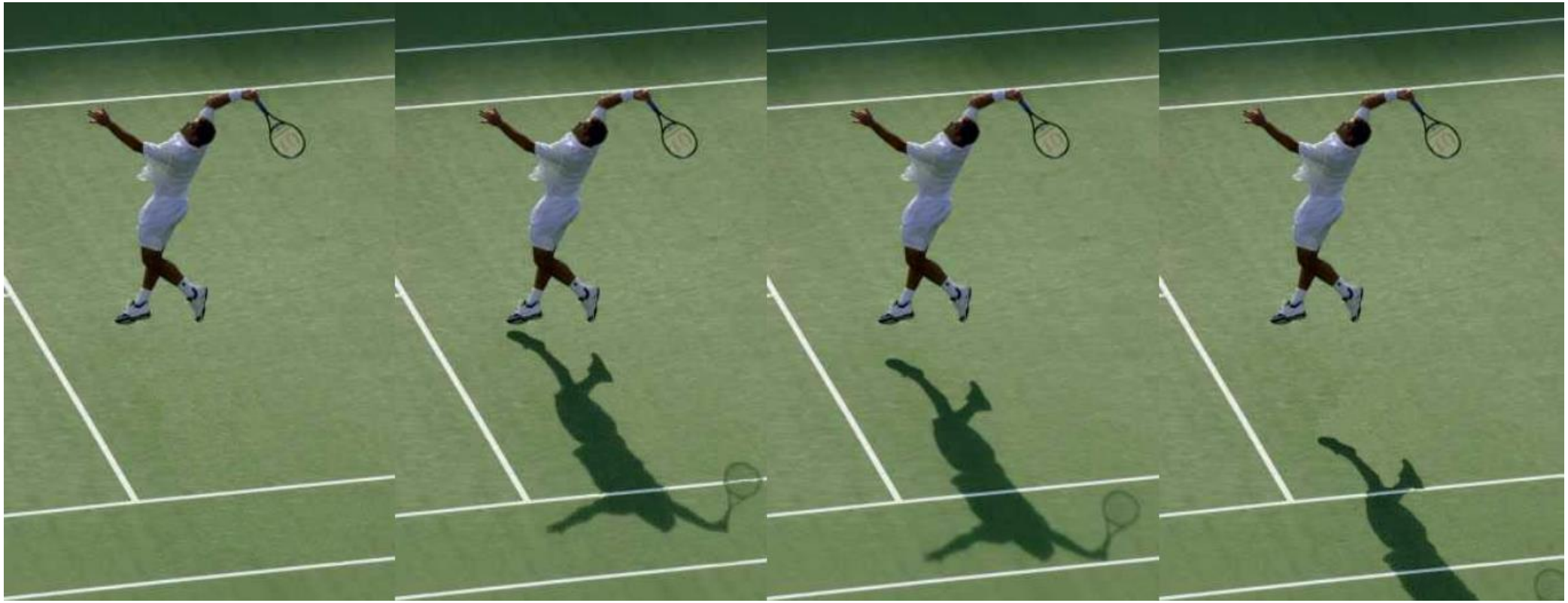
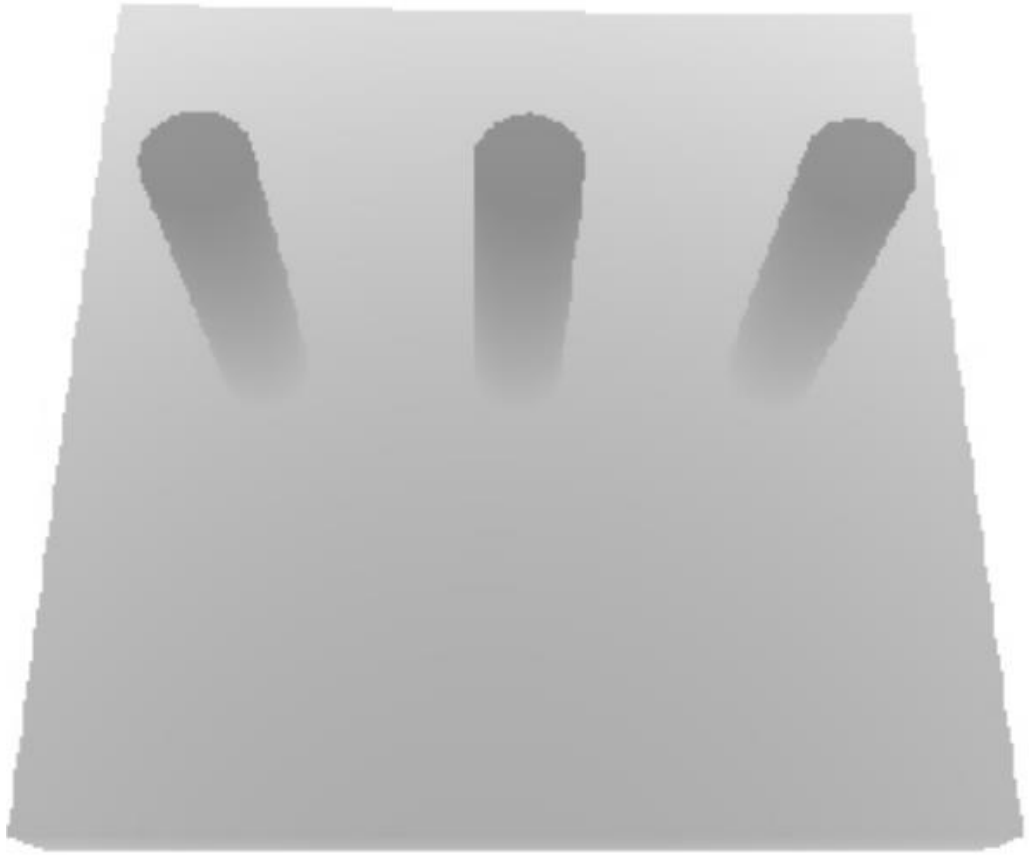
CGP 5

