JOHANNES KEPLER UNIVERSITY LINZ

INSTITUTE OF COMPUTER GRAPHICS

Computer Graphics

Lab 6: Advanced Texture Mapping

Fork me on GitHub

Dev Environment: Lab Package

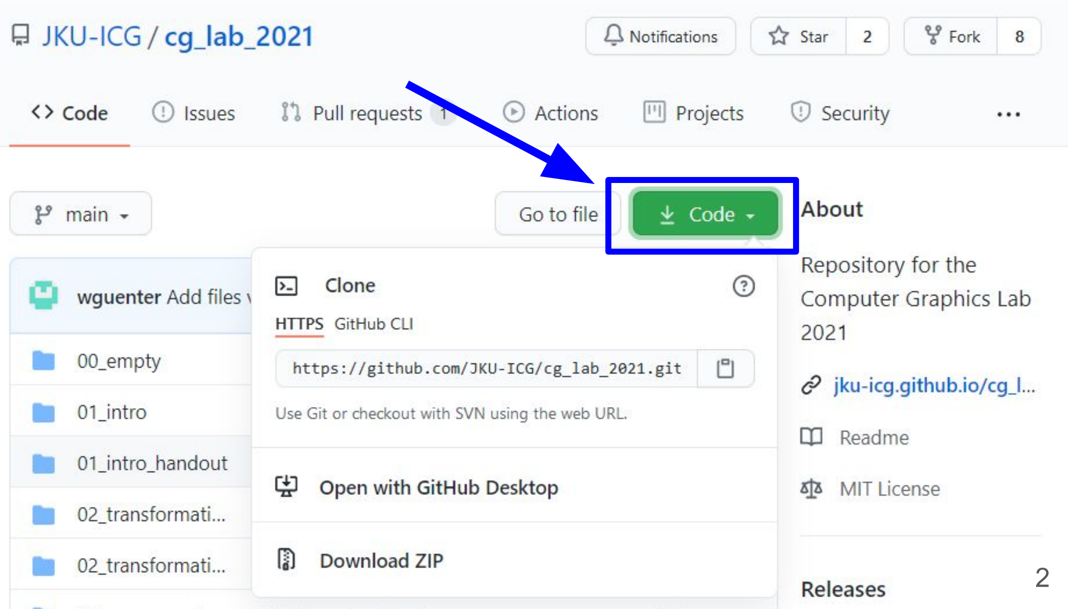
JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

Hosted on GitHub: https://github.com/jku-icg/cg_lab_2021

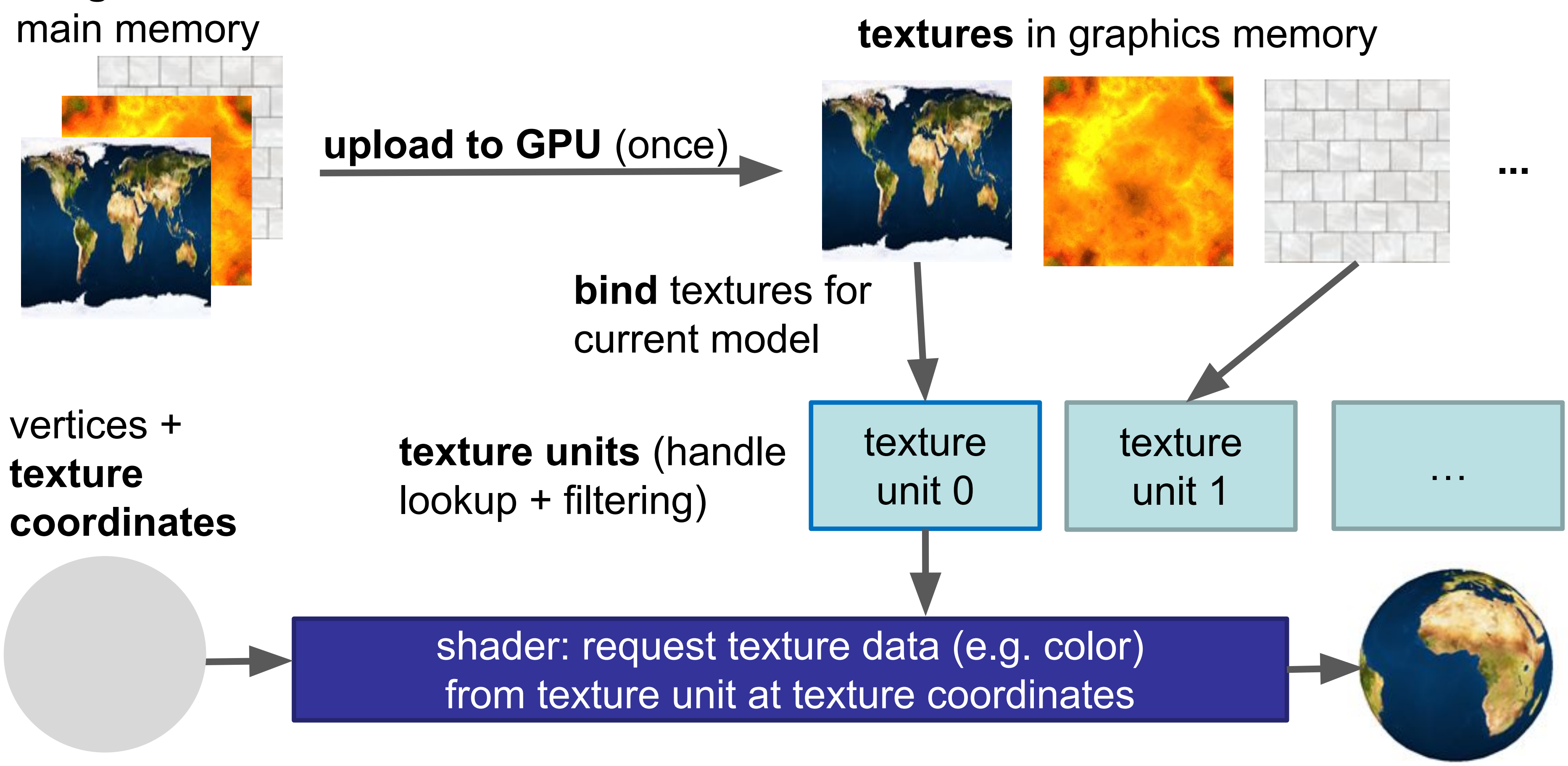
The repository will be updated during the lab with the new projects.

To get started (now):

1. Download the ZIP
2. Extract the folder
3. Open Visual Studio Code
4. Open `cg_lab_2021` folder (*File → Open*)
5. Click on **Go Live** button in lower right corner



Recap: Texturing



Agenda for Today

JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

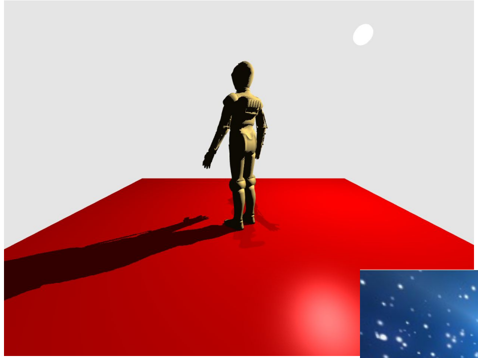
Shadow Mapping

- Overview
- Recap: Render to Texture
- Task 1: Setup Camera for Light
- Depth Comparison
- Eye-to-Light Matrix
- Task 2: Shadow Mapping
- Extra Task: Smooth Shadows

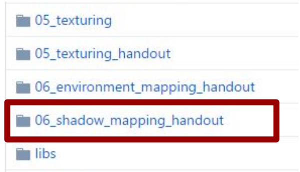
Environment Mapping

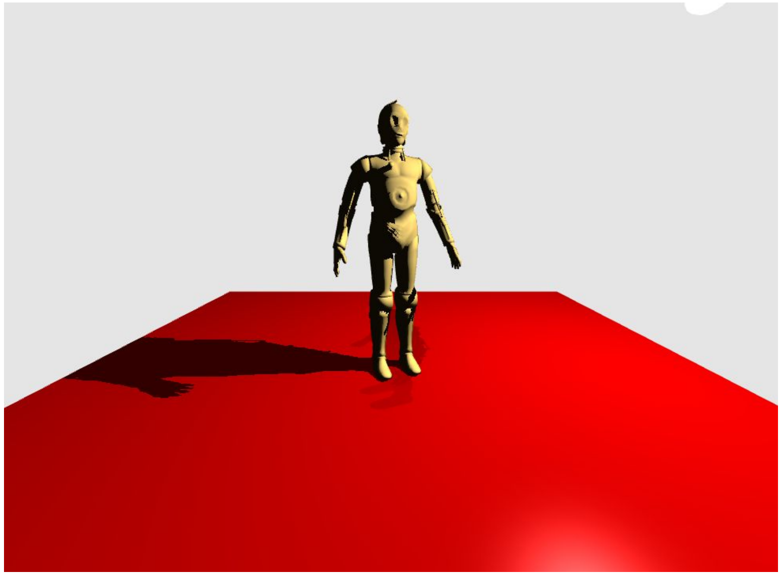
- Cube Mapping
- Differences to 2D Textures
- Task 3: Cube Mapping

Texture Filtering



Shadow Mapping





Shadow Mapping

Generate shadows using depth textures

Pure image based technique!

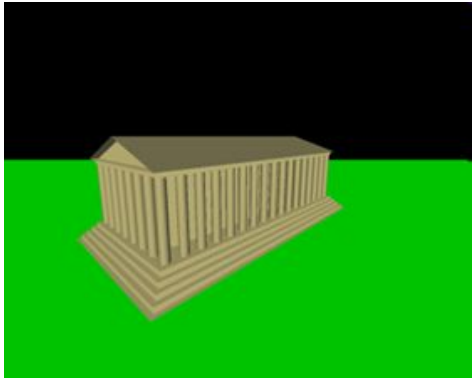
2 Render Passes:

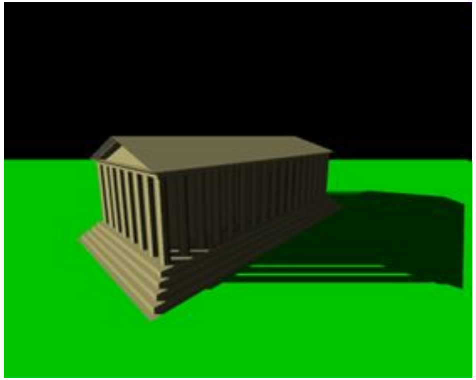
1. Render scene from perspective of light source into texture (we need the depth map!)

