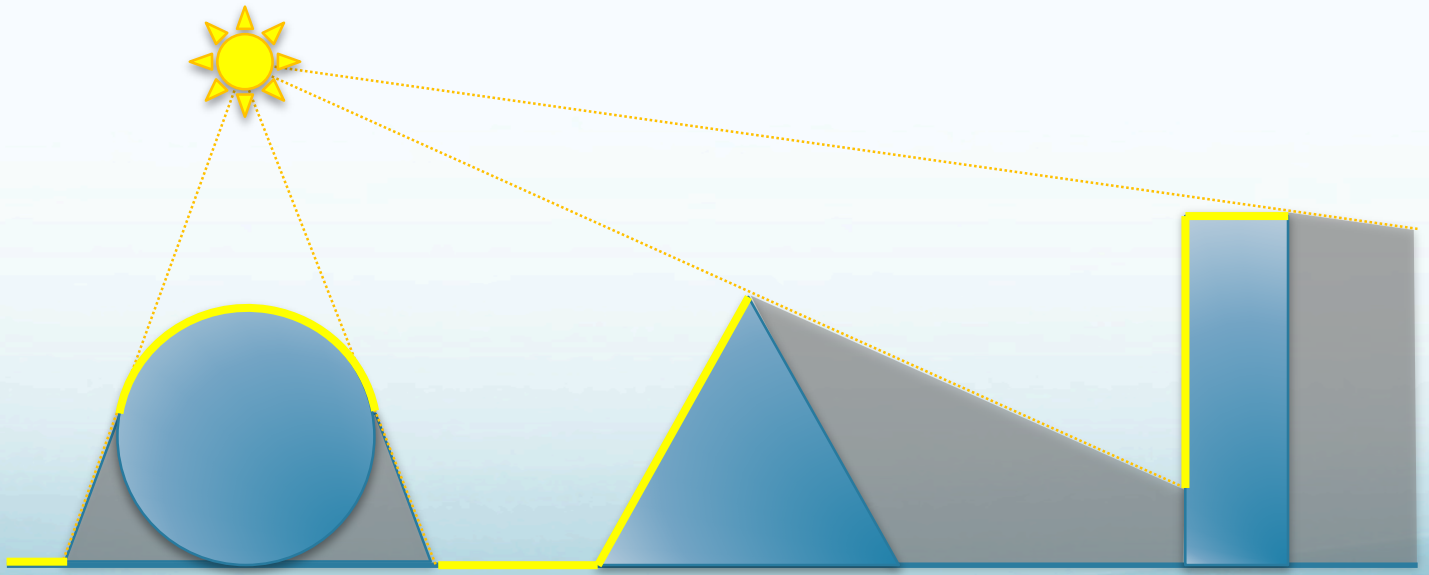
$$\left(\frac{174}{127.5}-1,\frac{66}{127.5}-1,\frac{230}{127.5}-1\right)=(0.36,-0.48,0.80)$$

Shadow Mapping

Shadow Mapping

How can we render shadows of objects on other objects?

