Lesson 2 – Shadows

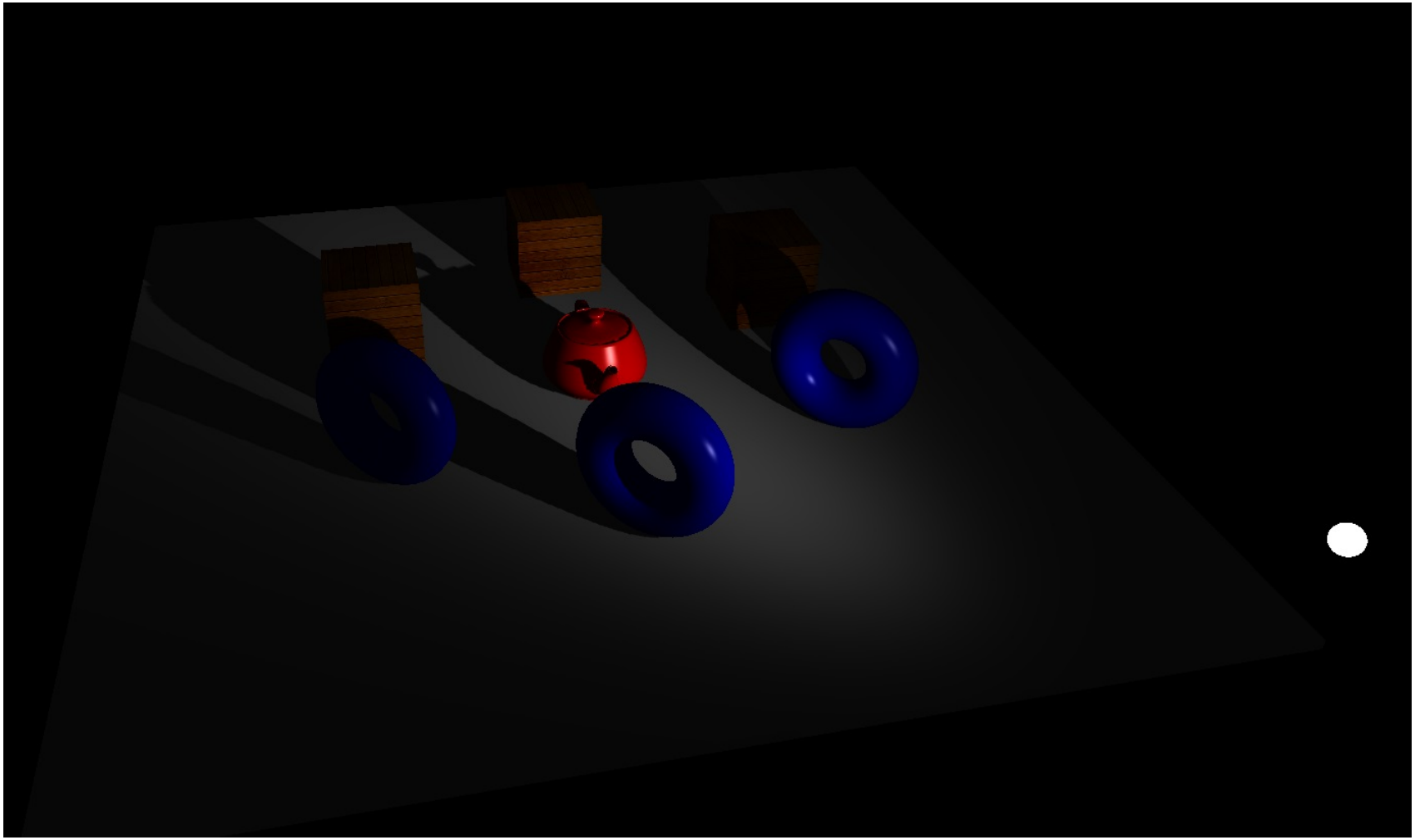
PV227 – GPU Rendering

Jiří Chmelík, Jan Čejka
Fakulta informatiky Masarykovy univerzity

24. 09. 2019

Shadow mapping

Motivation:



Shadow mapping

Lance Williams, 1978, *Casting curved shadows on curved surfaces*

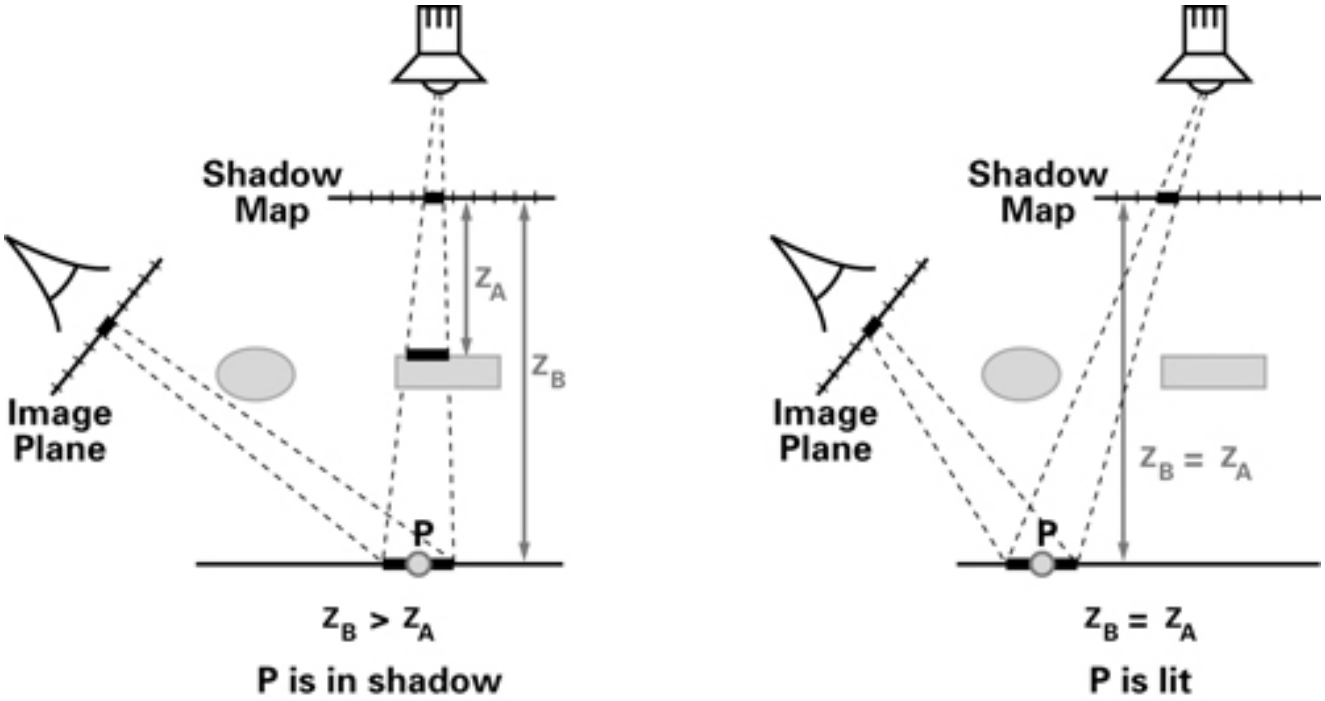


Illustration from *NVIDIA: Cg Tutorial*.

Shadow mapping

Individual steps:

- Allocate shadow texture, create a framebuffer object.
- Render the whole scene from the position of light into the shadow texture.
- Render the whole scene again, now from the position of the viewer. When rendering each object, use the shadow texture and compare the depths.

Let's add shadows to project Cv2 (in Study materials).

Task: Render from the position of the light

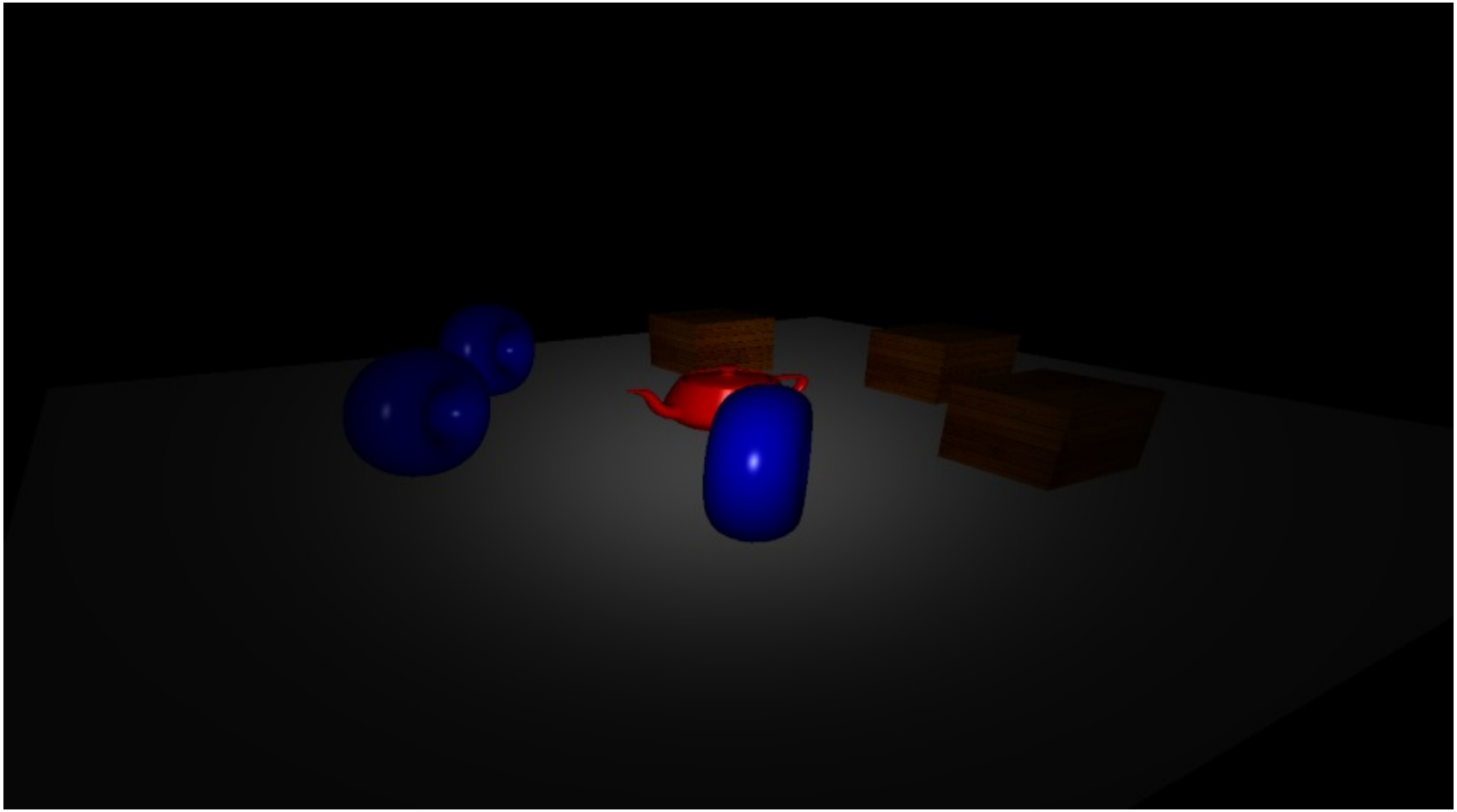
Let's split this task to ease debugging:

- **Task 1**
  - Compute and update projection and view matrices for rendering from the position of the light.
  - Render from the position of the light.
  - Render into the main window.
  - Use the same shaders that are used when rendering from the viewer.
  - Do not render the object that represents the light.

  Hint: Use function glm::perspective to create the projection matrix.

  Hint: Use function glm::lookAt to create the view matrix.

Task: Render from the position of the light

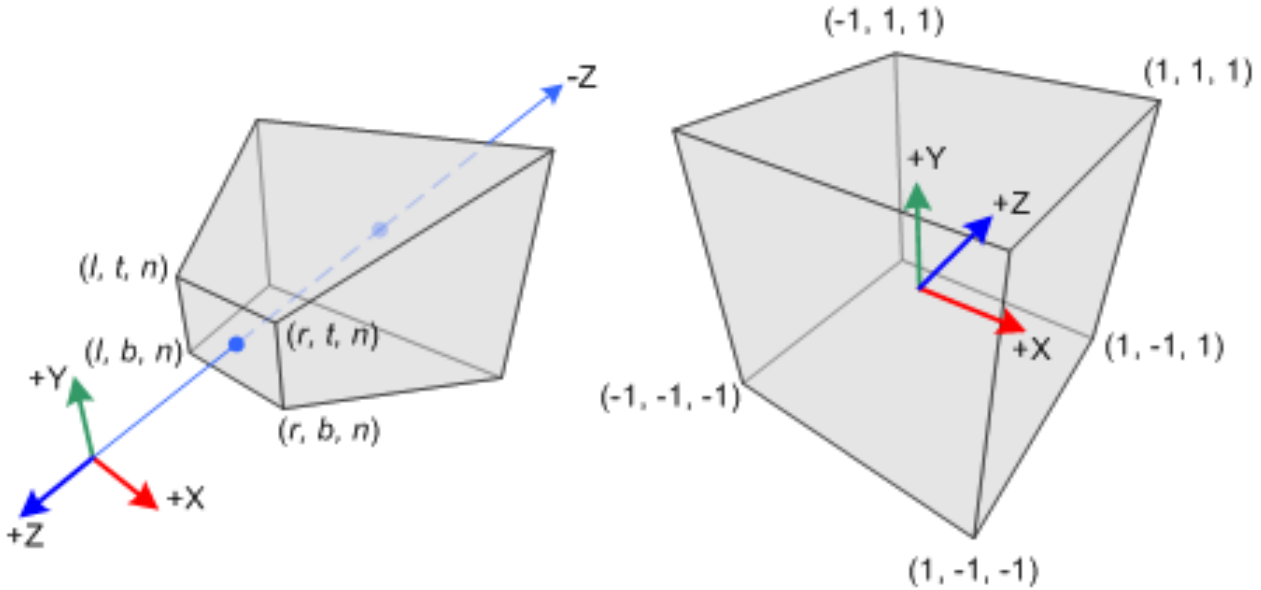


Task: Render from the position of the light

- **Task 2**
  - Render the scene into the shadow map instead (using the same shaders as before).
  - Display the shadow map on the screen.
- **Task 3 (Optional)**
  - Prepare and use simplified shaders for rendering into the shadow texture, i.e. don't compute the lighting, don't use textures, don't use materials, don't output any color.