2. Render scene from perspective of camera:

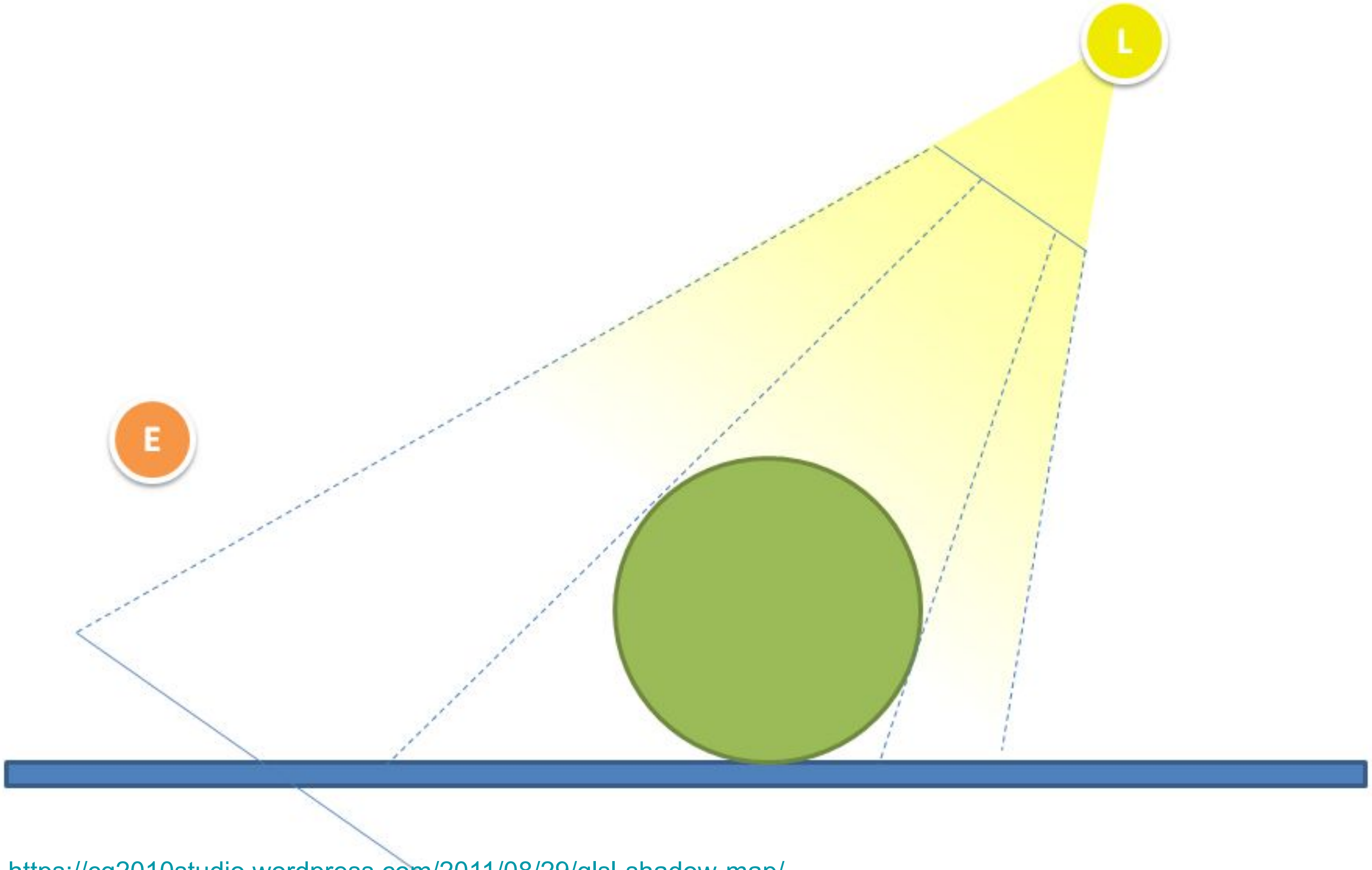
    For each fragment check if distance to light source is larger than stored depth in texture

    If distance is larger: fragment is behind an object → it is in the shadow!





Shadow Mapping Example

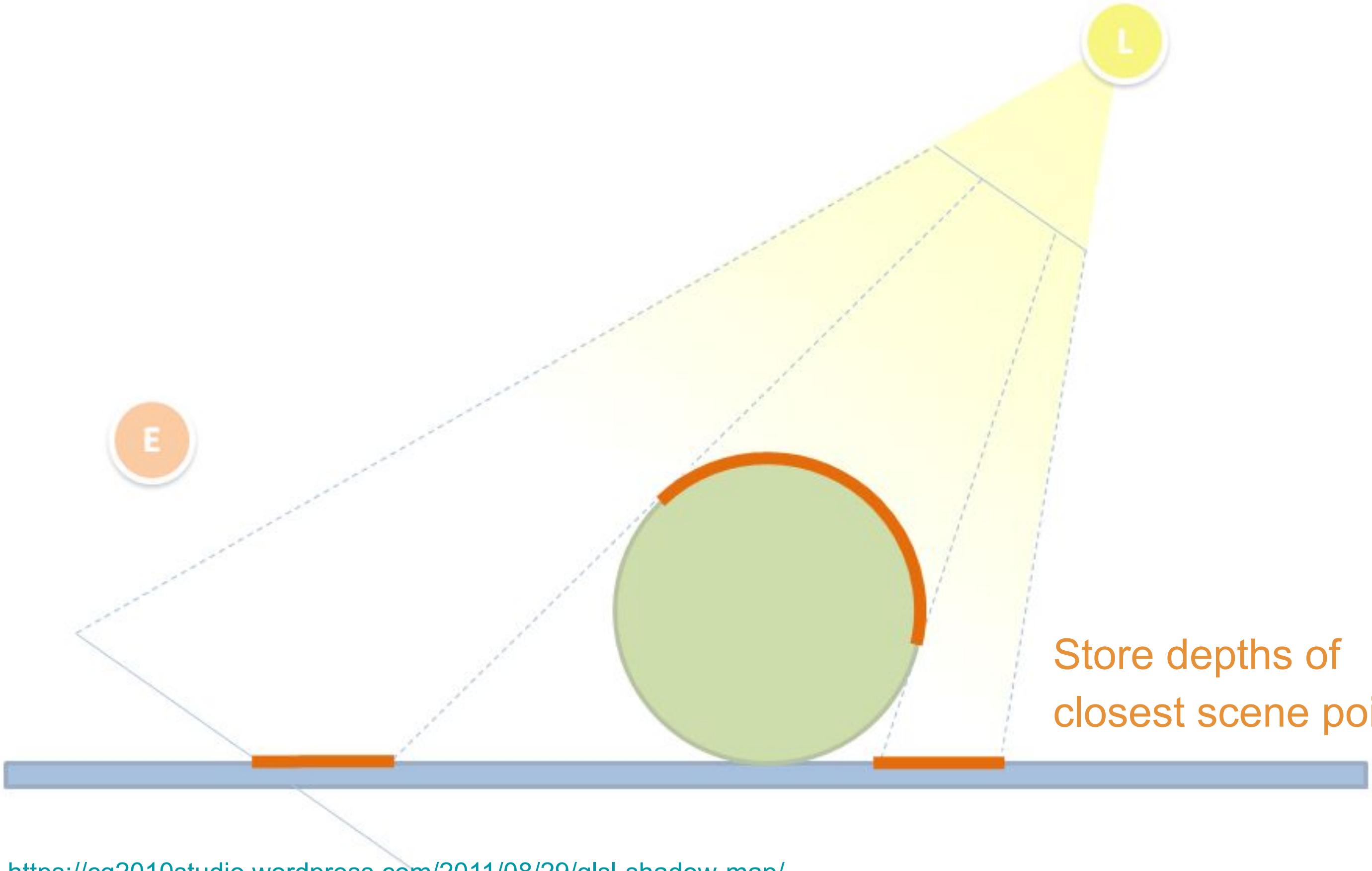


L = Light source
E = Eye / Camera

https://cg2010studio.wordpress.com/2011/08/29/glsl-shadow-map/

Render Scene From Light Perspective

JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

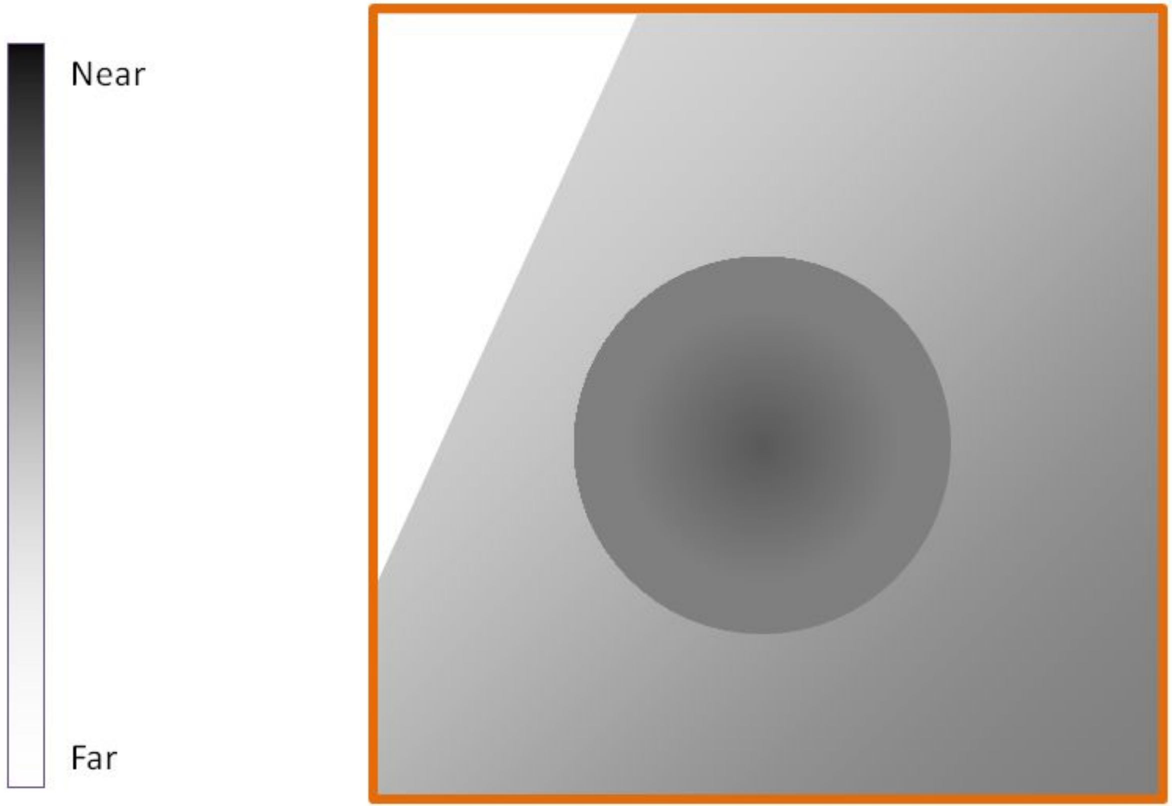


L = Light source
E = Eye / Camera

Store depths of closest scene points

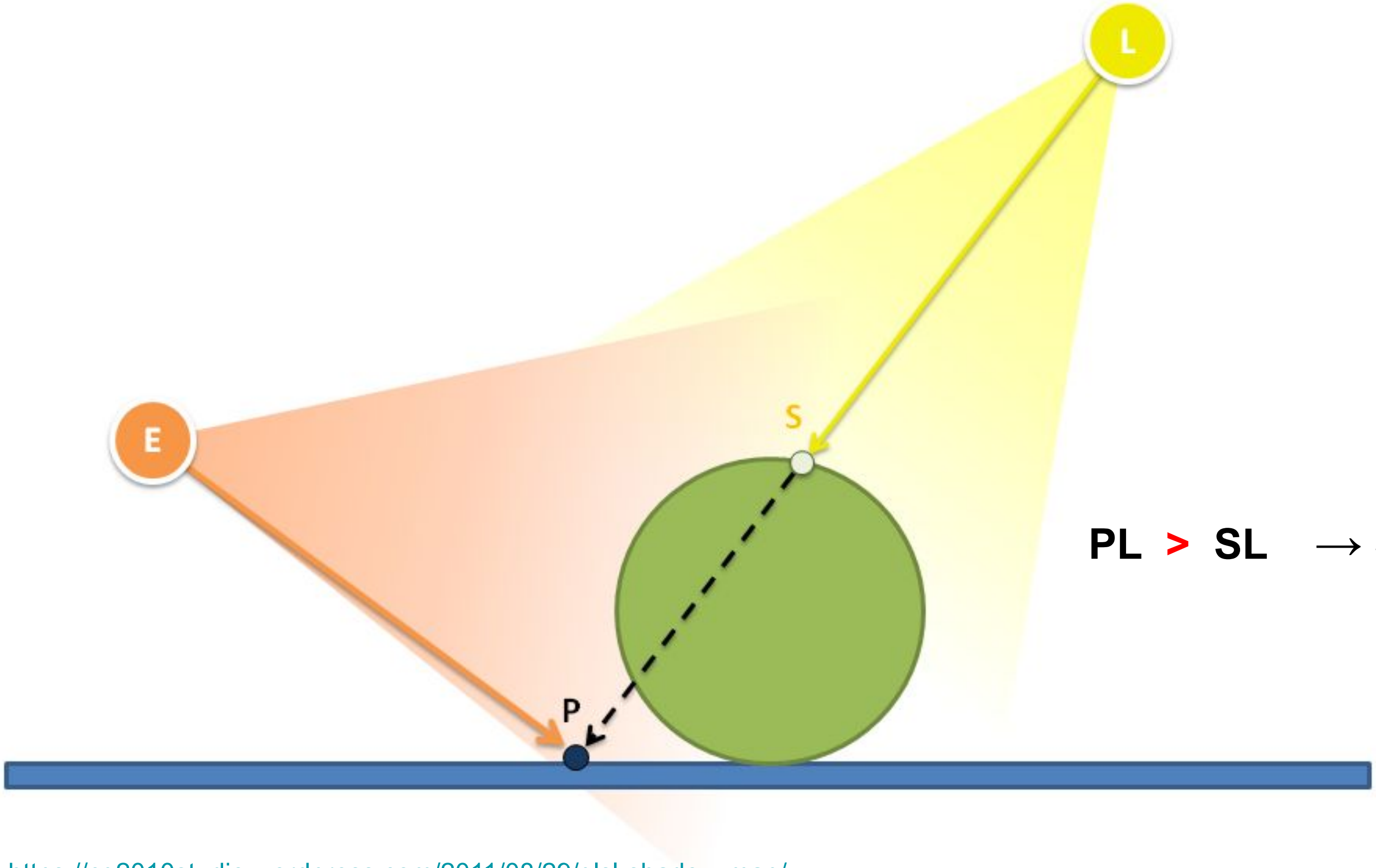
https://cg2010studio.wordpress.com/2011/08/29/glsl-shadow-map/