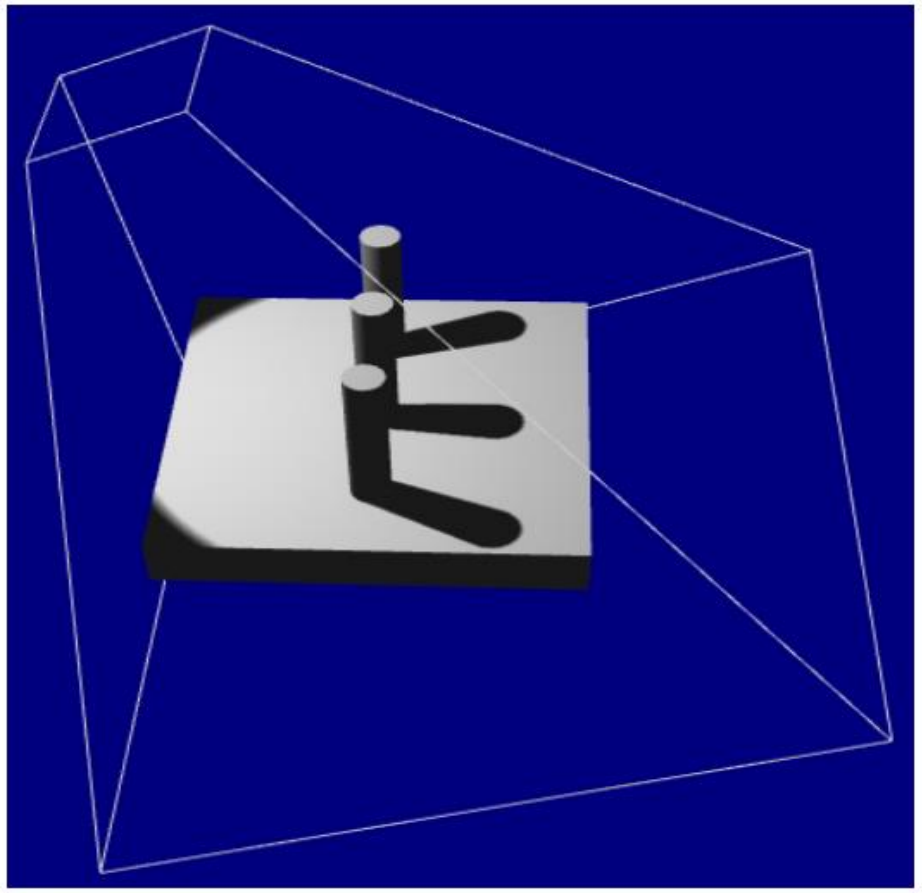
Dr Wojciech Palubicki

Shadows



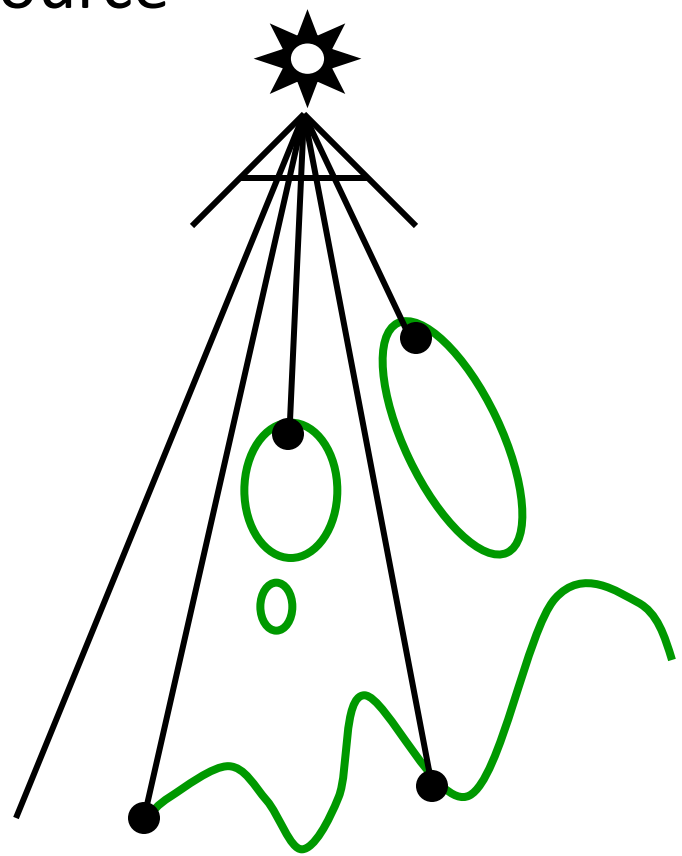
Shadow mapping





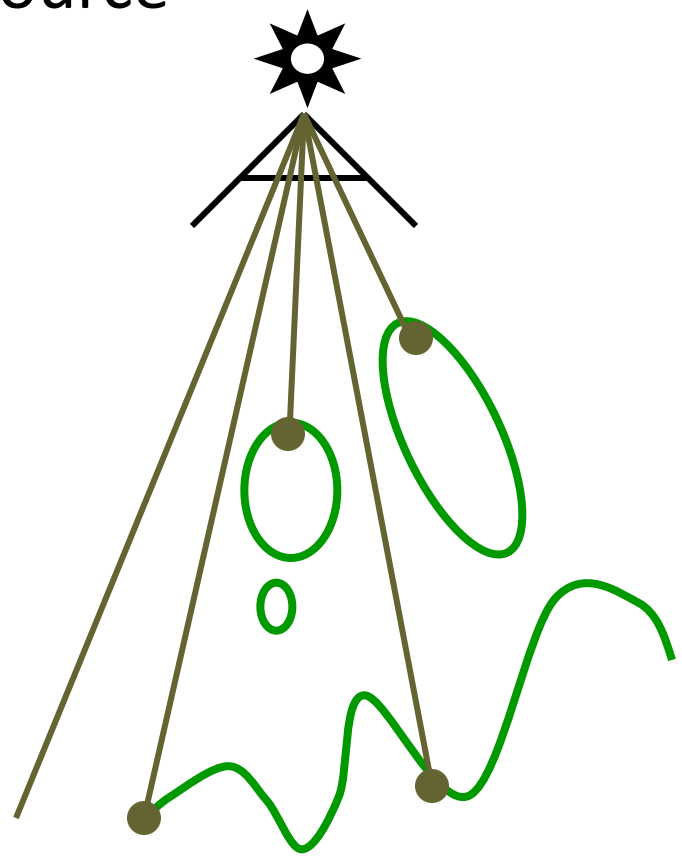
Phase 1: Render from Light

- Depth image from light source



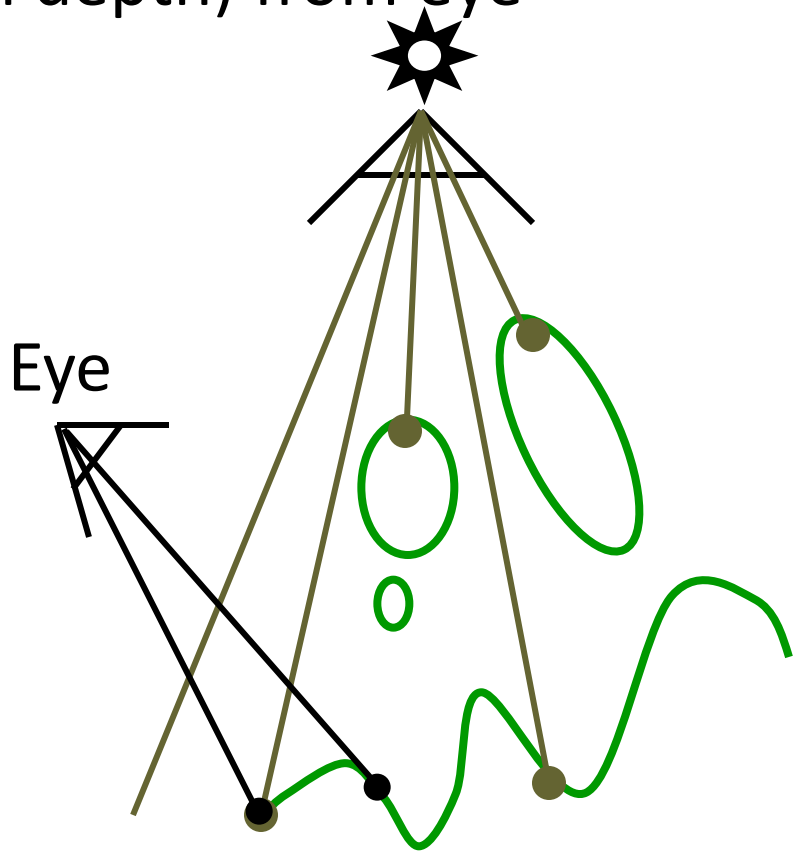
Phase 1: Render from Light

- Depth image from light source



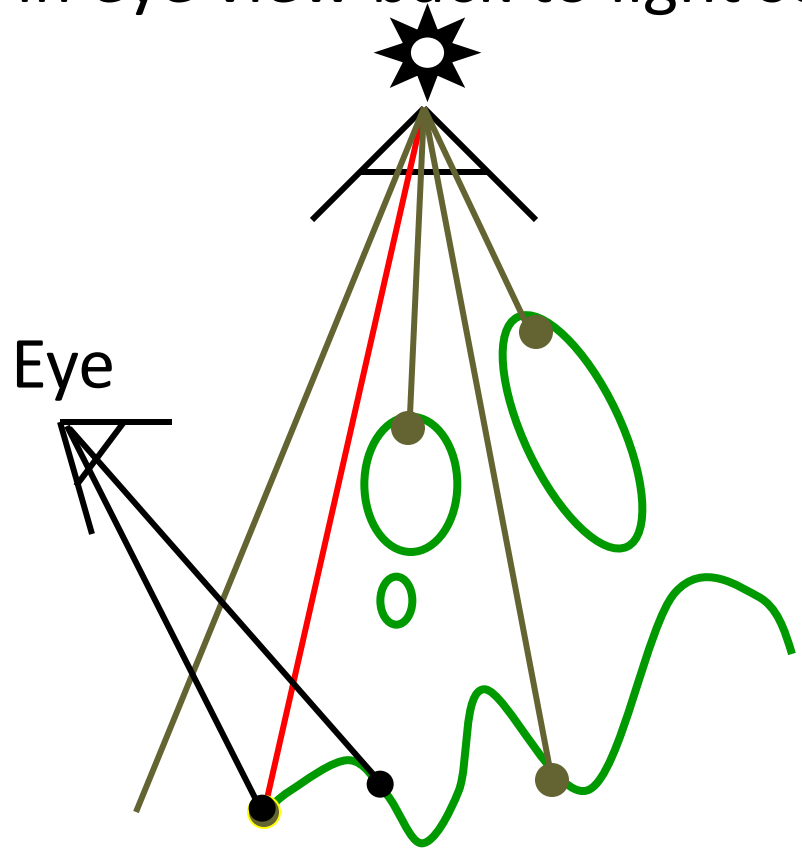
Phase 2: Render from Eye

- Standard image (with depth) from eye



Phase 2: Project to light for shadows

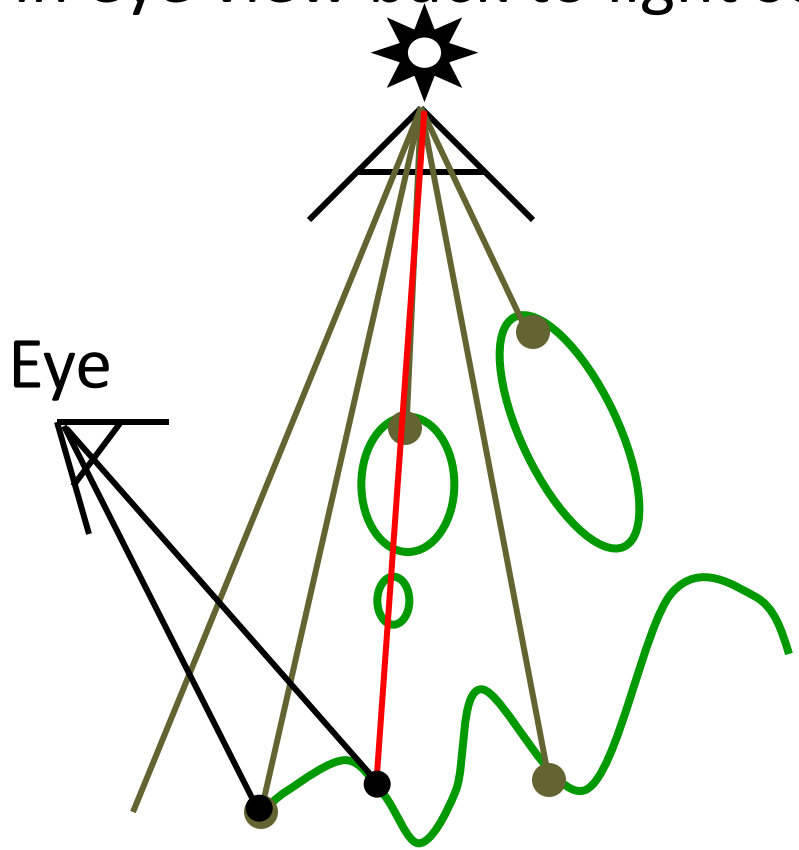
- Project visible points in eye view back to light source