Task: Render from the position of the light

Theory: Transformations and depth



Normalized device coordinates, from *songho.co*

$$V_{ndc} = DivideByW(proj * view * V_{world})$$

Theory: Transformations and depth

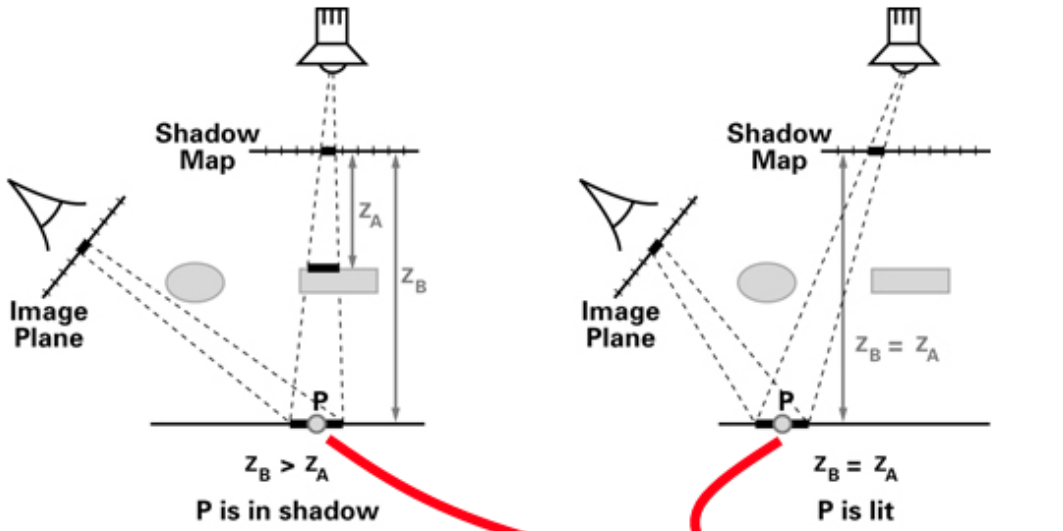
$$V_{ndc} \in \Big[(-1,-1,-1);(1,1,1)\Big]$$

After applying glViewport and glDepthRange:

$$V_{screen} \in \Big[(x,y,near);(x+w,y+h,far)\Big]$$

This $V_{screen}.z$ is used for depth tests, stored into the depth buffer etc.

Theory: Transformations and depth




```
void main()
{
    // Compute the lighting
    vec3 N = normalize(inData.normal_ws);
    vec3 Eye = normalize(eye_position - inData.position_ws);
```

- Texture coordinate?
- Reference value to compare?

Theory: Transformations and depth

Transform *inData.position_ws* ($= V_{world}$)

$$V_{ndc}^{light} = DivideByW(proj^{light} * view^{light} * V_{world})$$

$$V_{ndc}^{light} \in \Big[(-1,-1,-1);(1,1,1)\Big]$$

Then use transformation and scale

$$V = T/S * V_{ndc}^{light}$$

$$V \in \Big[(0,0,0);(1,1,1)\Big]$$

Here, $V.xy$ is the texture coordinate, $V.z$ is the reference value

Theory: Transformations and depth

Transform *inData.position_ws* ($= V_{world}$)

$$V_{ndc}^{light} = DivideByW(proj^{light} * view^{light} * V_{world})$$

$$V_{ndc}^{light} \in \Big[(-1,-1,-1);(1,1,1)\Big]$$

Then use transformation and scale

$$V = T/S * V_{ndc}^{light}$$

$$V \in \Big[(0,0,0);(1,1,1)\Big]$$

Here, $V.xy$ is the texture coordinate, $V.z$ is the reference value

Theory: Transformations and depth

Furthermore:

$$V = T/S * DivideByW(proj^{light} * view^{light} * V_{world})$$

is actually the same as:

$$V = DivideByW(\underbrace{T/S * proj^{light} * view^{light}}_{\text{Shadow matrix}} * V_{world})$$