Resulting Depth Map



https://cg2010studio.wordpress.com/2011/08/29/glsl-shadow-map/

Render Scene from Camera Perspective

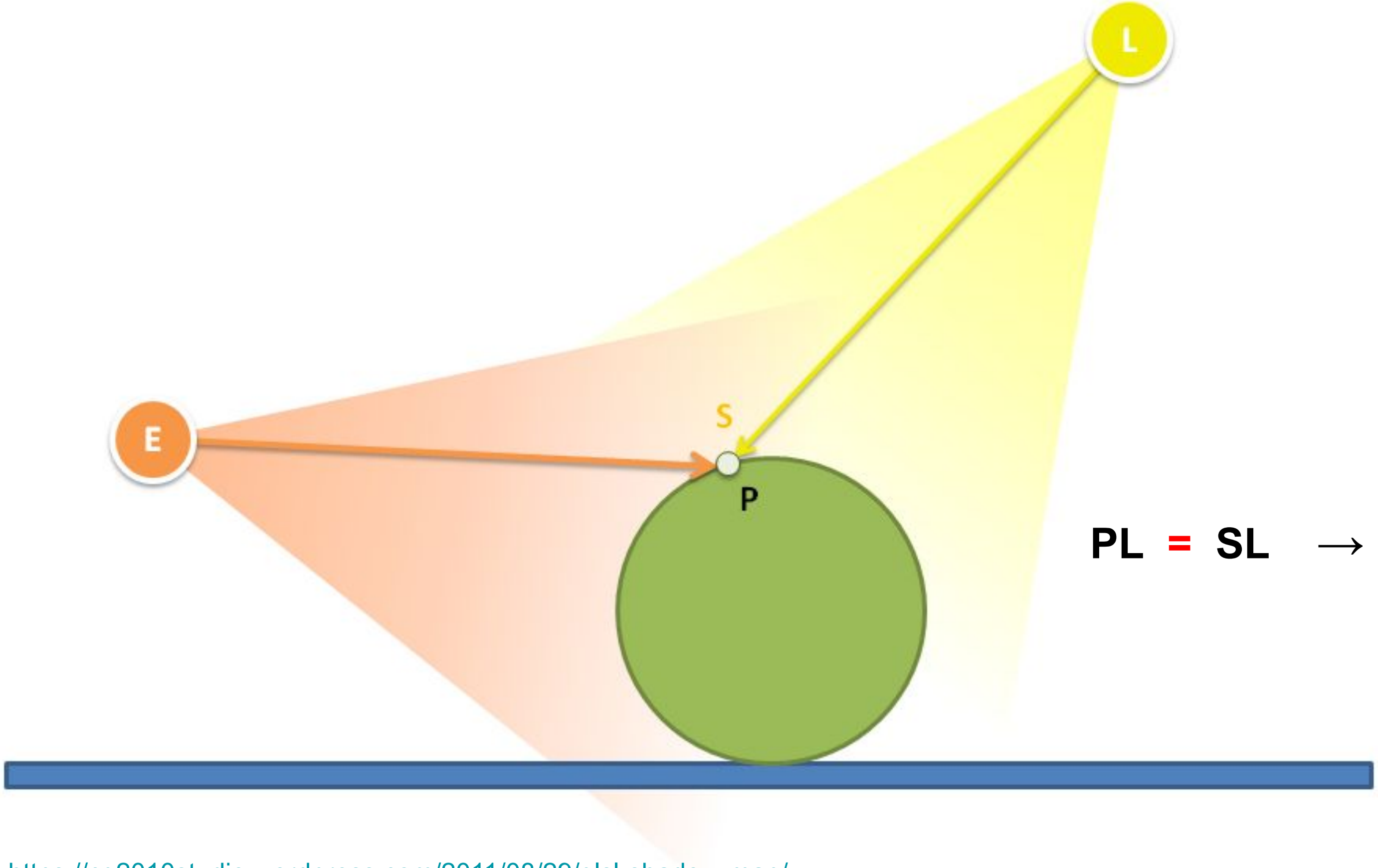


L = Light source
E = Eye / Camera
P = Rendered Point
S = Closest Point

PL **>** **SL** → **Shadow**

https://cg2010studio.wordpress.com/2011/08/29/glsl-shadow-map/

Render Scene from Camera Perspective



L = Light source
E = Eye / Camera
P = Rendered Point
S = Closest Point

PL = SL → No Shadow

https://cg2010studio.wordpress.com/2011/08/29/glsl-shadow-map/

Recap: Render to Texture

JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

Render into framebuffer:

1. Enable framebuffer:

```
gl.bindFramebuffer(gl.FRAMEBUFFER, renderTargetFramebuffer);
```

2. Setup viewport + camera + clear buffers + render scene graph
3. Disable framebuffer:

```
gl.bindFramebuffer(gl.FRAMEBUFFER, null);
```

Nothing will be shown on screen!

Framebuffer has attached textures to render into:

```
gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0, gl.TEXTURE_2D, renderTargetColorTexture, 0);
gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.DEPTH_ATTACHMENT, gl.TEXTURE_2D, renderTargetDepthTexture ,0);
```

See initRenderToTexture for complete initialization!

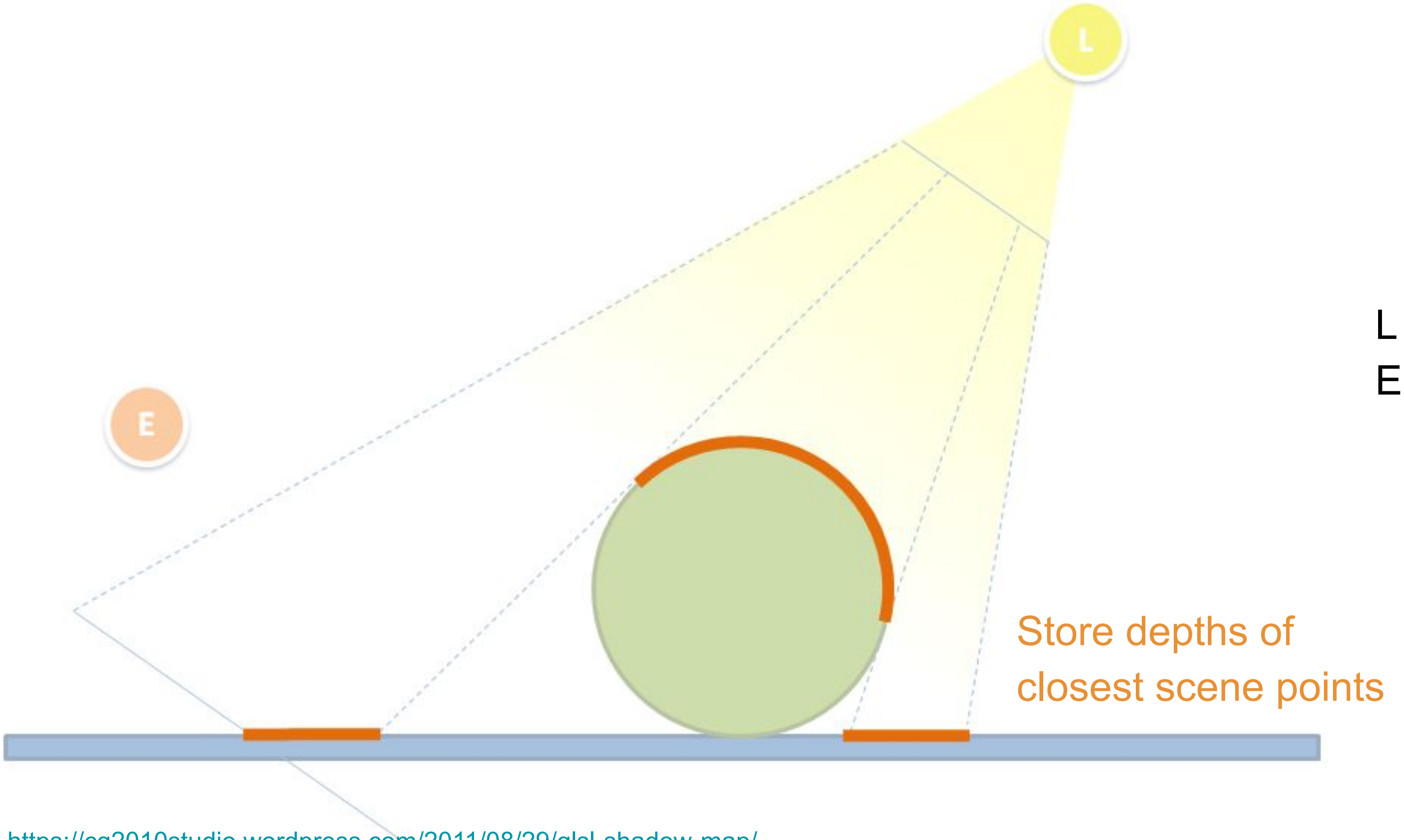
Recap: Render to Texture

JOHANNES KEPLER
UNIVERSITY LINZ
INSTITUTE OF
COMPUTER GRAPHICS

```
function renderToTexture(timeInMilliseconds)
{
  //bind framebuffer to draw scene into texture
  gl.bindFramebuffer(gl.FRAMEBUFFER, renderTargetFramebuffer);

  //setup viewport
  gl.viewport(0, 0, framebufferWidth, framebufferHeight);
  gl.clearColor(0.9, 0.9, 0.9, 1.0);
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

  //setup context and camera matrices
  const context = createSGContext(gl);
  context.projectionMatrix = mat4.perspective(mat4.create(), 30, framebufferWidth / framebufferHeight, 0.01, 100);
  context.viewMatrix = mat4.lookAt(mat4.create(), [0,-1,-4], [0,0,0], [0,1,0]);

  //render scenegraph
  rootnofloor.render(context);

  //disable framebuffer (to render to screen again)
  gl.bindFramebuffer(gl.FRAMEBUFFER, null);
}
```

Render Scene From Light Perspective

JKU
JOHANNES KEPLER
UNIVERSITY LINZ
INSTITUTE OF
COMPUTER GRAPHICS



L = Light source
E = Eye / Camera

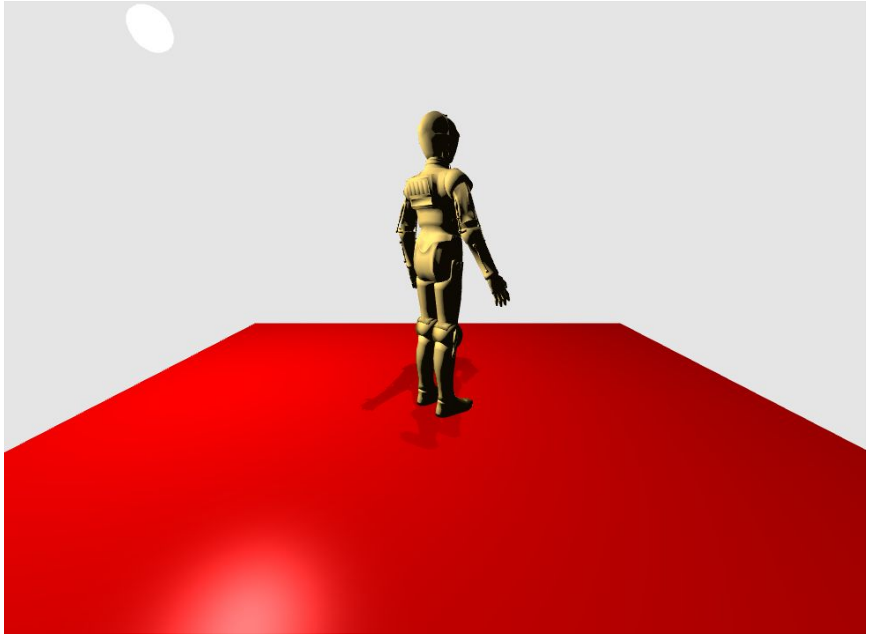
https://cg2010studio.wordpress.com/2011/08/29/glsl-shadow-map/

Task 1: Setup Light View Matrix

JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

Goal:

Render scene to texture from light's perspective. Resulting depth map can be seen in the texture on the floor.