Projected depths match for light and eye. VISIBLE

Phase 2: Project to light for shadows

- Project visible points in eye view back to light source

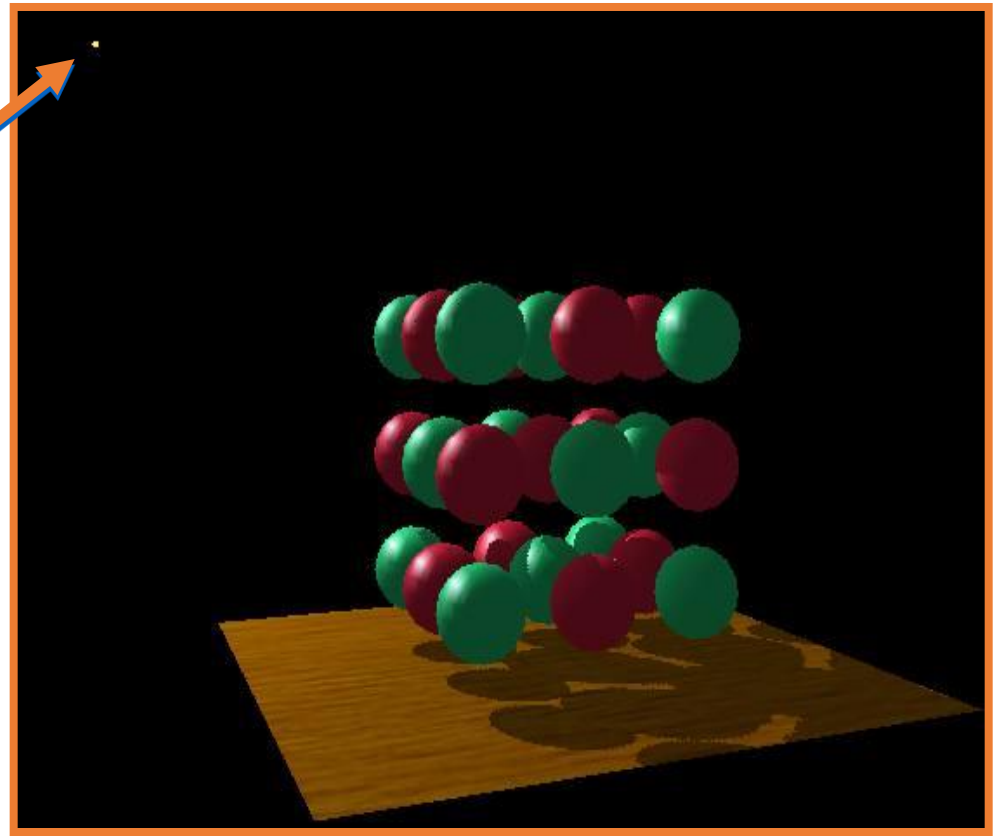


Projected depths from light, eye not the same. BLOCKED

Visualizing Shadow Mapping

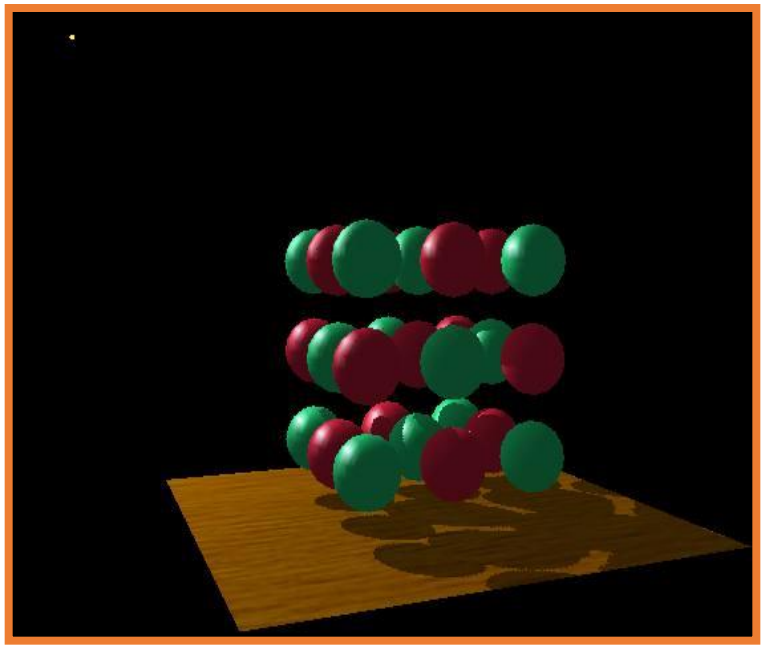
- A fairly complex scene with shadows

the point light source

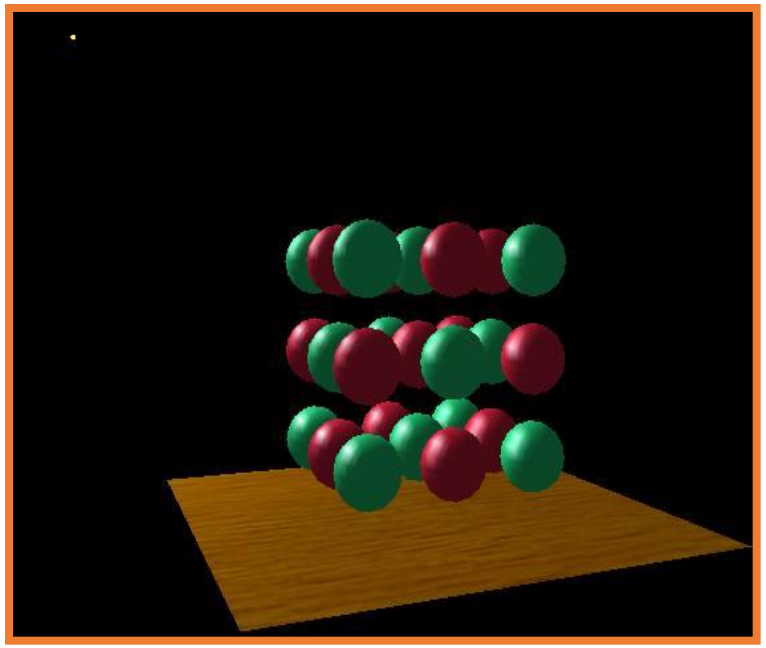


Visualizing Shadow Mapping

- Compare with and without shadows



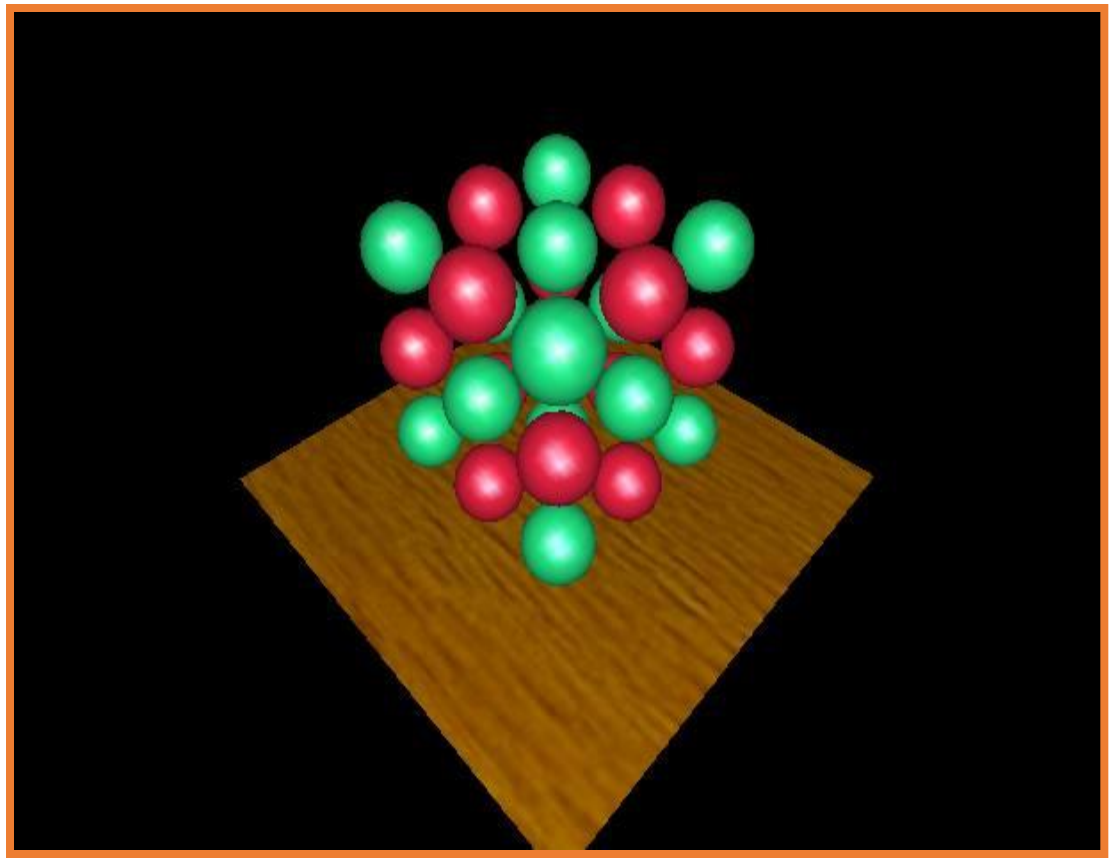
with shadows

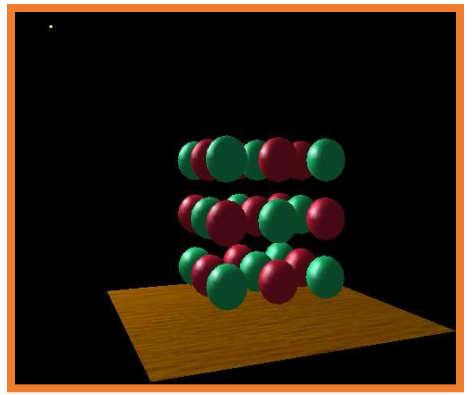


without shadows

Visualizing Shadow Mapping

- The scene from the light’s point-of-view