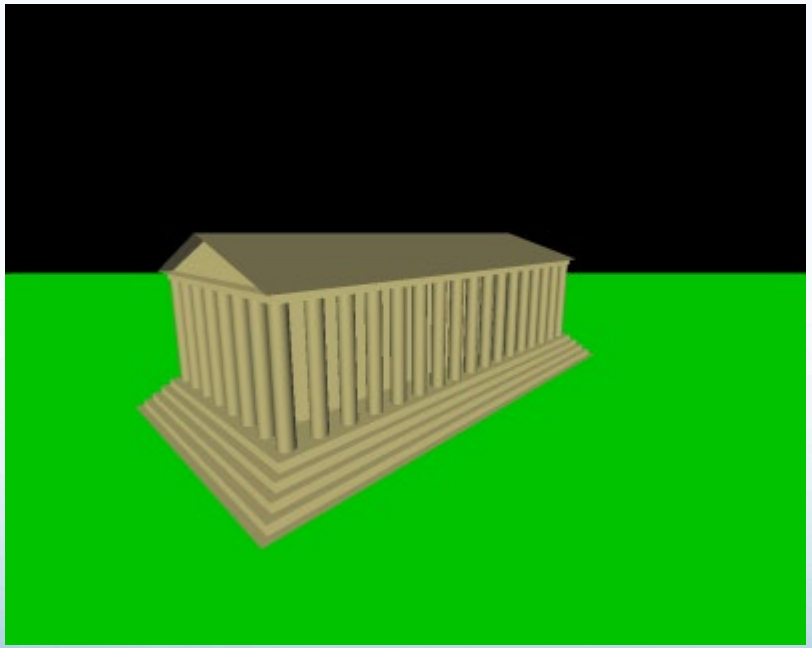
- Tricky problem: interactions = (number of objects)^2
- Reframe the problem: what surfaces are visible from a light source?



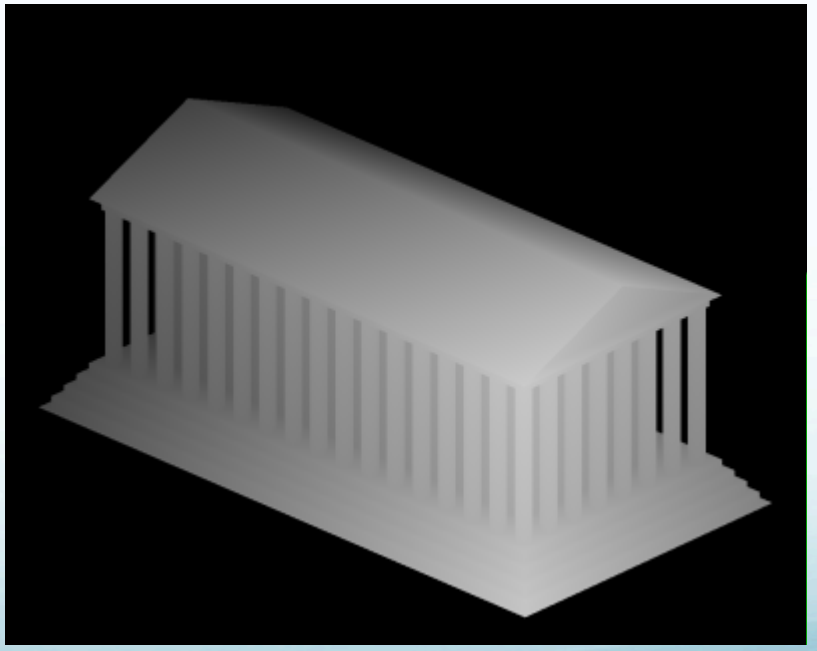
Shadow Mapping

Step one: Render scene as though light source is camera

- Only need depth – no lighting, texture
- (Use orthographic projection in case of directional light, perspective otherwise)