Tasks:

1.1 Adapt viewMatrix in renderToTexture function according to the light position.

Hint: Use provided light position in world space and the mat4.lookAt function.

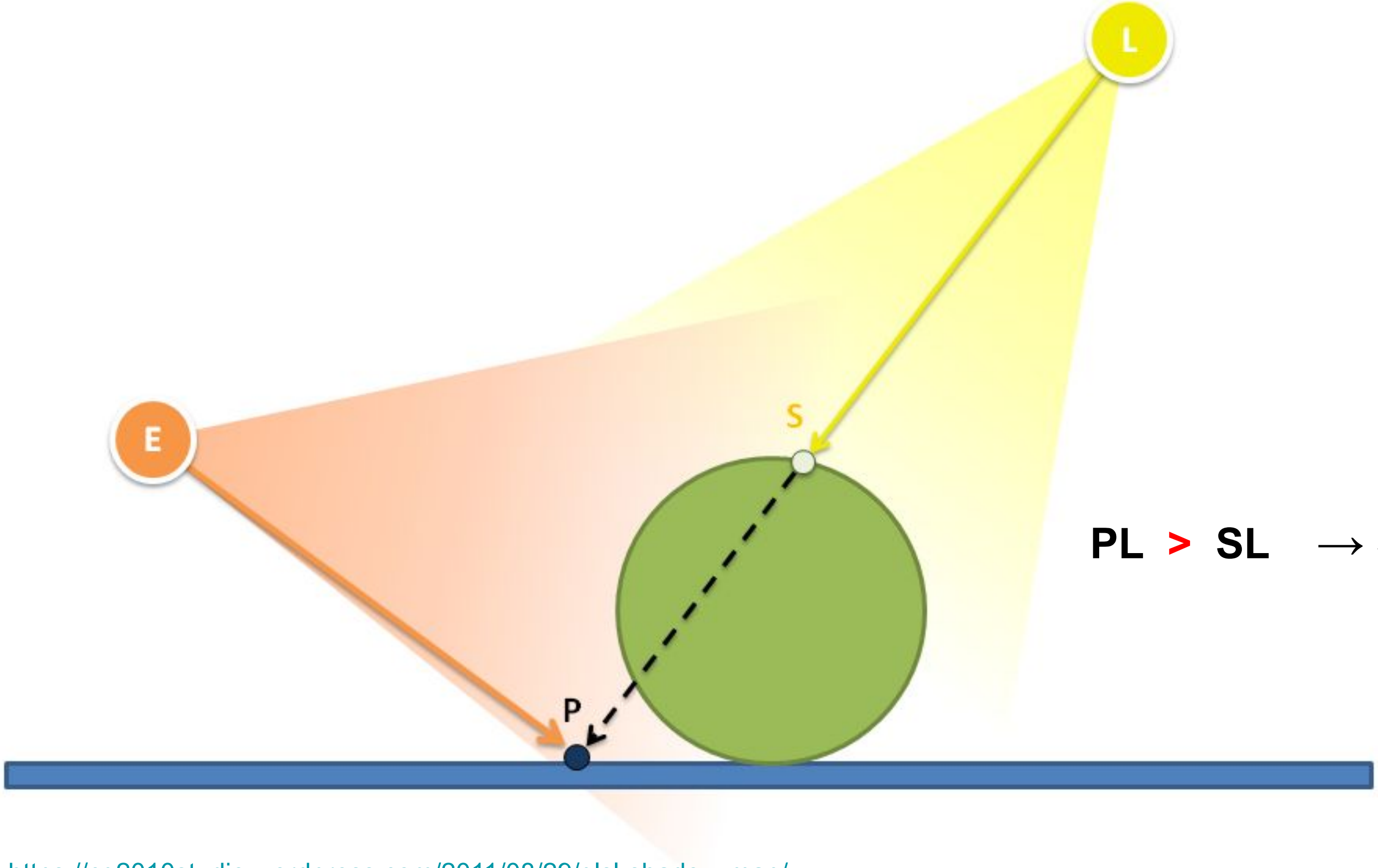
Source code in 06_shadow_mapping folder!

Solution: Setup Light View Matrix

JOHANNES KEPLER
UNIVERSITY LINZ
INSTITUTE OF
COMPUTER GRAPHICS

```
//TASK 1.1: setup camera to look at the scene from the light's perspective
let lookAtMatrix = mat4.lookAt(mat4.create(), worldLightPos, worldLightLookAtPos, upVector);
```

Render Scene from Camera Perspective

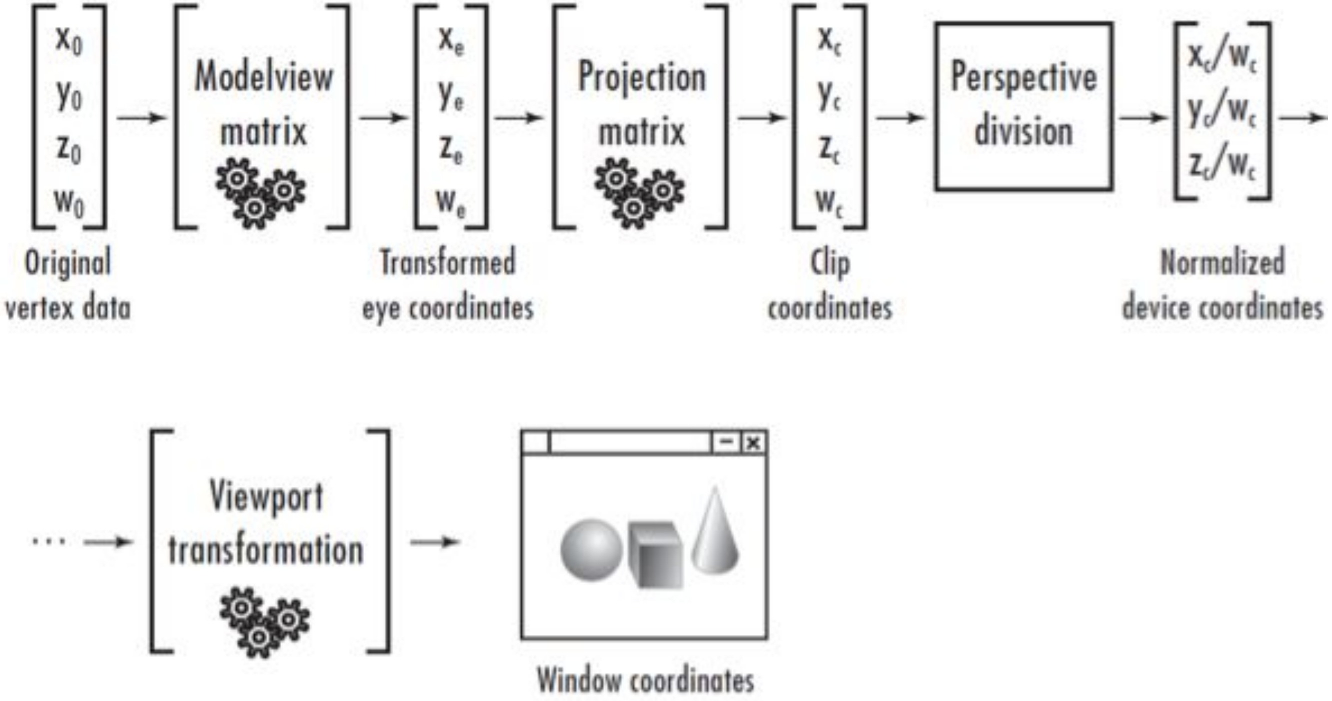


L = Light source
E = Eye / Camera
P = Rendered Point
S = Closest Point

PL > SL → Shadow

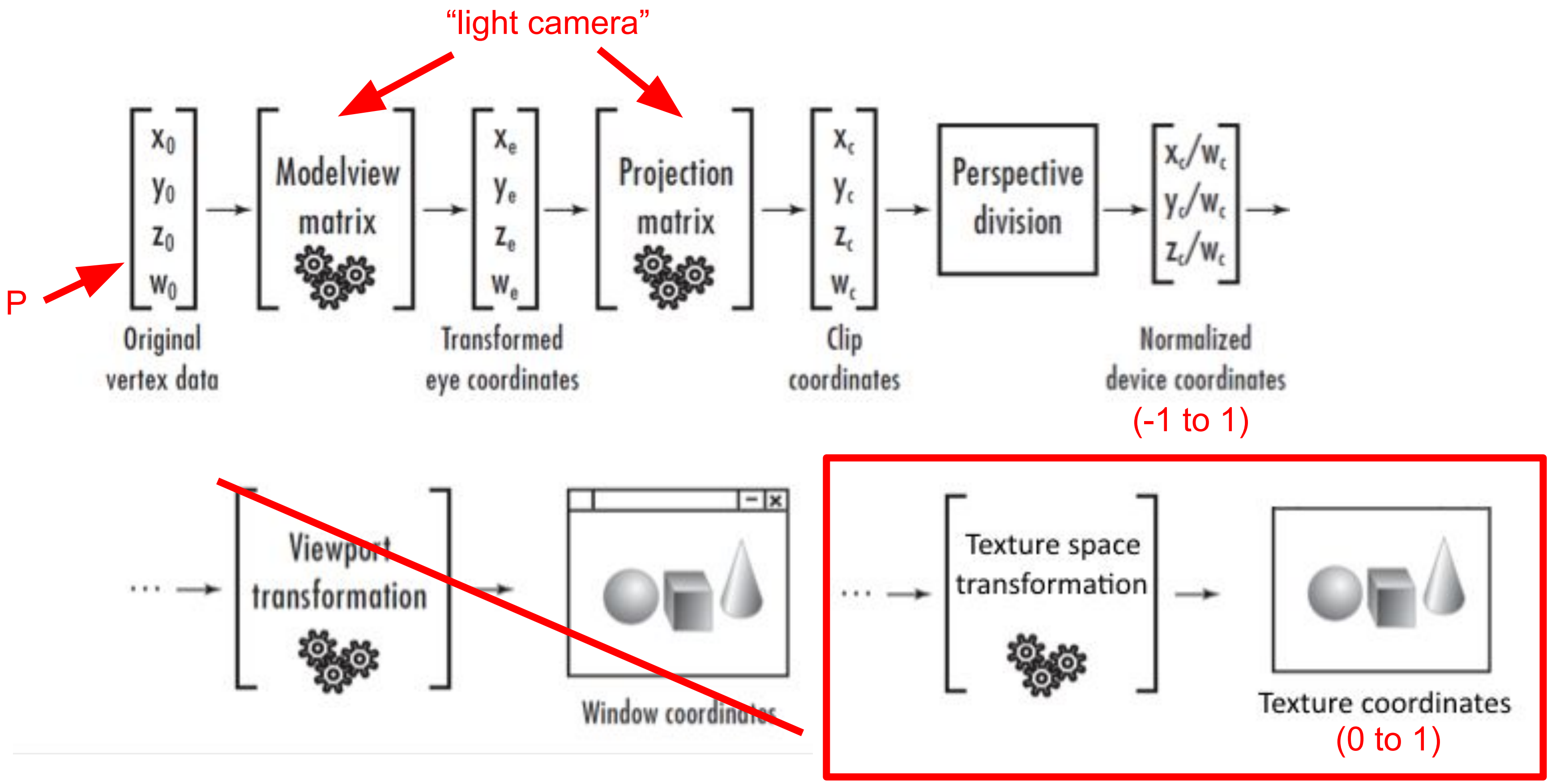
https://cg2010studio.wordpress.com/2011/08/29/glsl-shadow-map/