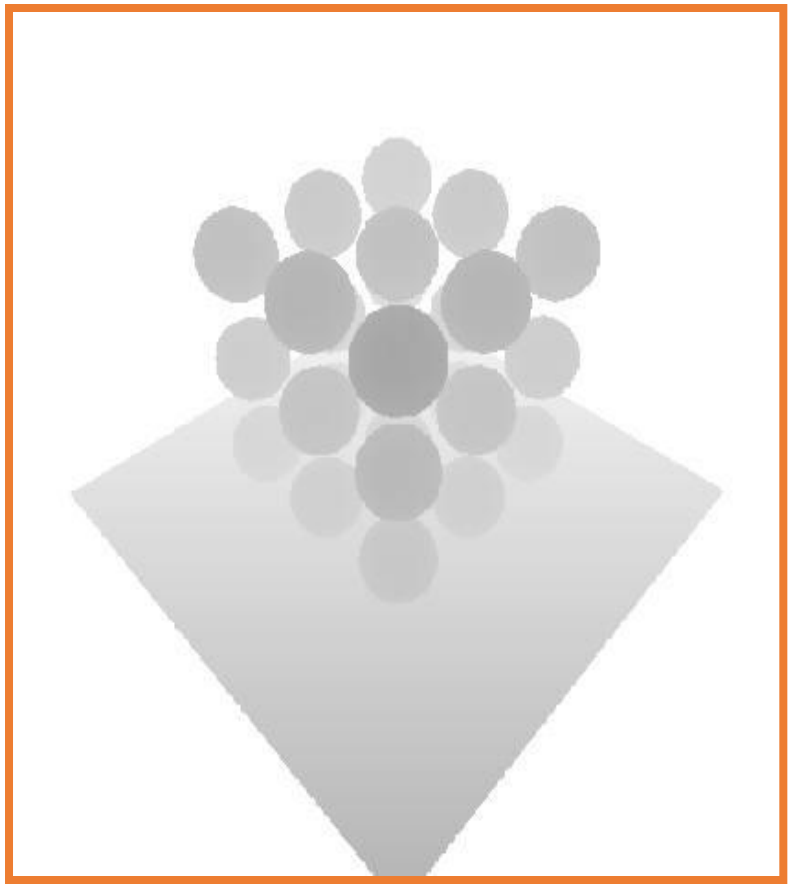


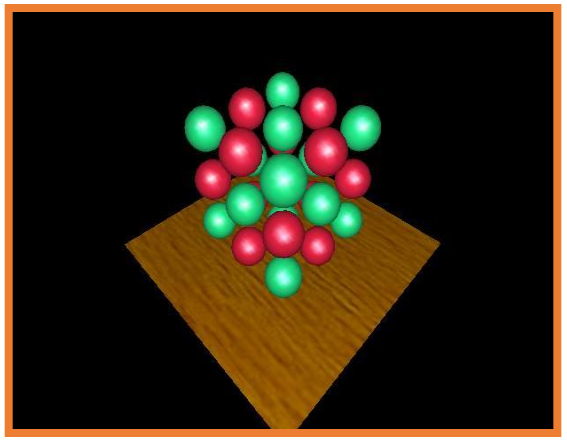


from the eye’s point-of-view again

Visualizing Shadow Mapping

- The depth buffer from the light's point-of-view

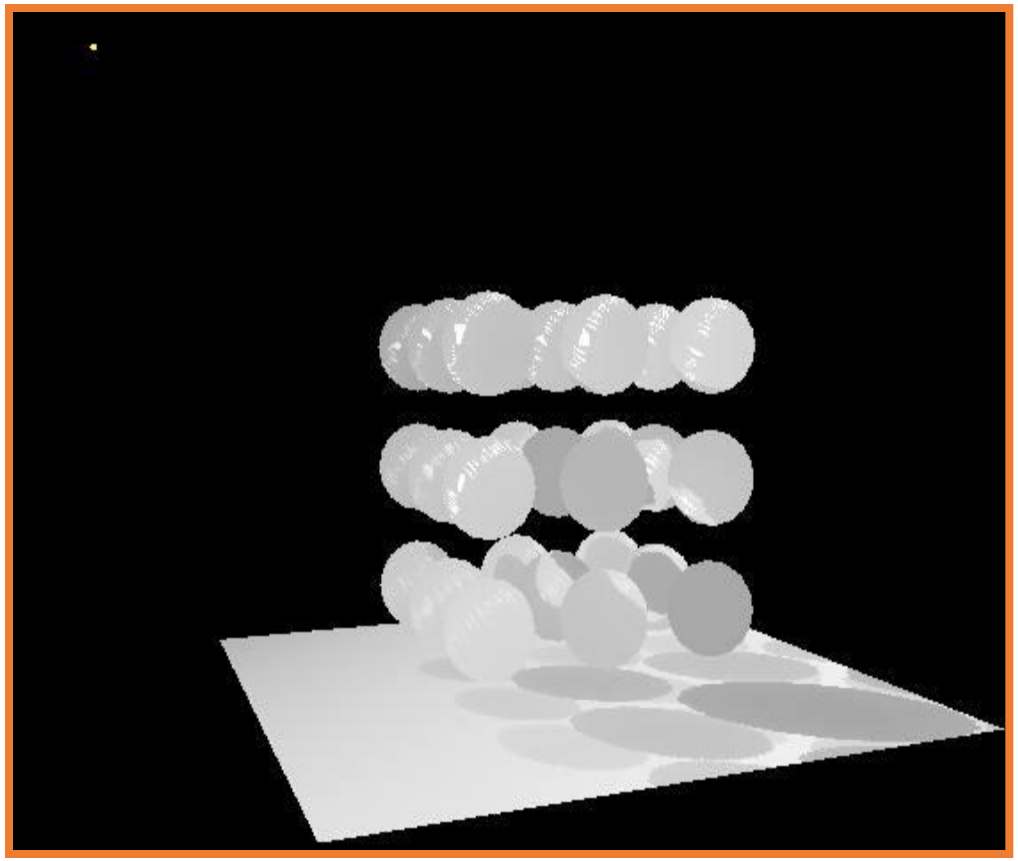


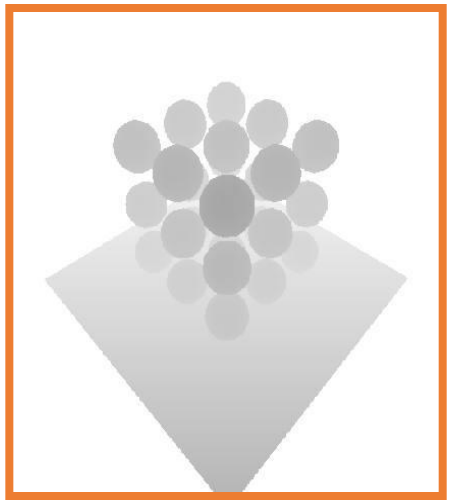


from the light's point-of-view again

Visualizing Shadow Mapping

- Projecting the depth map onto the eye's view



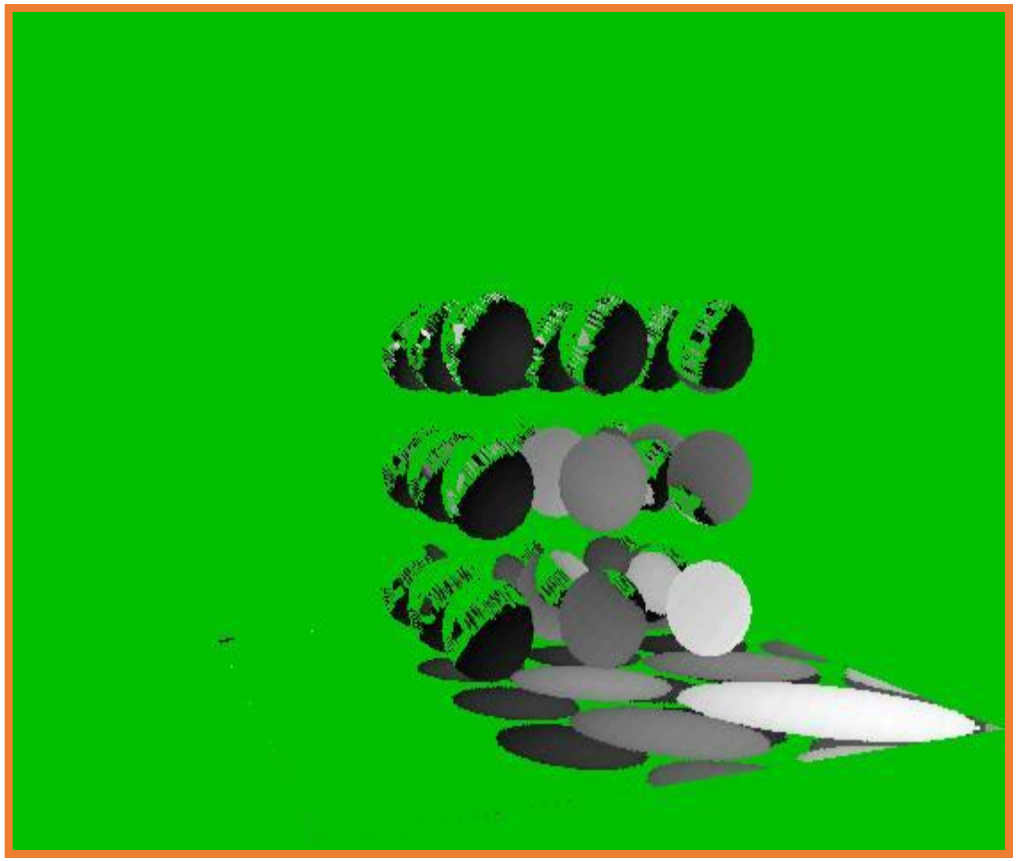


FYI: depth map for light's point-of-view again

Visualizing Shadow Mapping

- Comparing light distance to light depth map

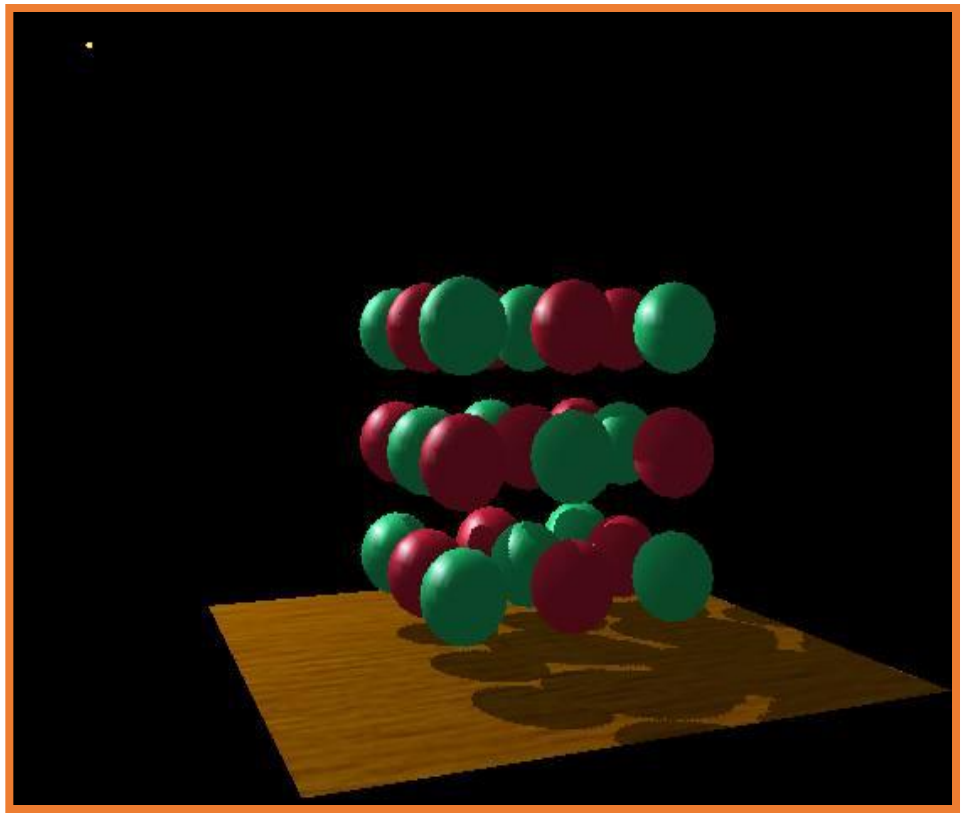
Green is where the light planar distance and the light depth map are approximately equal