Task: Render with shadows

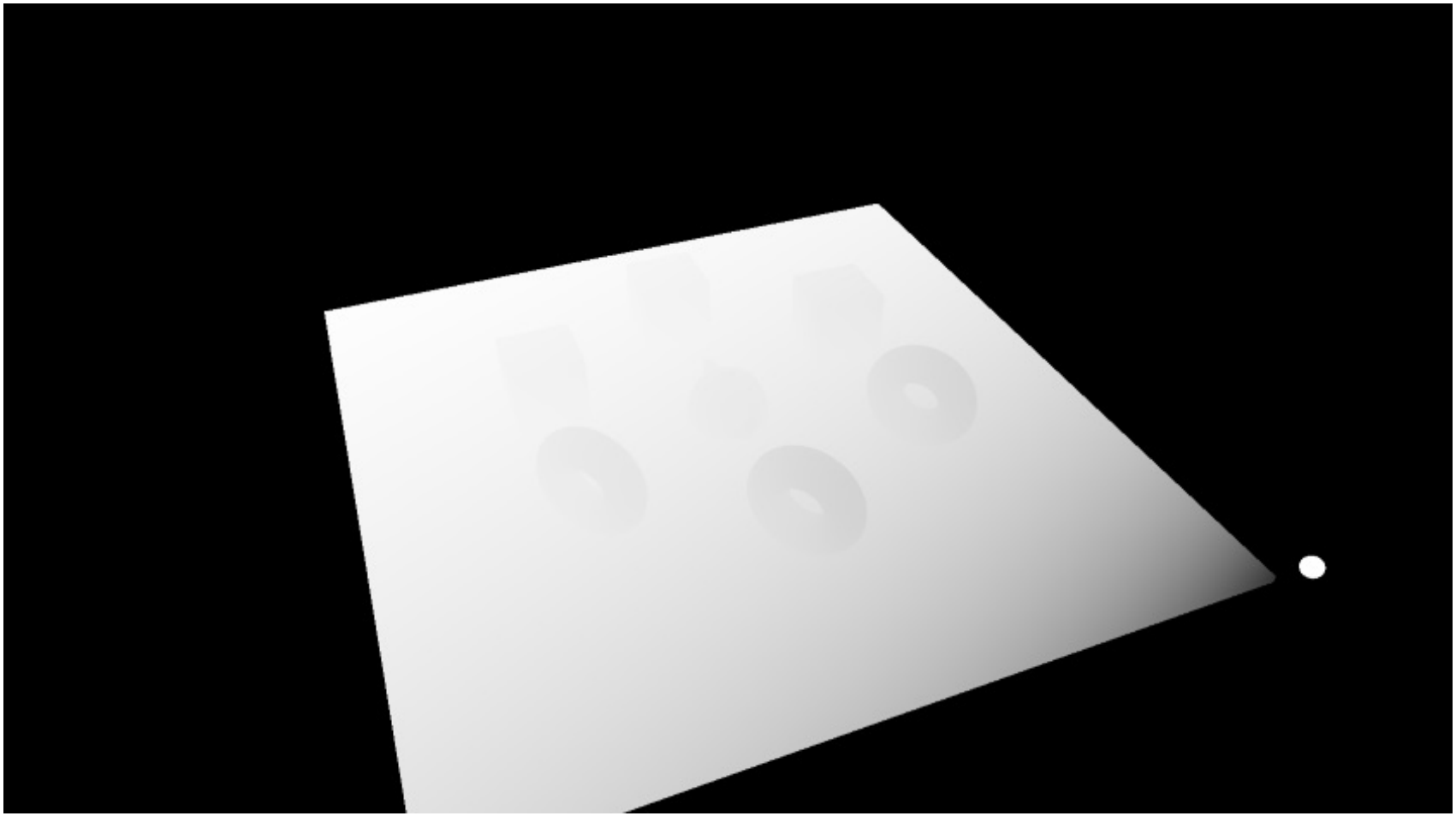
Let's again split this task to ease debugging:

- **Task 4**
  - Compute and update shadow matrix, send it into shaders.
  - Transform inData.position_ws.
  - Output the reference value $V.z$ as a color in grayscale.

  When rendered from the position of the light, this should be the same as the shadow texture (except for the background).

  When rendered from the position of the viewer, the color corresponds to the distance from the light.