Recap: Transformation Pipeline

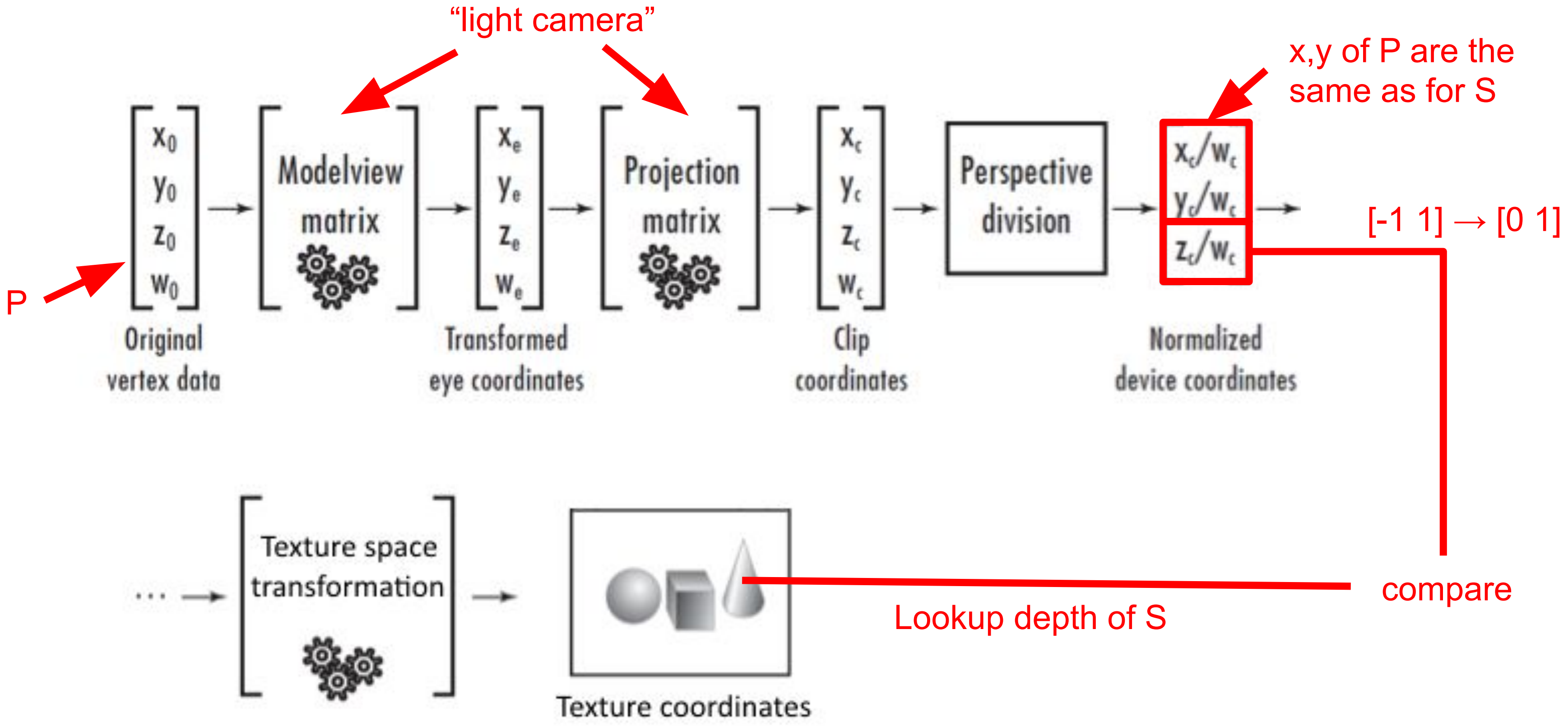


Depth Map Lookup For Each P

JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

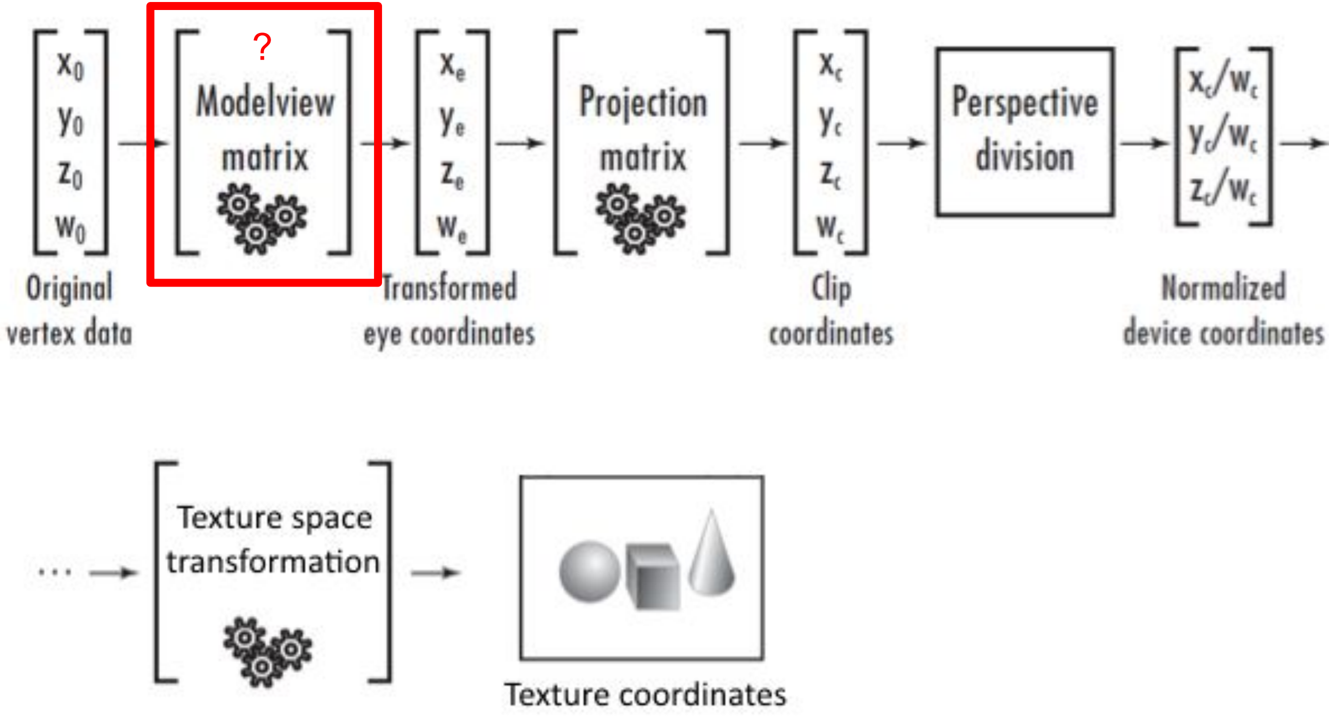


Depth Comparison



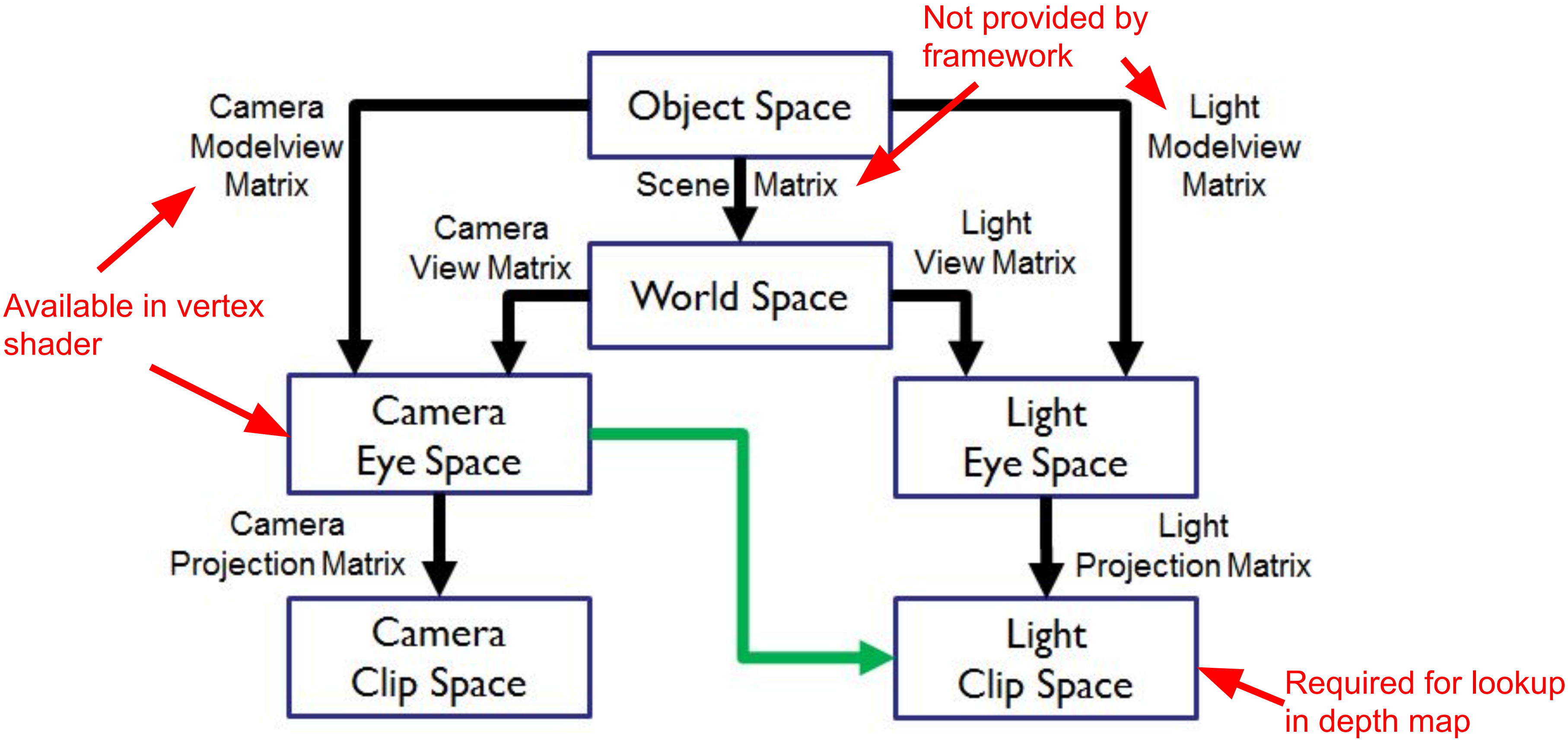
Render Scene from Camera Perspective

Unknown for “light camera” when rendering with “real camera” in our framework

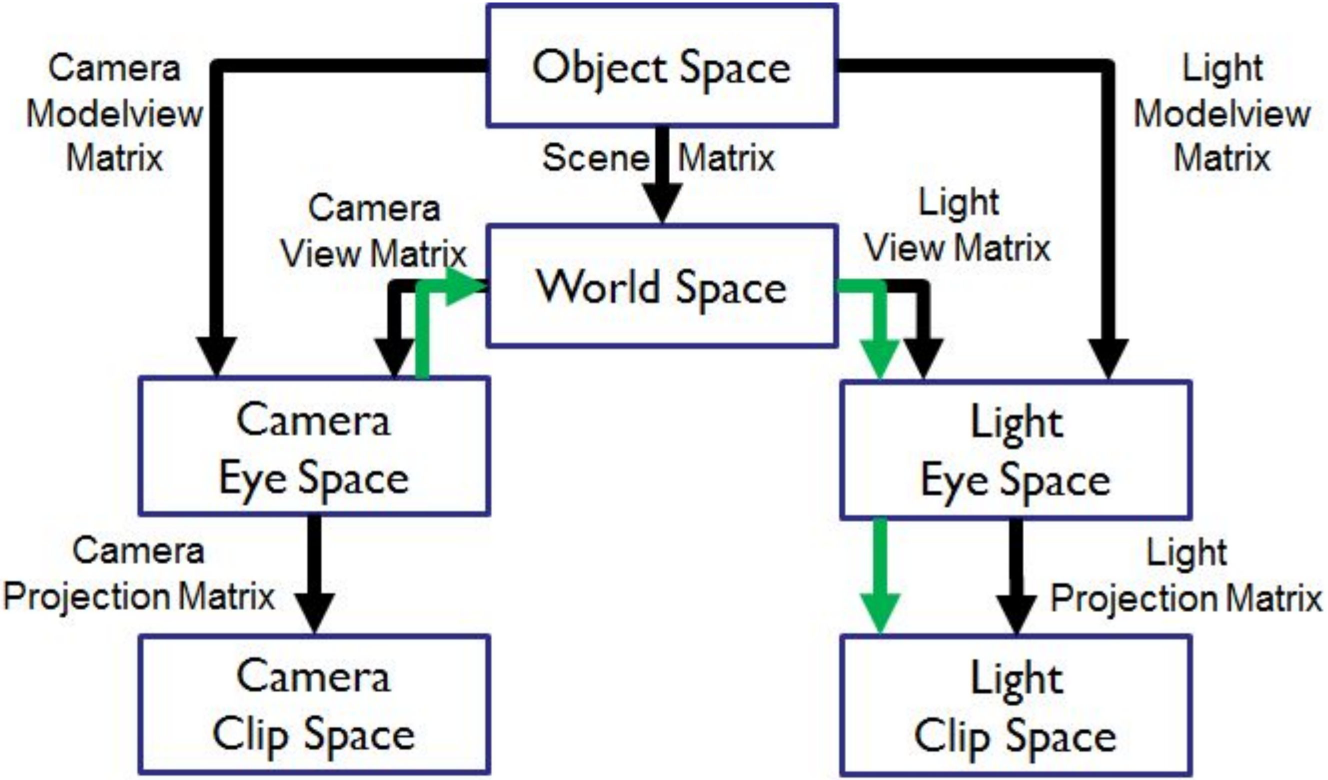


Eye to Light Matrix

JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS



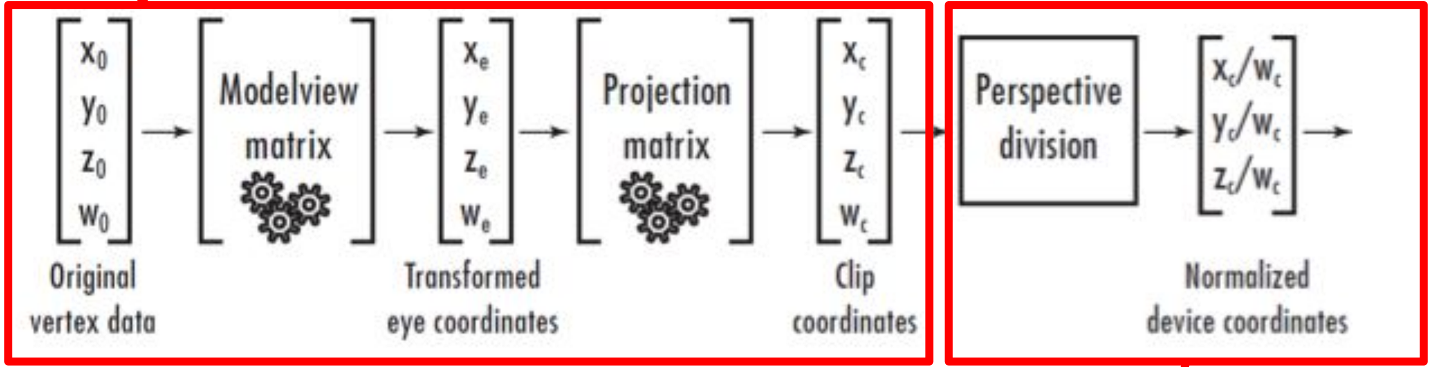
Eye to Light Matrix

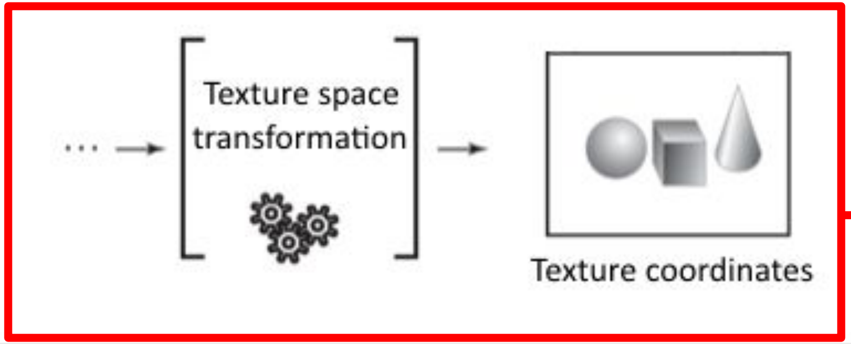


Eye-to-light matrix is constant for all rendered models

Compute Texture Coordinates

Do in vertex shader (using eye-to-light matrix + model-view matrix of “real camera”)





Do in fragment shader
(otherwise wrong interpolation because of perspective division)

Main Render Steps

Main render function:

1. Update light animation first!
2. Render scene from light's perspective into texture
   - generate shadow map
   - save light view and projection matrices (required for computing eye-to-light matrix)
3. Setup camera matrices
4. Compute inverted camera view matrix (required for computing eye-to-light matrix)
5. Render scene graph

ShadowSGNode:

1. Bind depth texture to a texture unit
2. Assign sampler in shader to depth texture unit
3. Compute eye-to-light matrix and pass to shader

Task 2: Shadow Mapping

Goal:

C3PO should cast a shadow on the floor

Tasks:

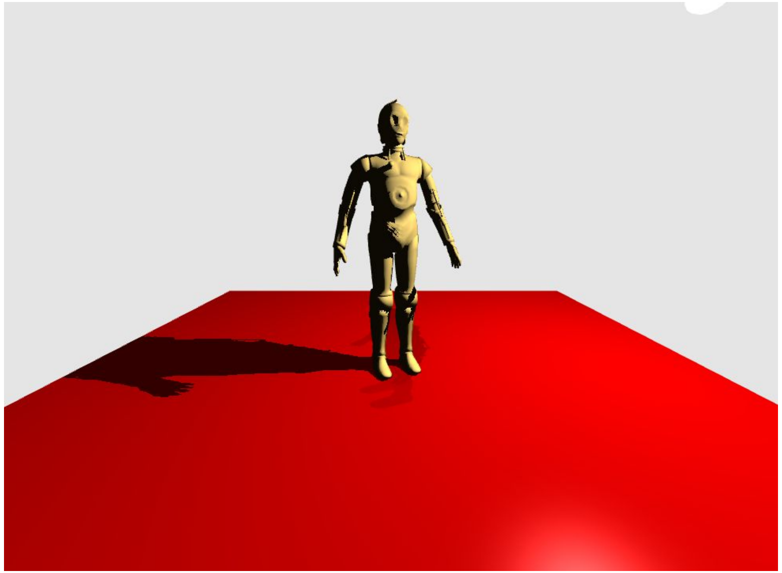
2.1 Compute eye-to-light matrix in ShadowSGNode

2.2 Compute light clip space coordinates (in shadow.vs.glsl) using eye-to-light matrix

2.3 Apply perspective division to light clip space coordinates (in shadow.fs.glsl)

2.4 Lookup depth in texture and compute shadow coefficient