Grey is where shadows should be

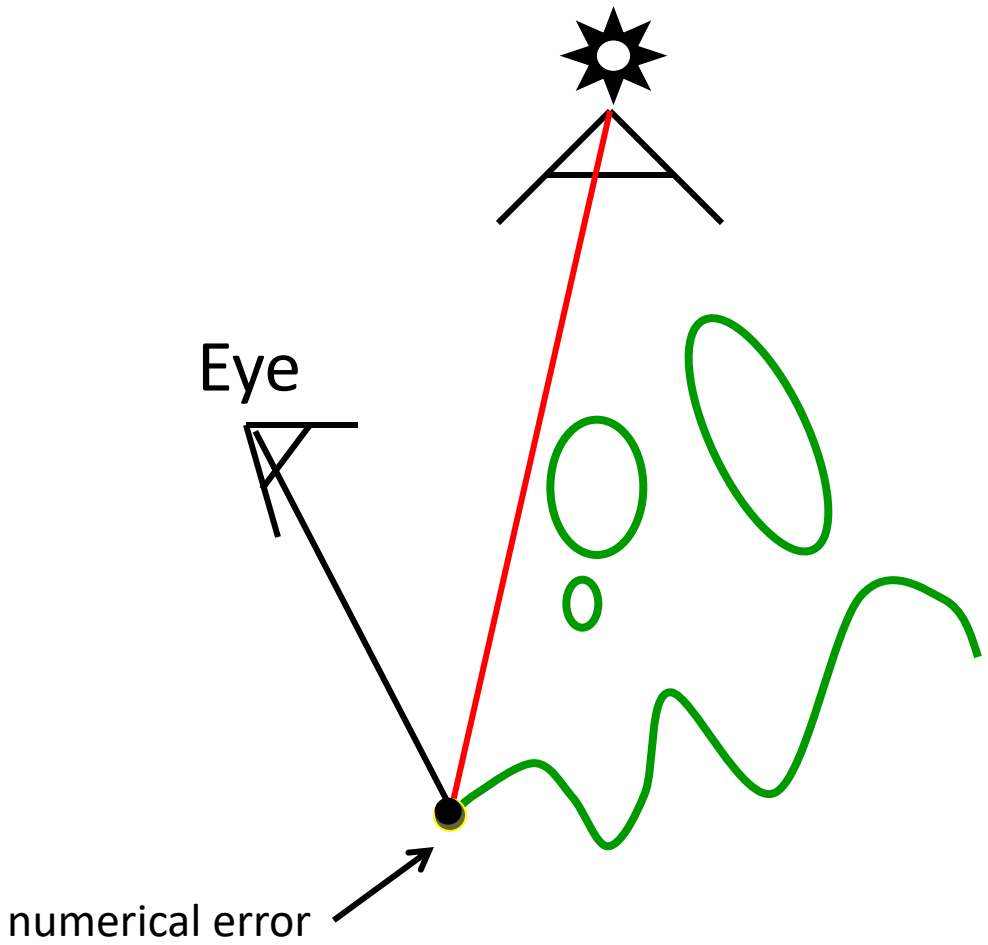
Visualizing Shadow Mapping

Notice how specular highlights never appear in shadows

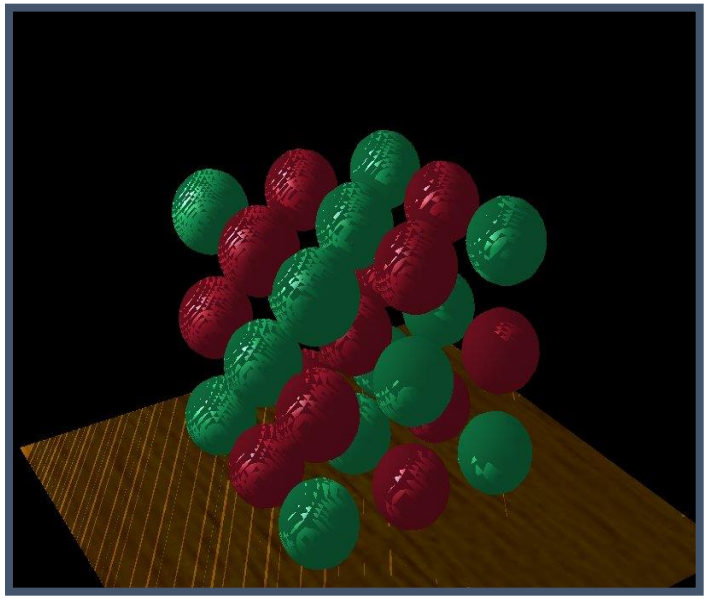


Notice how curved surfaces cast shadows on each other

Depth Map Bias

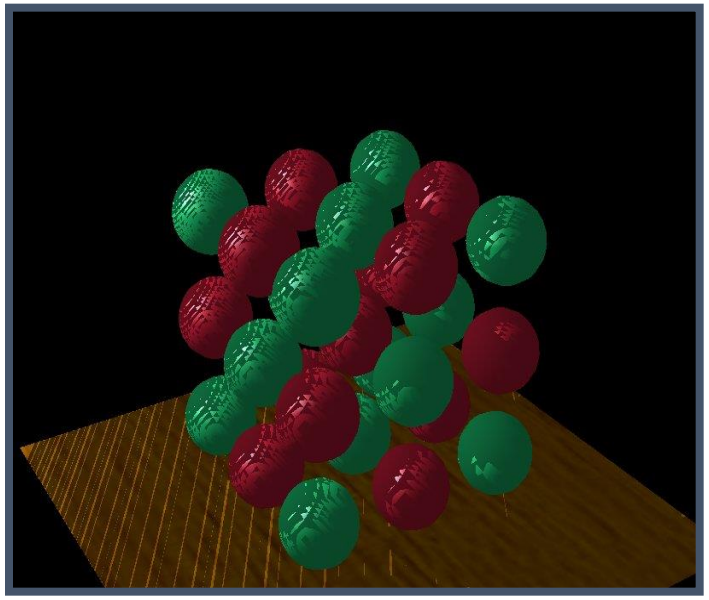


Depth Map Bias

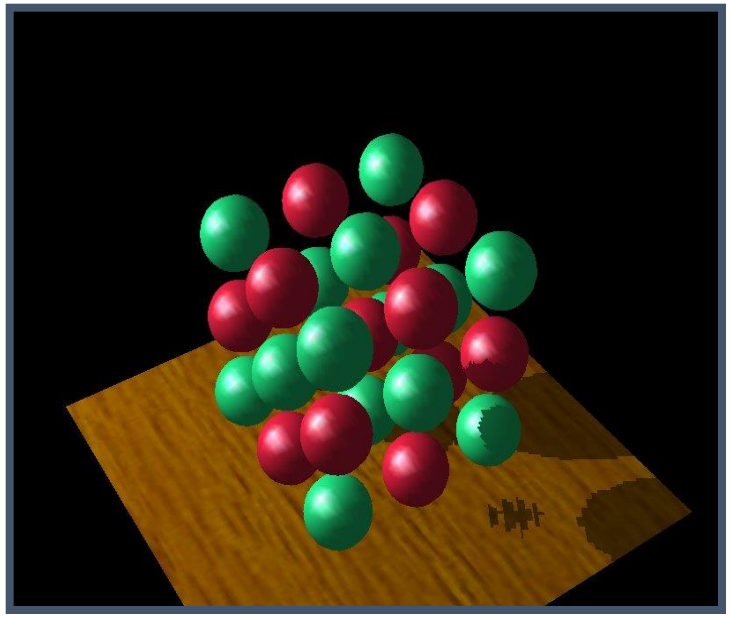


Too little bias, everything begins to shadow

Depth Map Bias

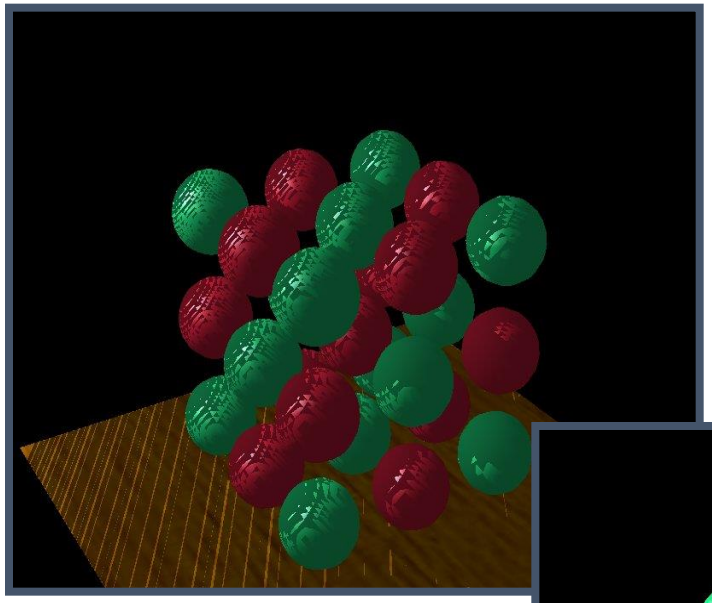


Too little bias, everything begins to shadow