View from camera



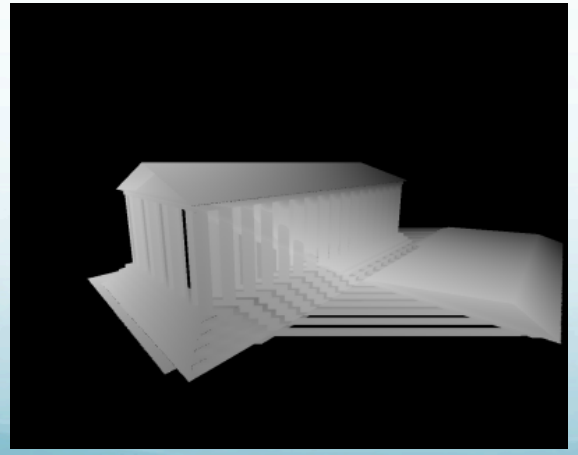
View from light

wikipedia

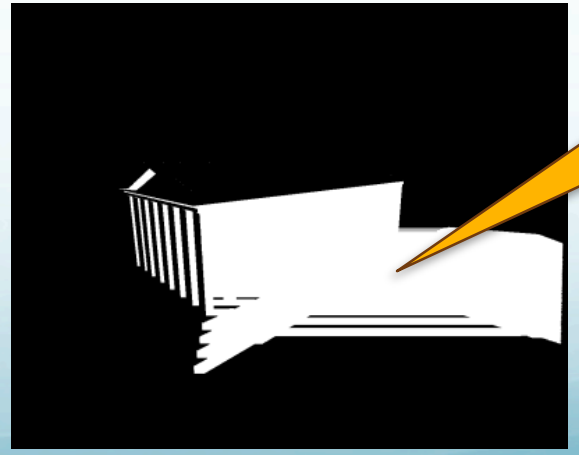
Shadow Mapping

Step two: Figure out shadow map

- Render from camera, computing world (x, y, z) for each visible point
- Convert (x, y, z) to light's camera frame (transformation matrix)
- Compare depth of camera-visible point to light-visible point
  - Roughly the same z: pixel is lit
  - Significant difference: pixel is in shadow