Task: Render with shadows



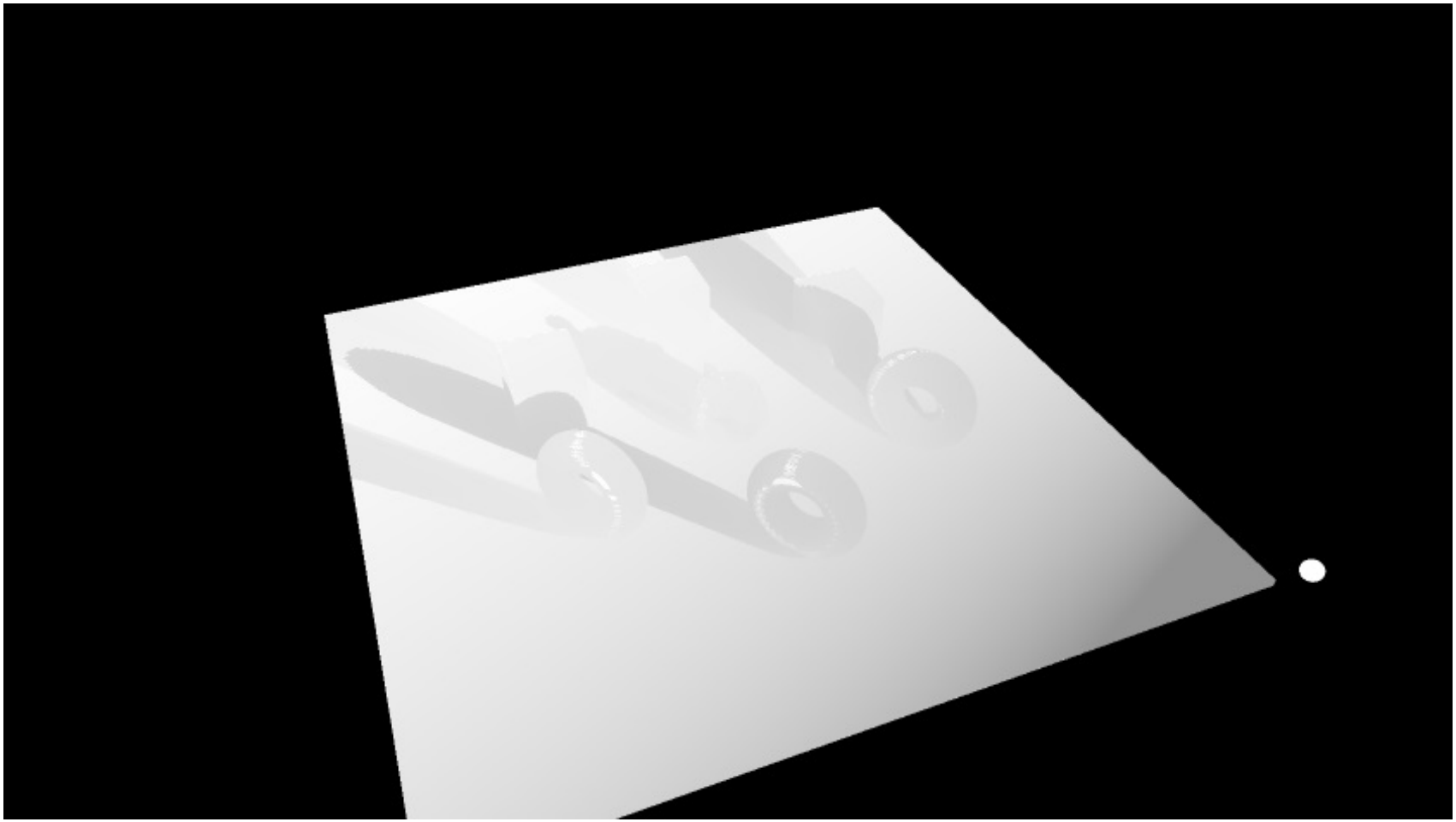
Task: Render with shadows

- **Task 5**
  - Use the texture coordinates to obtain the value in the shadow texture.
  - Display this value as a grayscale color.

When rendered from the position of the light, this should again be the same as the shadow texture (except for the background).

When rendered from the position of the viewer, the color corresponds to the closest distance in the shadow map. Since the closer values correspond to darker colors, this looks like some sort of shadows.