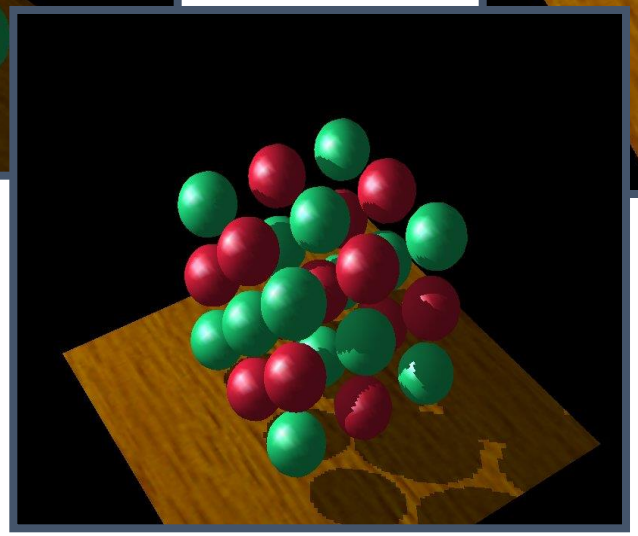


Too much bias, shadow starts too far back

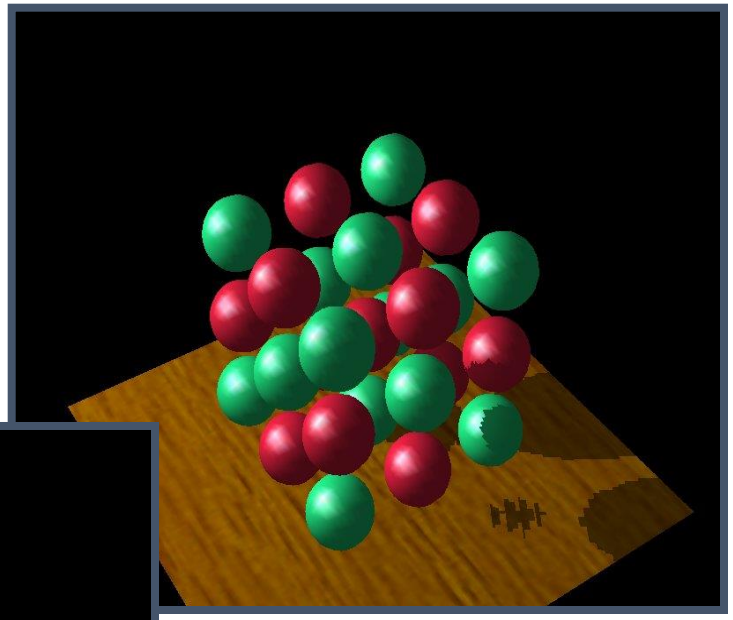
Depth Map Bias



Too little bias, everything begins to shadow



Right amount of bias



Too much bias, shadow starts too far back

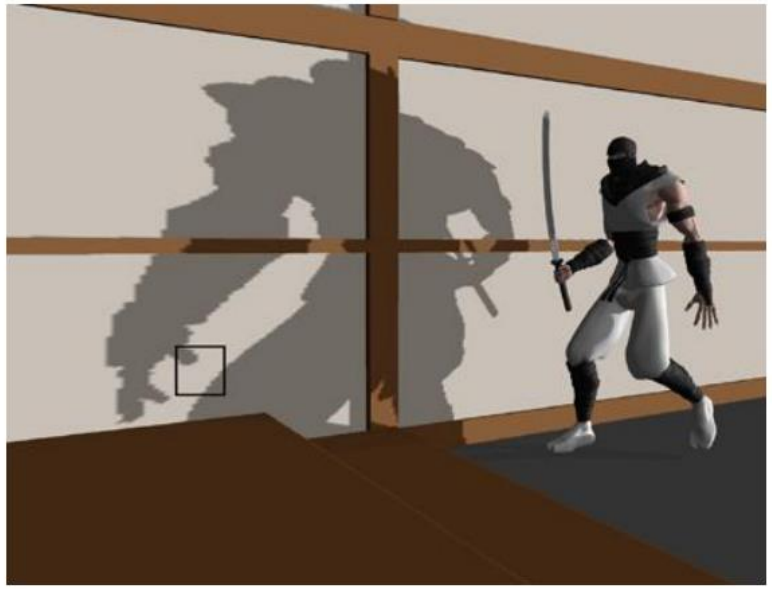
Slope Scaled Bias

```
float bias = max(0.05 * (1.0 - dot(normal, light)), 0.005);
```

Percentage closer filtering (PCF)

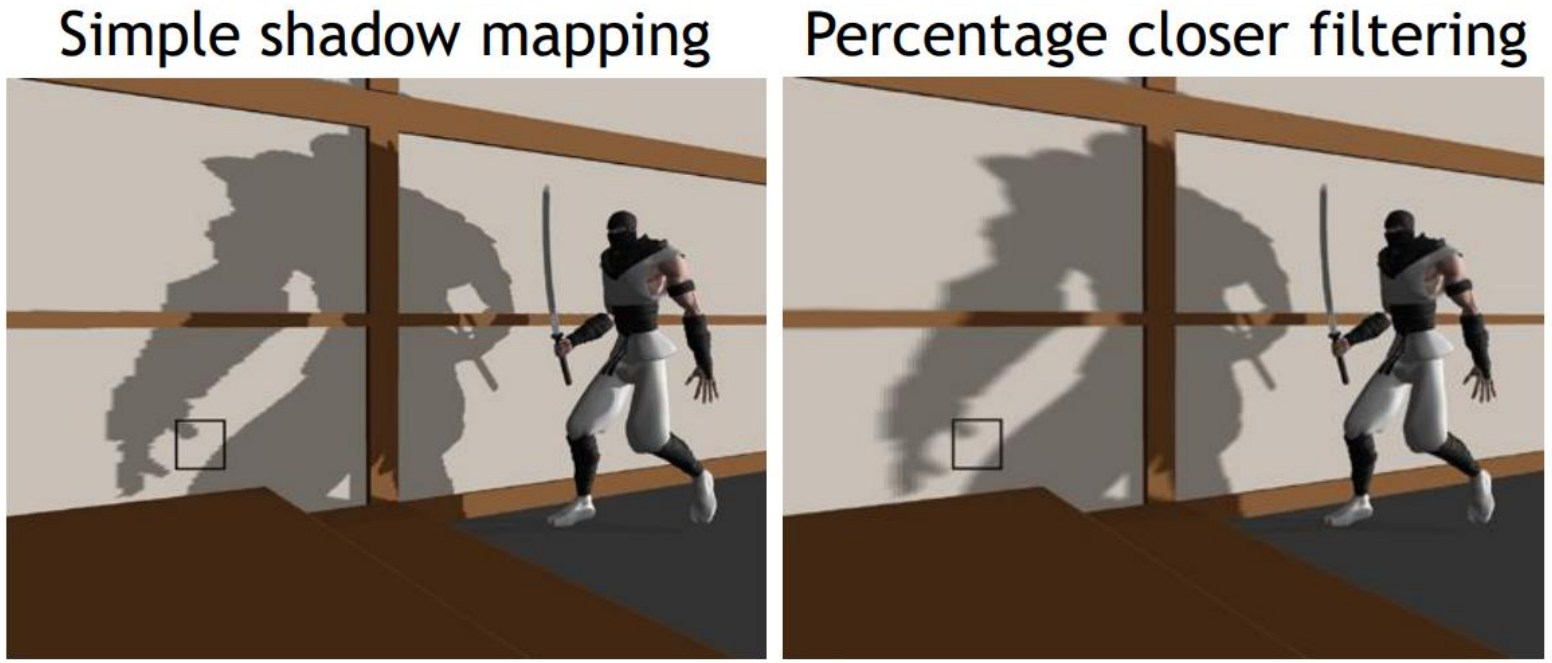
- Goal: avoid stair-stepping artifacts
- Similar to texture filtering