2.5 Apply shadow coefficient to diffuse and specular part of phong computation



Solution: Shadow Mapping

Task 2.1 - ShadowSGNode:

```
var eyeToLightMatrix = mat4.multiply(mat4.create(),this.lightViewProjectionMatrix,context.invViewMatrix);
gl.uniformMatrix4fv(gl.getUniformLocation(context.shader, 'u_eyeToLightMatrix'), false, eyeToLightMatrix);
```

Task 2.2 - Vertex Shader (`shadow.vs.glsl`):

```
v_shadowMapTexCoord = u_eyeToLightMatrix*eyePosition;
```

Solution: Shadow Mapping

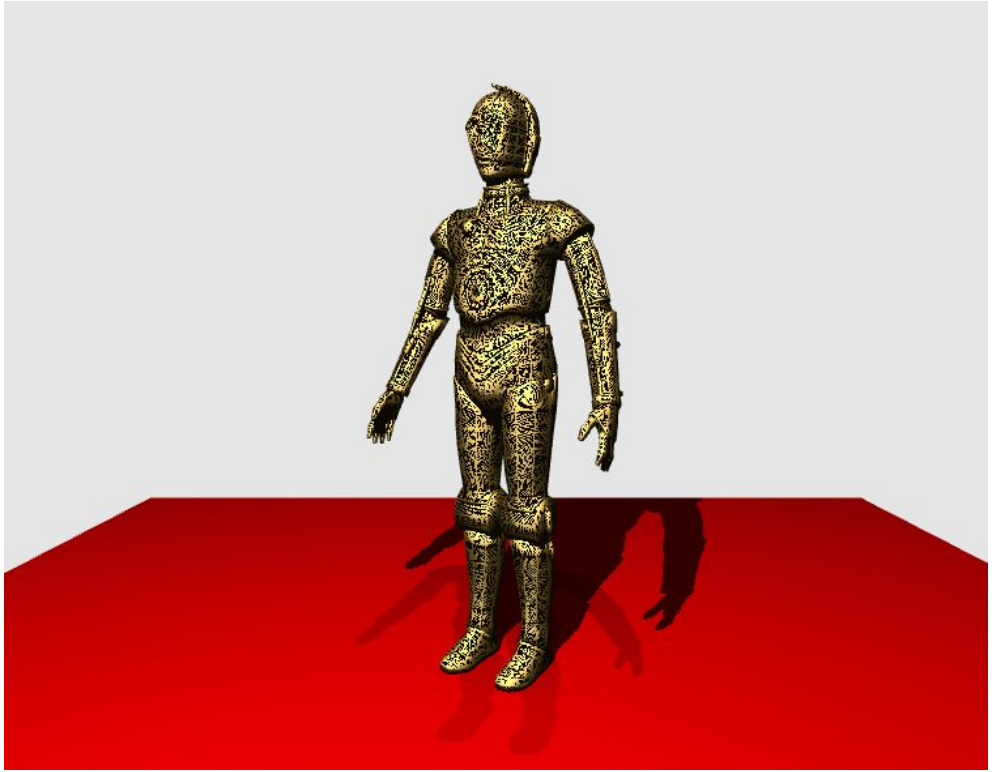
JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

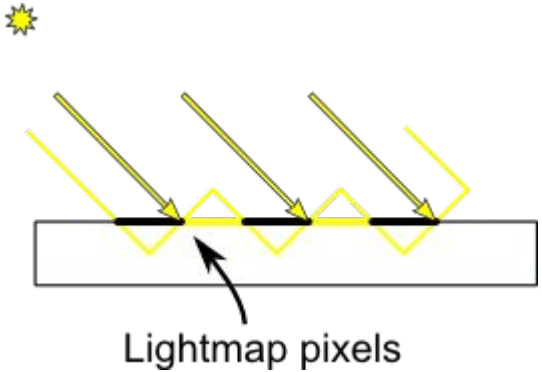
Task 2.3,2.4,2.5 - Fragment Shader (`shadow.fs.glsl`):

```
//TASK 2.3: apply perspective division to v_shadowMapTexCoord
vec3 shadowMapTexCoord3D = v_shadowMapTexCoord.xyz/v_shadowMapTexCoord.w; //do perspective division

//do texture space transformation (-1 to 1 -> 0 to 1)
shadowMapTexCoord3D = vec3(0.5,0.5,0.5) + shadowMapTexCoord3D*0.5;
//substract small amount from z to get rid of self shadowing (EXTRA TASK: disable to see difference)
shadowMapTexCoord3D.z -= 0.003;

float shadowCoeff = 1.0; //set to 1 if no shadow!
//TASK 2.4: look up depth in u_depthMap and set shadow coefficient (shadowCoeff) to 0 based on depth comparison
float zShadowMap = texture2D(u_depthMap, shadowMapTexCoord3D.xy).r;
if(shadowMapTexCoord3D.z > zShadowMap)
  shadowCoeff = 0.0;

//TASK 2.5: apply shadow coefficient to diffuse and specular part
return c_amb + shadowCoeff * (c_diff + c_spec) + c_em;
```

Self Shadowing

Try to disable subtraction of self shadowing bias in shader





http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-16-shadow-mapping/

EXTRA TASK: Smooth Shadows

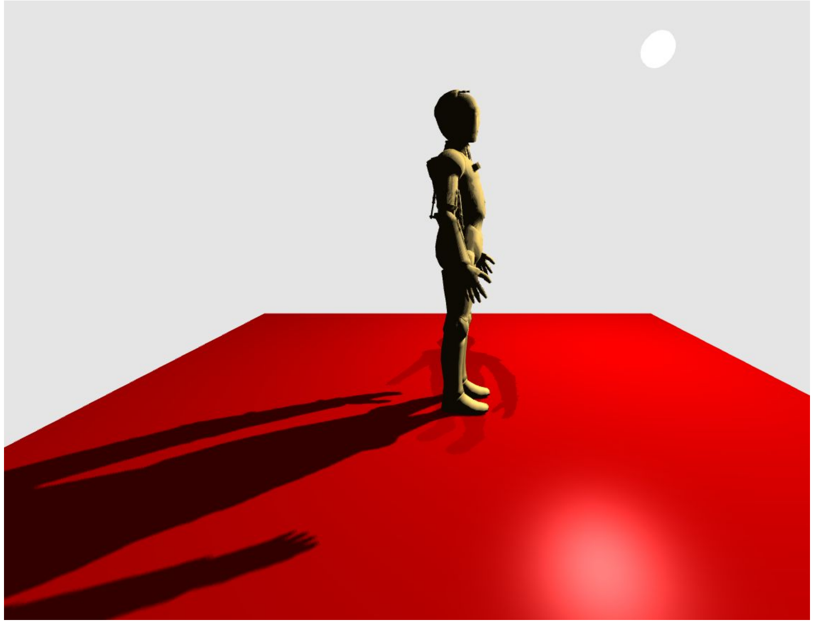
Goal:

Smooth shadow by sampling and averaging shadow coefficient over a 3x3 neighborhood in depth texture.