Task: Render with shadows

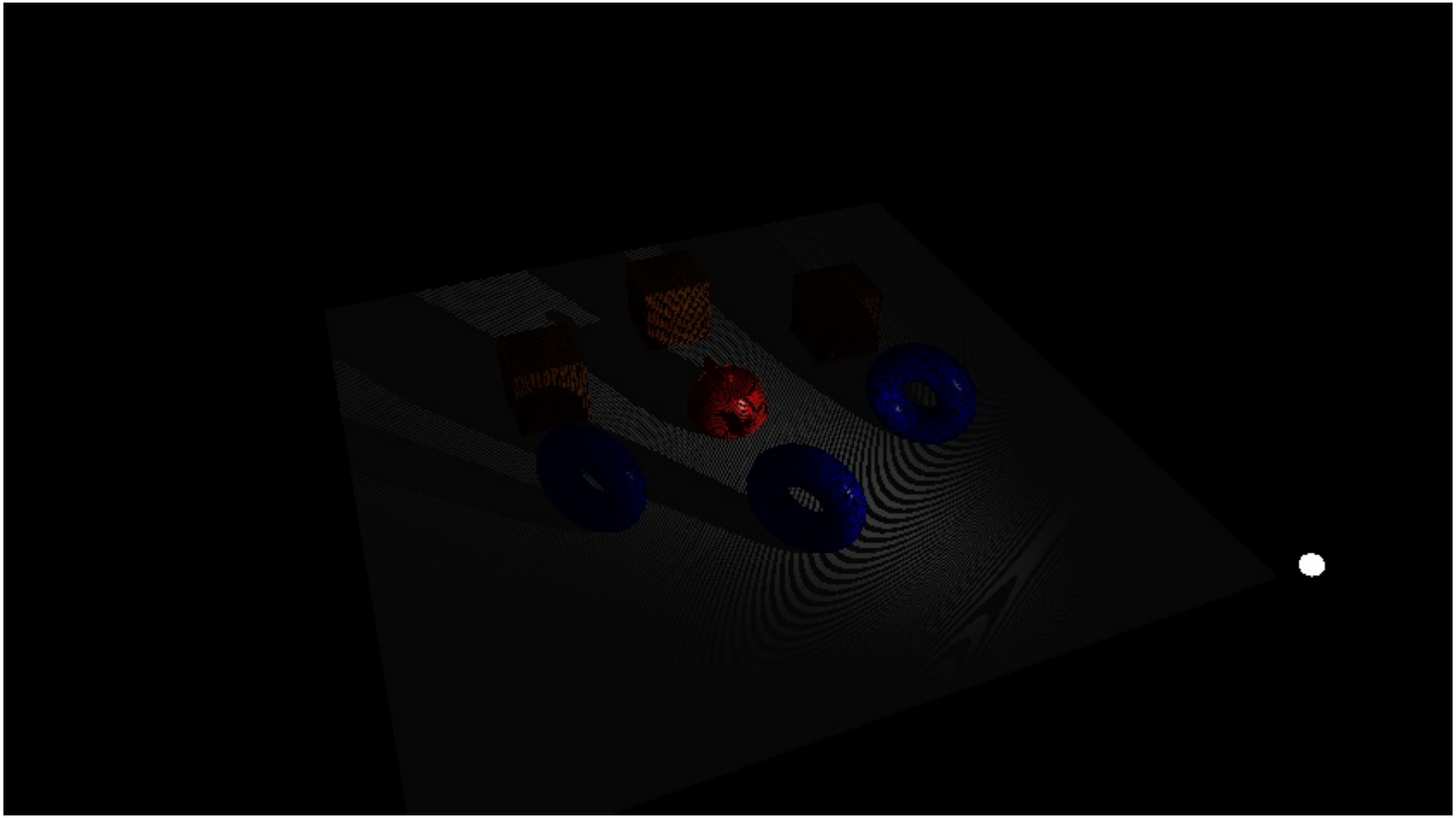


Task: Render with shadows

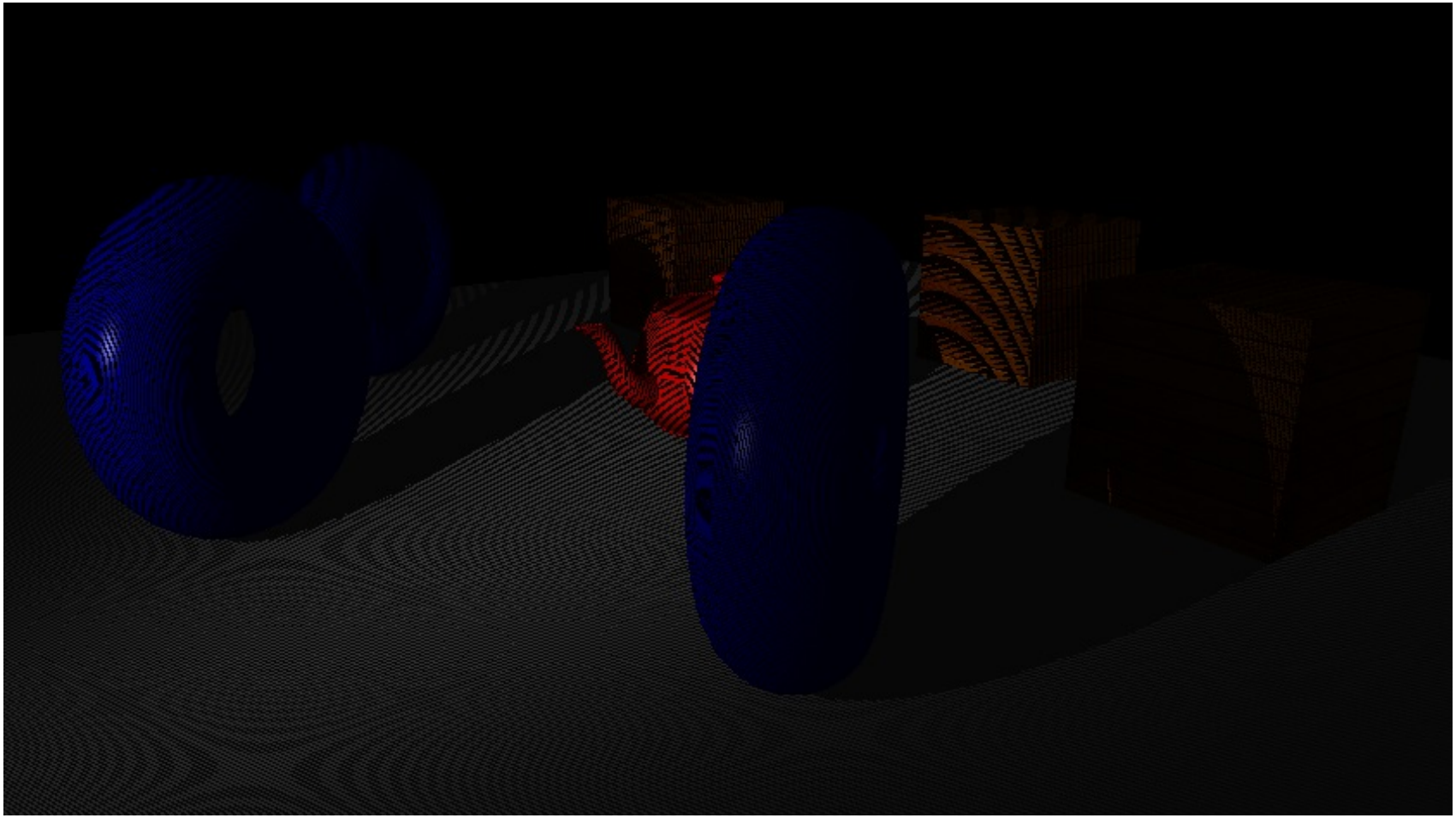
- **Task 6**
  - Compare the reference value and the value in the shadow texture
  - Apply the result to the lighting.
  - The result is shadows (and a lot of artefacts).