Simple shadow mapping



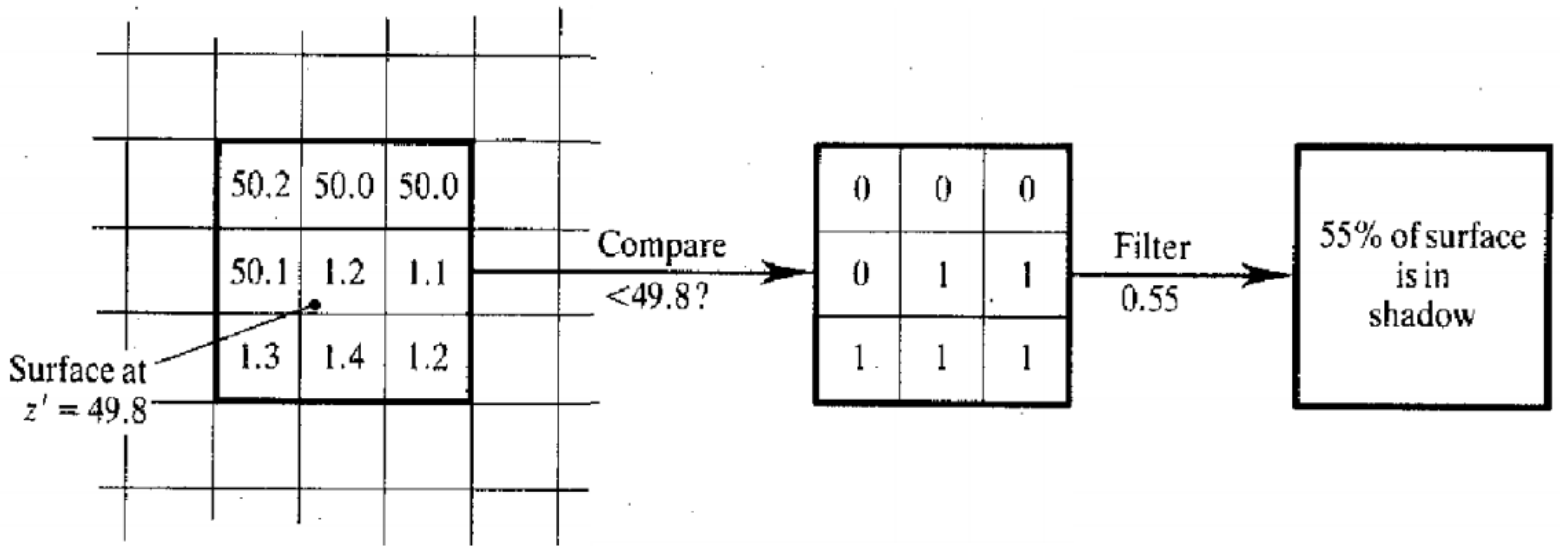
Percentage closer filtering (PCF)

- Goal: avoid stair-stepping artifacts
- Similar to texture filtering

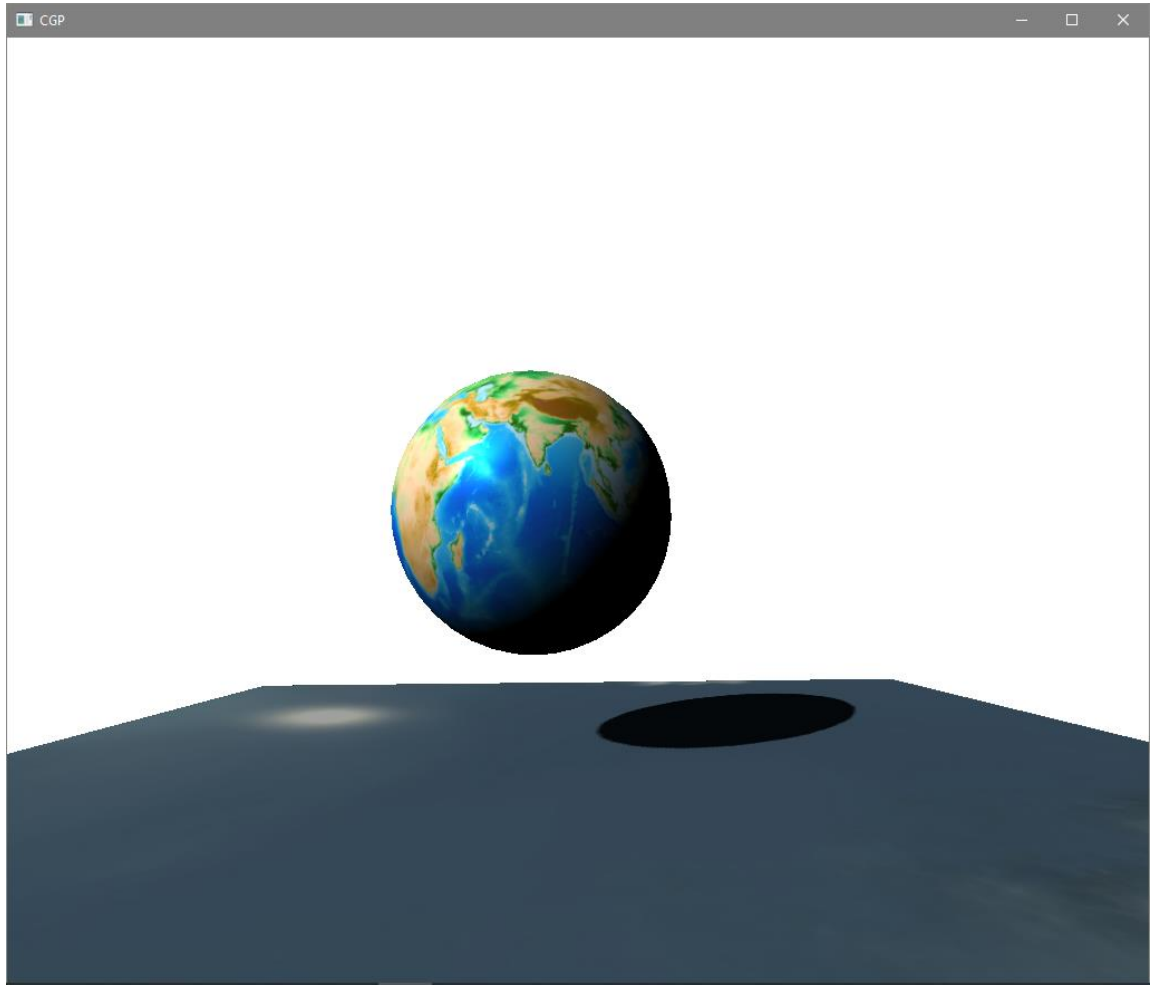


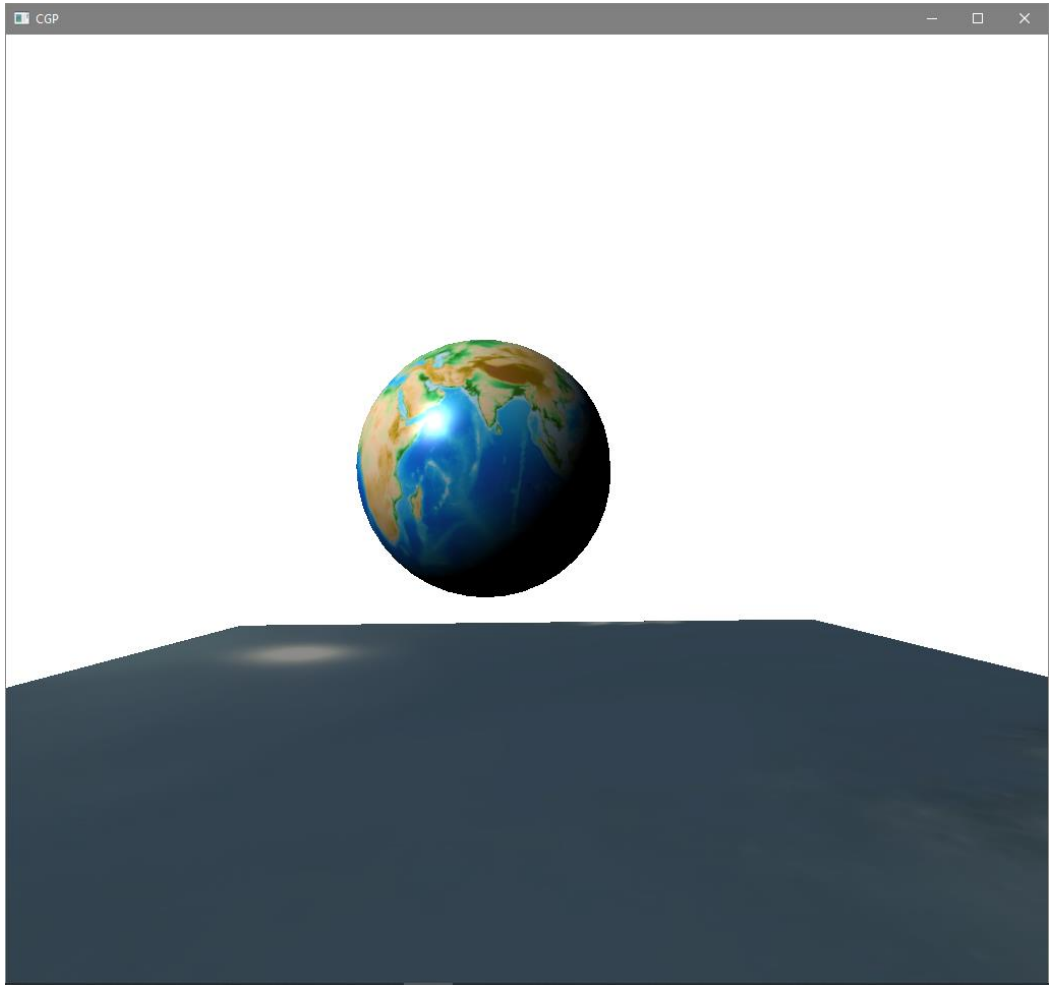
Percentage closer filtering (PCF)