Hints:

You can use "for loops" in a shader! (E.g. loop x,y offsets from -1 to 1)

Texture coordinates are normalized (0 to 1) → Use texture size (u_shadowMapWidth, ...) to compute 1 step in x,y direction.

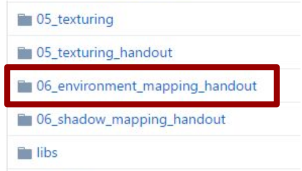


Solution: Smooth Shadows

JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

```
//EXTRA TASK: Improve shadow quality by sampling multiple shadow coefficients (a.k.a. PCF)
float avgShadowCoeff = 0.0;
for(float dx=-1.0; dx <= 1.0; dx++)
{
  for(float dy=-1.0; dy <= 1.0; dy++)
  {
    float subShadowCoeff = 1.0; //set to 1 if no shadow!
    float zShadowMap = texture2D(u_depthMap, shadowMapTexCoord3D.xy+vec2(dx/u_shadowMapWidth,dy/u_shadowMapHeight)).r;
    if(shadowMapTexCoord3D.z > zShadowMap)
      subShadowCoeff = 0.0;

    avgShadowCoeff += subShadowCoeff;
  }
}
shadowCoeff = avgShadowCoeff/9.0;
```

Environment Mapping



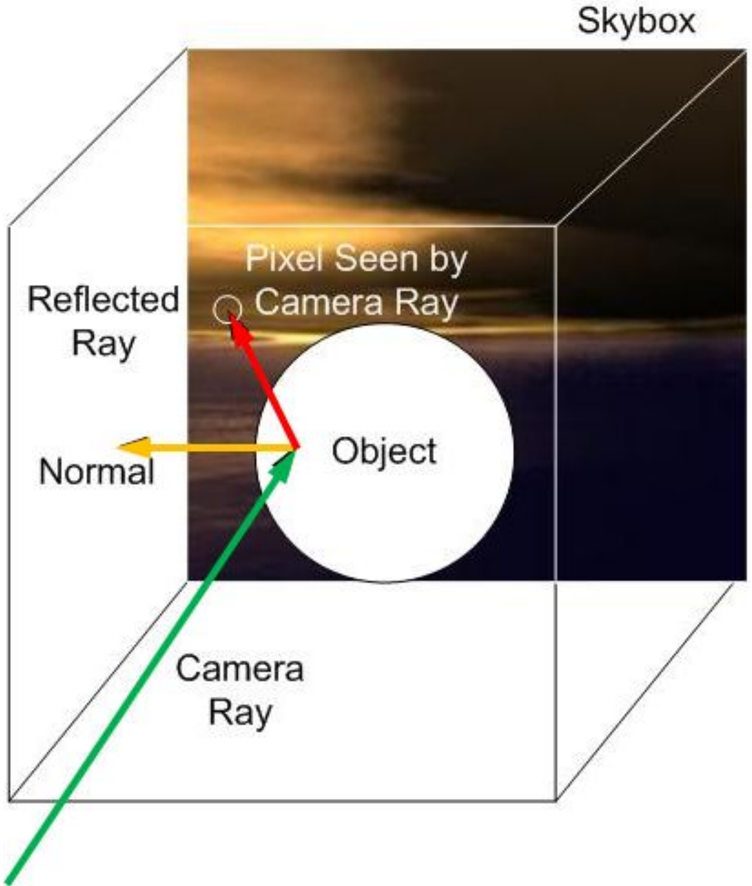
Environment Mapping

Cube Mapping

OpenGL supports cube maps

Lookup with 3D texture coordinates

We will set up a cube map containing six images forming an environment



Scene Description

First a space ship is rendered

- Use reflected camera ray for lookup → mirror like surface

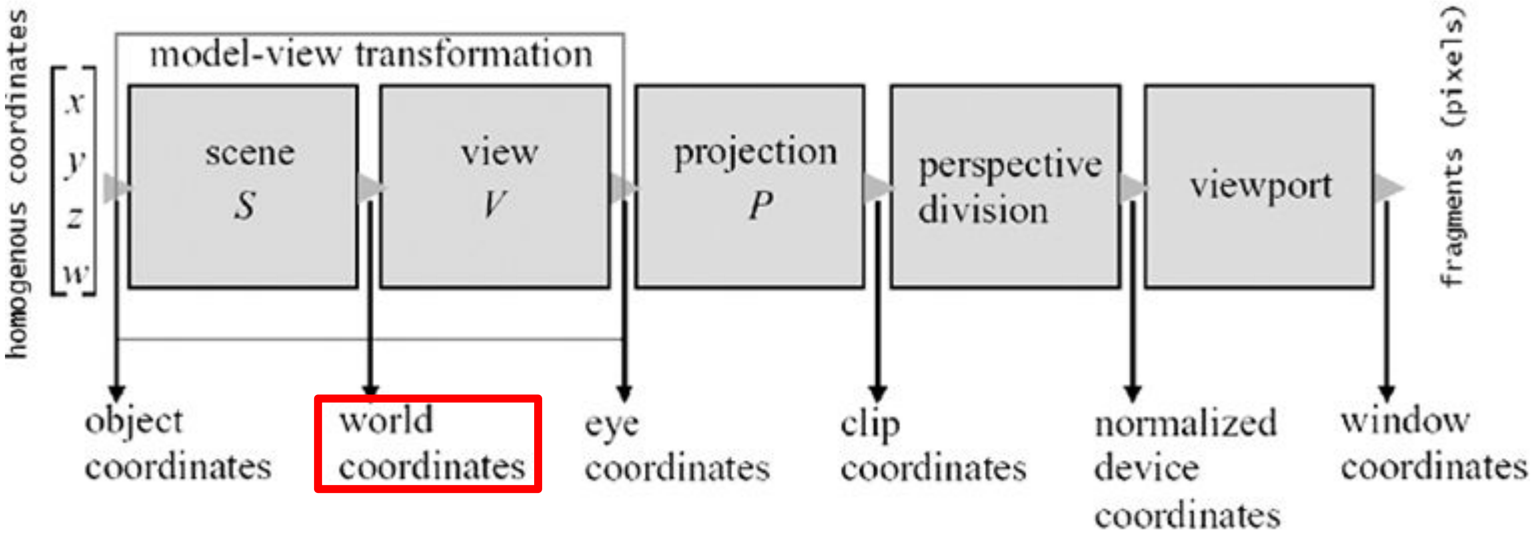
Second a sphere is rendered around the viewer

- Use non-reflected camera ray for lookup to directly display the environment (i.e. the stars)

Reminder: Coordinates Systems

The environment map represents the world around us

For our space scenario it defines the stars



Cube Map Texture Coordinates

We need the (reflected) camera ray for lookups in the cube map

in eye space of our camera (model-view transformation):

camera ray direction = vertex position

The environment map represents our world:

→ we have to transform camera ray direction from eye space to world space

use inverse view matrix to get from eye space to world space coordinates

```
invViewMatrix = mat4.invert(mat4.create(), context.viewMatrix);
```

since we deal with direction vectors, 3x3 matrix is sufficient (translation is ignored)

```
let invView3x3 = mat3.fromMat4(mat3.create(), context.invViewMatrix);
```

Difference to 2D texture

JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

Texture coordinates are 3D

Sampler in shader has type samplerCube

Texture lookup in shader is done with textureCube(...)

Texture target (texture type) is gl.TEXTURE_CUBE_MAP instead of gl.TEXTURE_2D

Texture is initialized with 6 images (one for each side of the cube)

Cube Map Texture Initialization

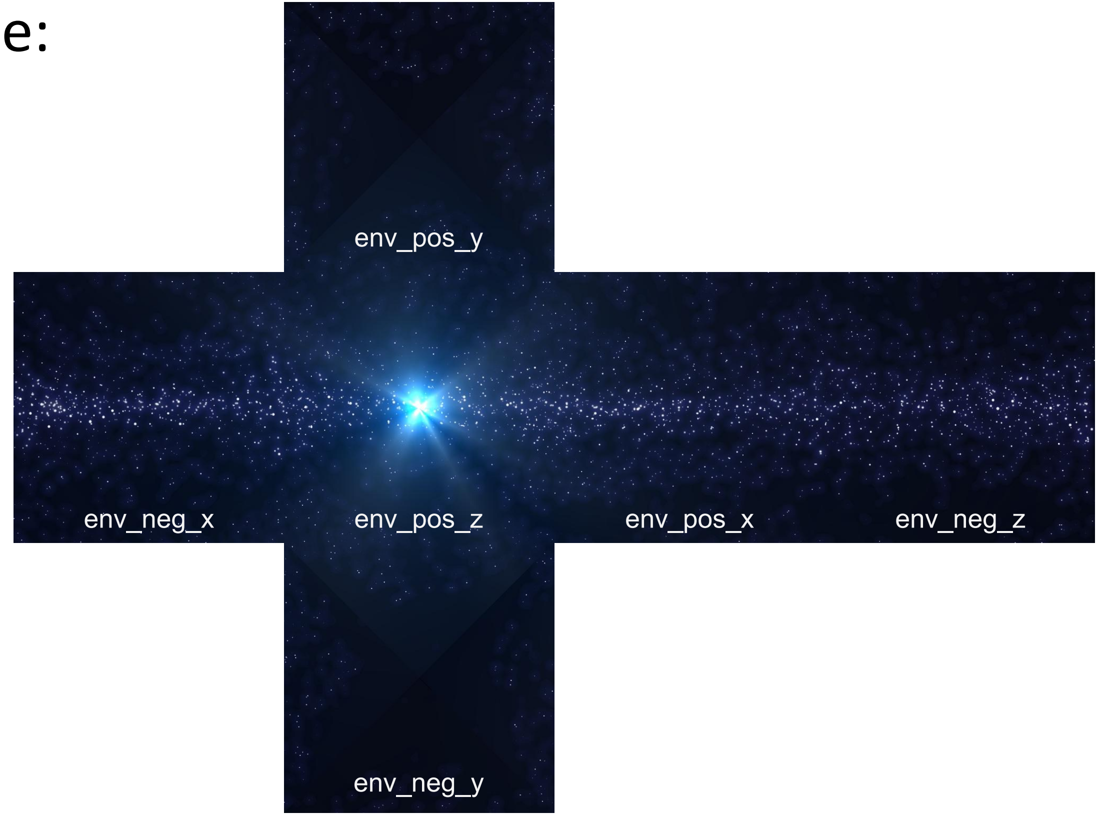
JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

Upload an image for each side of the cube:

```
function initCubeMap(resources) {
  //create the texture
  envcubetexture = gl.createTexture();
  //define some texture unit we want to work on
  gl.activeTexture(gl.TEXTURE0);
  //bind the texture to the texture unit
  gl.bindTexture(gl.TEXTURE_CUBE_MAP, envcubetexture);
  //set sampling parameters
  gl.texParameteri(gl.TEXTURE_CUBE_MAP, gl.TEXTURE_WRAP_S, gl.MIRRORED_REPEAT);
  gl.texParameteri(gl.TEXTURE_CUBE_MAP, gl.TEXTURE_WRAP_T, gl.MIRRORED_REPEAT);
  //gl.texParameteri(gl.TEXTURE_CUBE_MAP, gl.TEXTURE_WRAP_R, gl.MIRRORED_REPEAT); //will be available in WebGL 2
  gl.texParameteri(gl.TEXTURE_CUBE_MAP, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
  gl.texParameteri(gl.TEXTURE_CUBE_MAP, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
  //set correct image for each side of the cube map
  gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);//flipping required for our skybox, otherwise images don't fit together
  gl.texImage2D(gl.TEXTURE_CUBE_MAP_POSITIVE_X, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, resources.env_pos_x);
  gl.texImage2D(gl.TEXTURE_CUBE_MAP_NEGATIVE_X, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, resources.env_neg_x);
  gl.texImage2D(gl.TEXTURE_CUBE_MAP_POSITIVE_Y, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, resources.env_pos_y);
  gl.texImage2D(gl.TEXTURE_CUBE_MAP_NEGATIVE_Y, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, resources.env_neg_y);
  gl.texImage2D(gl.TEXTURE_CUBE_MAP_POSITIVE_Z, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, resources.env_pos_z);
  gl.texImage2D(gl.TEXTURE_CUBE_MAP_NEGATIVE_Z, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, resources.env_neg_z);
  //unbind the texture again
  gl.bindTexture(gl.TEXTURE_CUBE_MAP, null);
}
```

Cube Map Texture

Texture we'll use:



Task 3: Cube Mapping

JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

Goal:

Show stars and their reflection on spaceship.