Depth map from light view

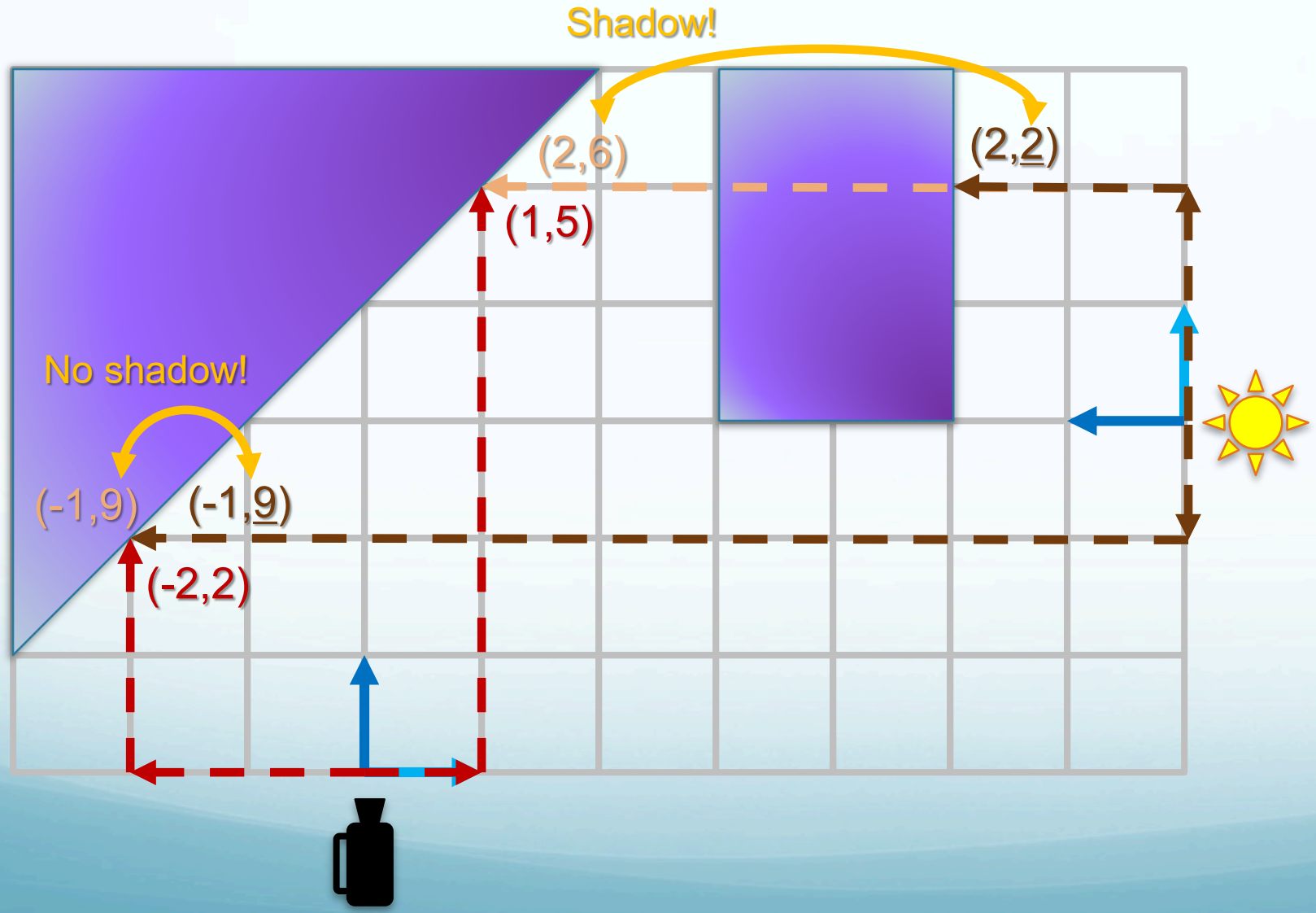


Depth mismatch

White areas indicate places where point seen by camera is not visible to light: shadows!

wikipedia

Shadow Mapping Example



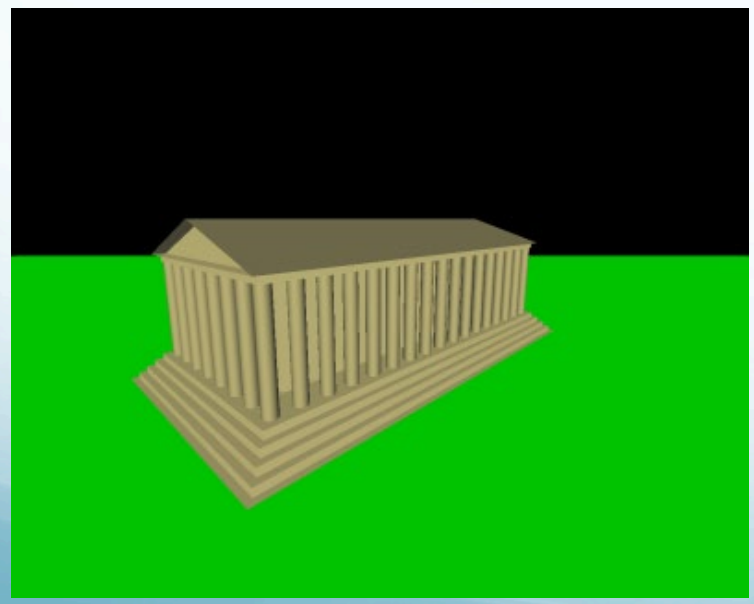
2D example (XZ plane) orthographic projection

1. Render scene from light.
2. Record z at each point
3. Render scene from camera
4. Record (x,z) at each point
5. Convert (x,z) to light's frame of reference
6. Compare to stored z values