- Instead of looking up one shadow map pixel, look up several
- Perform depth test for each shadow map pixel
- Compute percentage of lit shadow map pixels and use for coloring



Exercise – Shadow Mapping





Framebuffer objects

```
//Generate Framebuffer
FramebufferObject = 0;
glGenFramebuffers(1, &FramebufferObject);
glBindFramebuffer(GL_FRAMEBUFFER, FramebufferObject);

//Generate depth texture → compare to LoadTexture function!
glGenTextures(1, &depthTexture);
glBindTexture(GL_TEXTURE_2D, depthTexture);
glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, 1024, 1024, 0, GL_DEPTH_COMPONENT,
GL_FLOAT, NULL);
```

Framebuffer objects

```
//Filtering
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);

//Attach depth texture to frame buffer
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D,
depthTexture, 0);
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

Phase 1: render depth map from light's perspective

- Bind buffer → glBindFramebuffer(GL_FRAMEBUFFER, FramebufferObject); // check out setup of framebuffer in init function
- Create orthogonal projection matrix → glm::ortho<float>(-20, 20, -20, 20, -20, 30);
- Create Camera matrix with inverse light direction → glm::lookAt(-lightDir, glm::vec3(0, 0, 0), glm::vec3(0, 1, 0));
- Concatenate above matrices with model matrix of object and send to GPU → create new draw method that uses depth shaders only
- Unbind framebuffer after finishing drawing ALL objects to depth texture → glBindFramebuffer(GL_FRAMEBUFFER, 0);

Phase 1: Vertex Shader

```
layout(location = 0) in vec3 vertexPosition;

uniform mat4 lightMVP;

void main()
{
	gl_Position =  lightMVP * vec4(vertexPosition, 1.0);
}
```

Phase 1: Fragment Shader

```
#version 430 core


void main()
{

}
```

Phase 2: draw objects with shadows

- **Clear the buffers!** → glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
- Send light MVP matrix to GPU
- Send both the object texture as well as its depth map to the GPU (you can use SetActiveTexture for both textures but the depth map texture should have the final parameter index of 1)
- Draw objects using shadow mapping shaders (create a separate draw routine in your code)

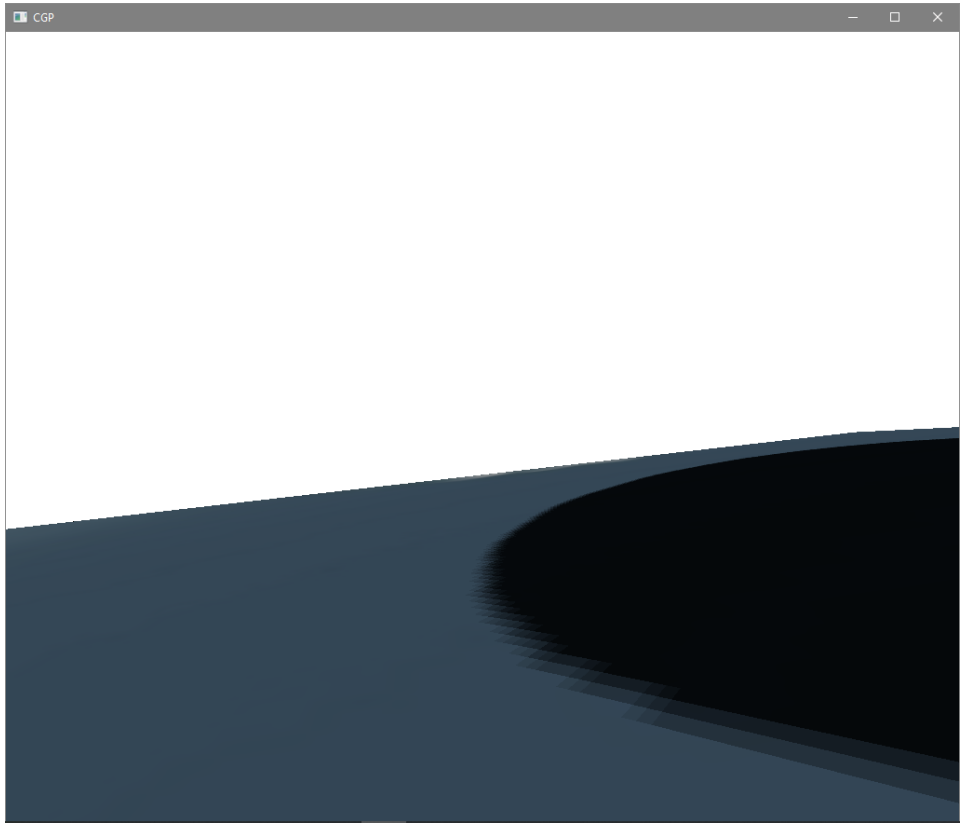
Phase 2: Vertex Shader

- Create new shader file
  - Create light MVP matrix uniform
  - Rest as in “shader_tex.vert” but transform **world space** vertex positions also to light space and send them to fragment shader

Phase 2: Fragment Shader

- Create new shader file
  - As texture shader but add an extra variable of type sampler2D for the depth map
  - Create a shadow calulcation function that takes as input the transformed world space position passed from the vertex shader and returns a floating point number between 0 and 1 → this is the shadow value
    - Apply perspective division on light space positions (divide .xyz of vector by .w)
    - Project position values from [-1, 1] → [0, 1]
    - Sample depth value from depth map using projected position values
    - Compare to z value of projected positions – bias and return either 0 or 1
  - Mix color and shadow values to determine final color in ratios of e.g. 15% and 85% respectively

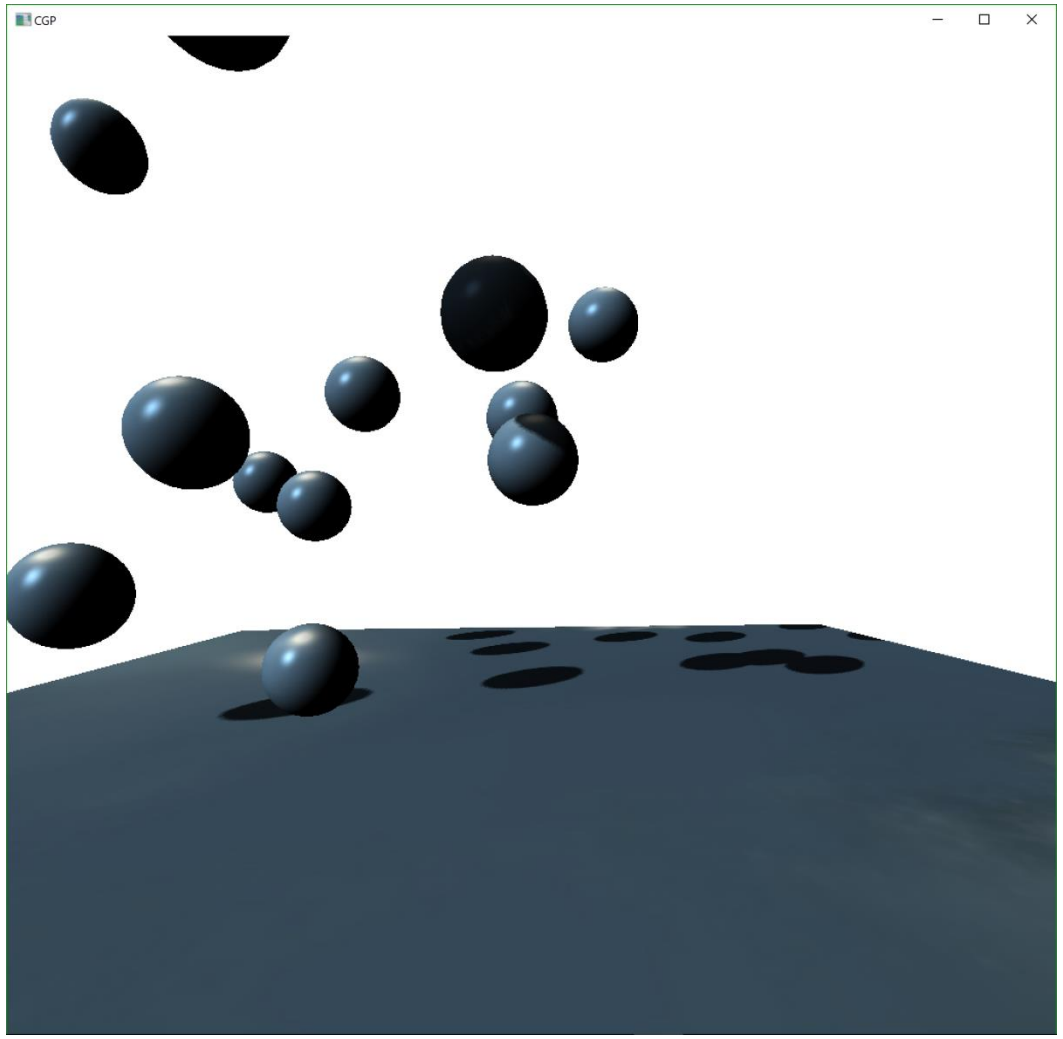
Add PCF (calculate weighted average shadow value using neighboring texel depth values)



PCF

standard shadow mapping

Create a fancier scene with many objects



Exercise – Dynamic Environment Map

- 1. set viewport to texture size,
- 2. clear buffers
- 3. orient camera to represent view from current cube side
- 4. render scene without the environment-mapped object
- 5. loop 2,3 and 4 for all six cube sides
- 6. restore previous camera parameters and viewport