Tasks:

3.1 Compute camera ray in vertex shader
3.2 Reflect camera ray in fragment shader
3.3 Do texture lookup in cube map

Source code now in 06_environment_mapping folder!

Solution: Cube Mapping

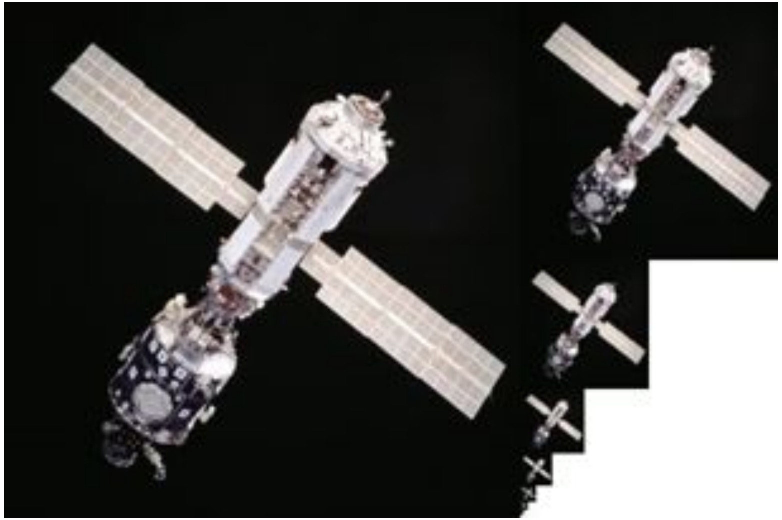
Task 3.1 - Vertex Shader:

```
v_cameraRayVec = u_invView * eyePosition.xyz;
```

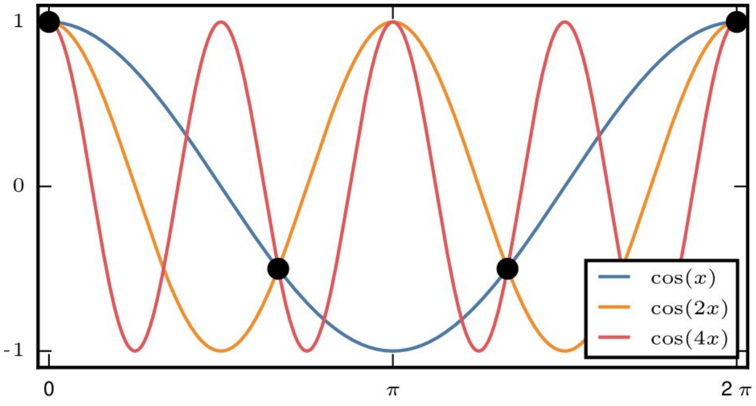
Task 3.2,3.3 - Fragment Shader:

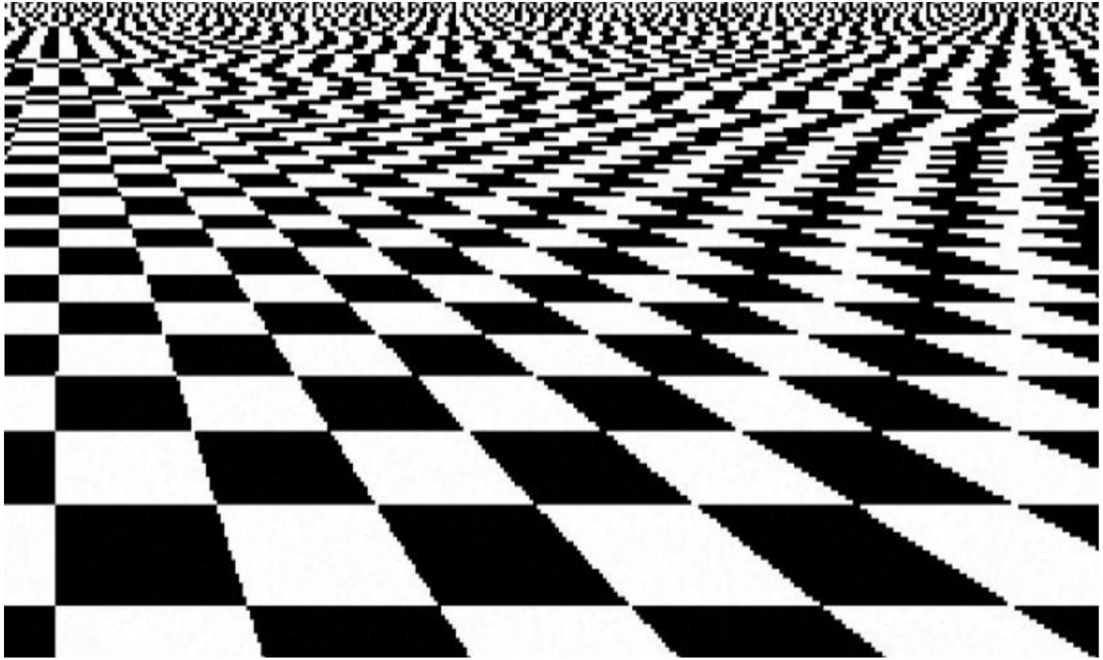
```
vec3 texCoords;
if(u_useReflection)
    //TASK 3.2: compute reflected camera ray
    texCoords  = reflect(cameraRayVec, normalVec);
    //texCoords = vec3();
else
    texCoords = cameraRayVec;

//TASK 3.3: do texture lookup in cube map using the textureCube function
gl_FragColor = textureCube(u_texCube, texCoords);
```

Texture Filtering



Aliasing





use texture filtering to reduce aliasing

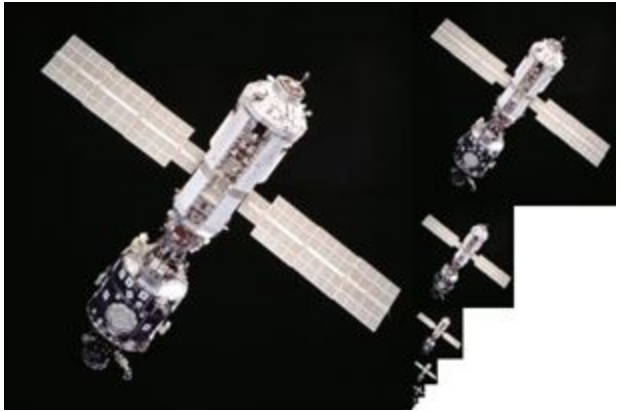
Demo: https://jku-icg.github.io/cg_demo/00_texturing/

Mipmapping

Low Resolution Versions of Texture

Mipmapping chooses the best texture size depending on the distance the texture is viewed from

Press “m“ to enable mipmapping in our example
Avoids flickering stars, but adds blur to reflections



Generate Mipmaps:

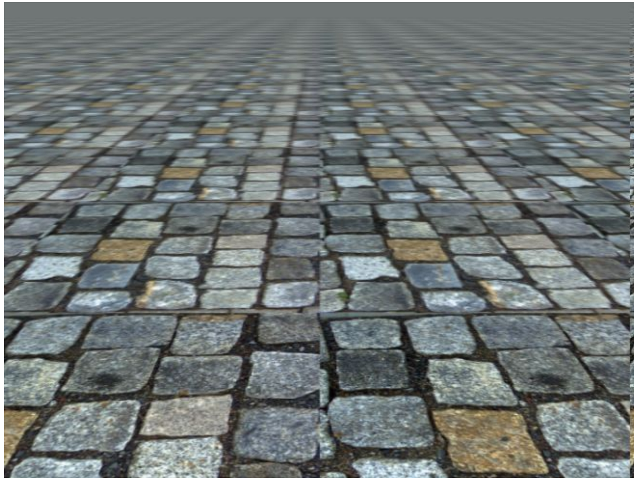
gl.generateMipmap(gl.TEXTURE_CUBE_MAP); during texture definition (built by iterative downsampling)

Enable Mipmaps:

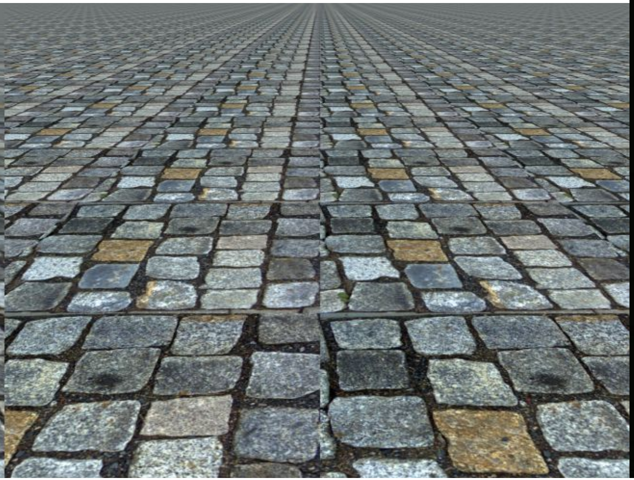
set gl.TEXTURE_MIN_FILTER parameter to gl.LINEAR_MIPMAP_LINEAR

Anisotropic Filtering

Improves texture filter quality for oblique viewing angles by non-isotropic filtering:



mipmapping



mipmapping + AF

Anisotropic Filtering

JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

Improves texture filter quality for oblique viewing angles

press “i”-key

This restores some details of the reflections but still avoids aliasing effects.

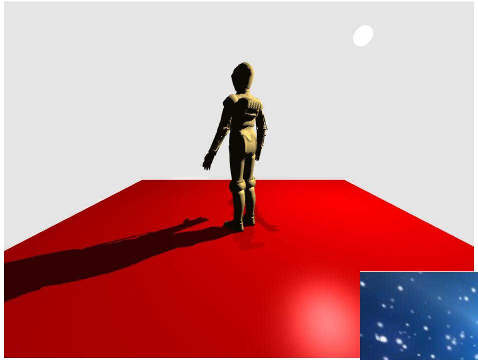
no mipmapping | mipmapping | mipmapping + AF

Recap

JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

Shadow Mapping

- Overview
- Recap: Render to Texture
- Depth Comparison
- Eye-to-Light Matrix
- Smooth Shadows



Environment Mapping

- Cube Mapping
- Differences to 2D Textures

Texture Filtering

- Mipmapping, Anisotropic Filtering

CG Project: Multiple Shaders in Scene

JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF COMPUTER GRAPHICS

Remember:

- Uniforms set for one ShaderSGNode are not transfered to another ShaderSGNode!
  - E.g. LightSGNode only affects one ShaderSGNode!

Important:

- Make sure ShaderSGNode is added to scene graph before nodes which set any uniform parameters of the shader
  - E.g. LightSGNode should be child of ShaderSGNode!
- Make sure to set all required uniform parameters before adding first RenderSGNode

Workaround for duplicate light specification:

- MaterialSGNode allows to add light sources to .lights variable
- Instead of adding lights + transformations again to other ShaderSGNode do:
  - Add LightSGNode + light transformations to first ShaderSGNode
  - Add **same** LightSGNode to **first** material in **second** ShaderSGNode (sets again light uniform params)

Thanks!
Have fun with your CG-Projects.

Questions / Feedback: cg-lab@jku.at

Final Submission Deadline:
22.06.2021