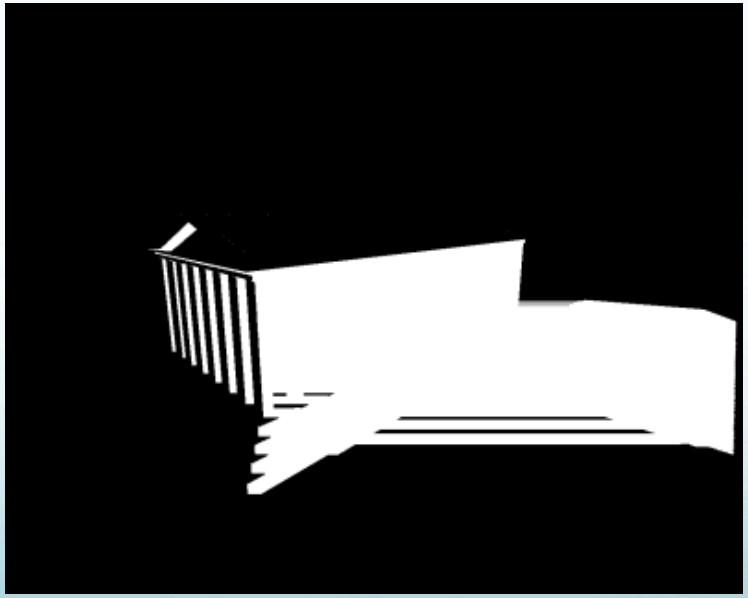
Shadow Mapping

Final step is to render scene with shadow

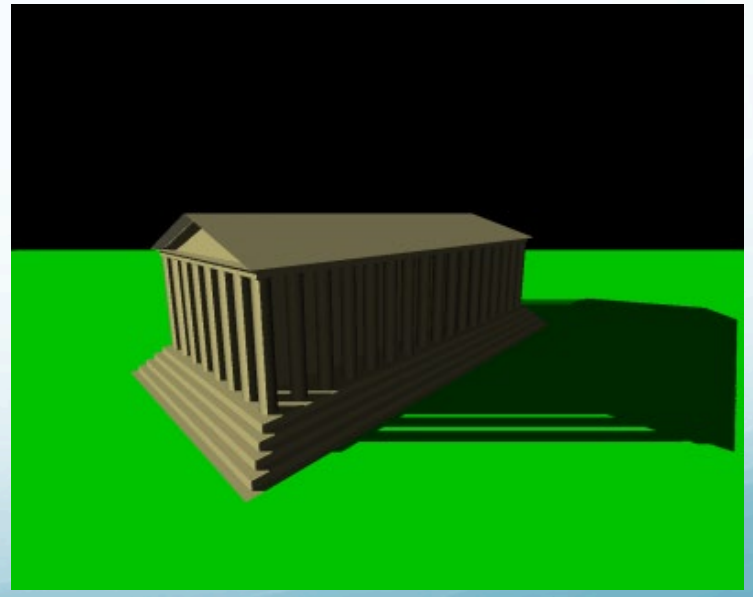
- Can treat shadow as a texture!
- Older way: render whole image with & without light; choose by pixel



Base render



Shadow pixels



Final render

wikipedia

Shadow Mapping in Three.js

Shadows must be enabled in several places in three.js:

1. Renderer must be told to render shadows

```
renderer.shadowMap.enabled = true;
renderer.shadowMap.type = THREE.PCFSoftShadowMap;
```

2. Lights must be told to cast shadows

```
light.castShadow = true;
```

3. Objects must be told to cast & receive shadows

```
obj.castShadow = true;
obj.receiveShadow = false;
```

4. Object material must be compatible with shadows

```
var material = new THREE.MeshPhongMaterial( { map: myTexture } );
```

Shadow mapping

Demo: https://threejs.org/examples/webgl_shadowmap.html

Questions

PAUSE NOW & ANSWER

1. Which sort of projection is necessary to compute shadows for the following types of light source?
   a. PointLight — *Perspective projection*
   b. AmbientLight — *None (ambient light casts no shadows)*
   c. DirectionalLight — *Orthographic projection*

2. In my Three.js program, I have turned shadows on in the renderer, told my objects to send and receive shadows, and used a shadow-compatible material. Shadows still aren't showing up. What did I forget?

   The light must also be told to cast shadows.

Review

After watching this video you should be able to…

- Explain the need for perspective correction in texture mapping
- Compute perspective-corrected UV coordinates
- Understand the normal map and bump map formats
- Use normal maps and/or bump maps to produce textured shading effects
- Describe the shadow mapping algorithm
- Carry out shadow computations in a simple model

Music: https://www.bensound.com