  Hint: Use float to represent the result of the comparison, use 1.0 for “not-in-shadow”, and 0.0 for “in-shadow”.

Task: Render with shadows



Artefacts: Acne and Peter-Panning



Shadow Acne

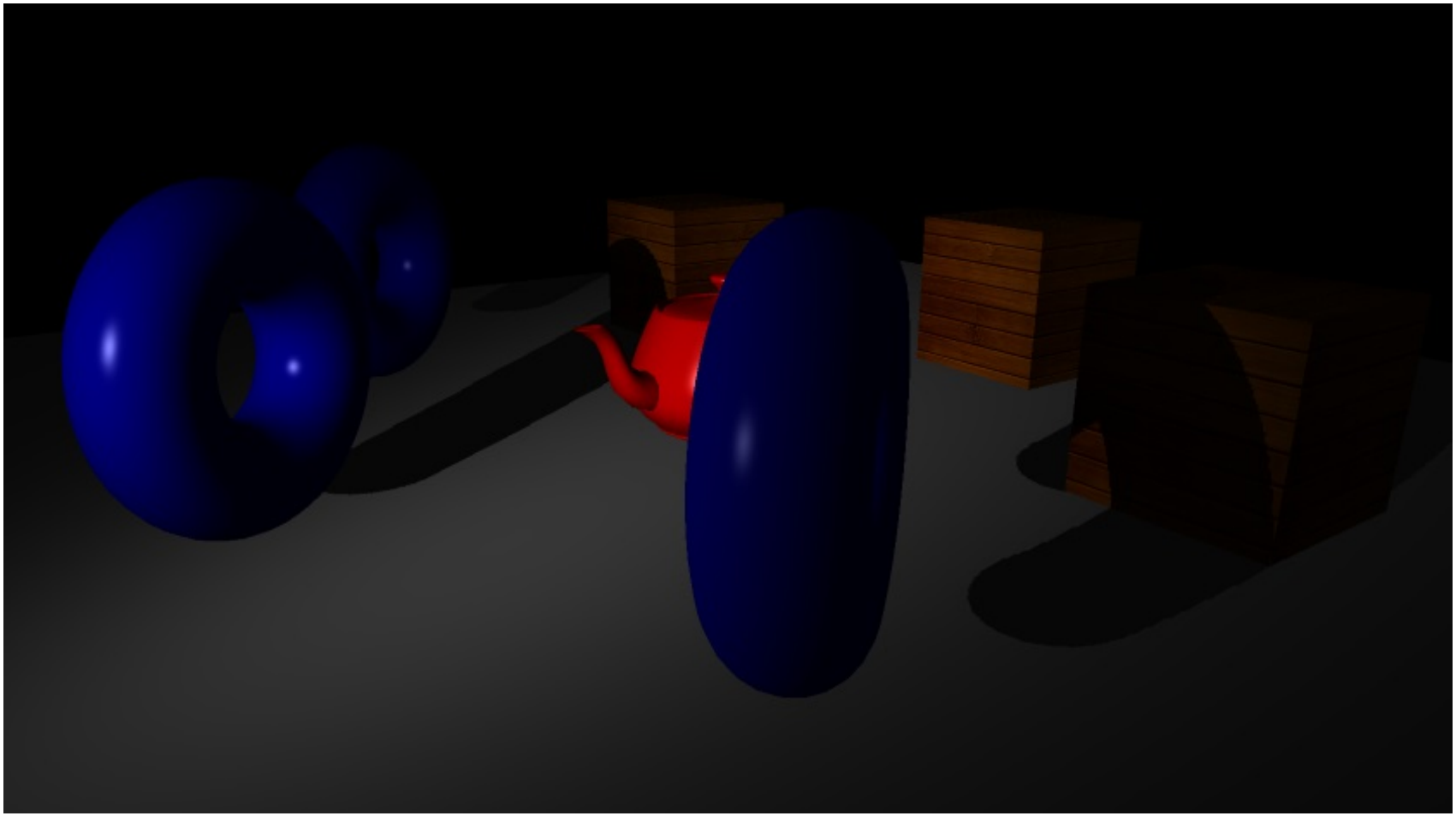
Artefacts: Acne and Peter-Panning

Shadow Acne:

- Objects cast shadows on themselves when they shouldn't
- Caused by quantization of the depth value when stored into the texture
- To fix this issue:
  - When rendering into the depth texture, use near and far planes as close to the other as possible.
  - Offset the depth before storing into the depth texture, preferably by using glPolygonOffset.
  - Offset the reference value a bit before comparing it with the value in the texture (using an additional translation matrix or add a value in the shader).
  - When rendering into the depth texture, render only the back faces, do not render the front faces.

Artefacts: Acne and Peter-Panning

Don’t overshoot the offset, the object would levitate, creating a Peter-Panning effect.



Peter-Panning

Task: Remove the artefacts

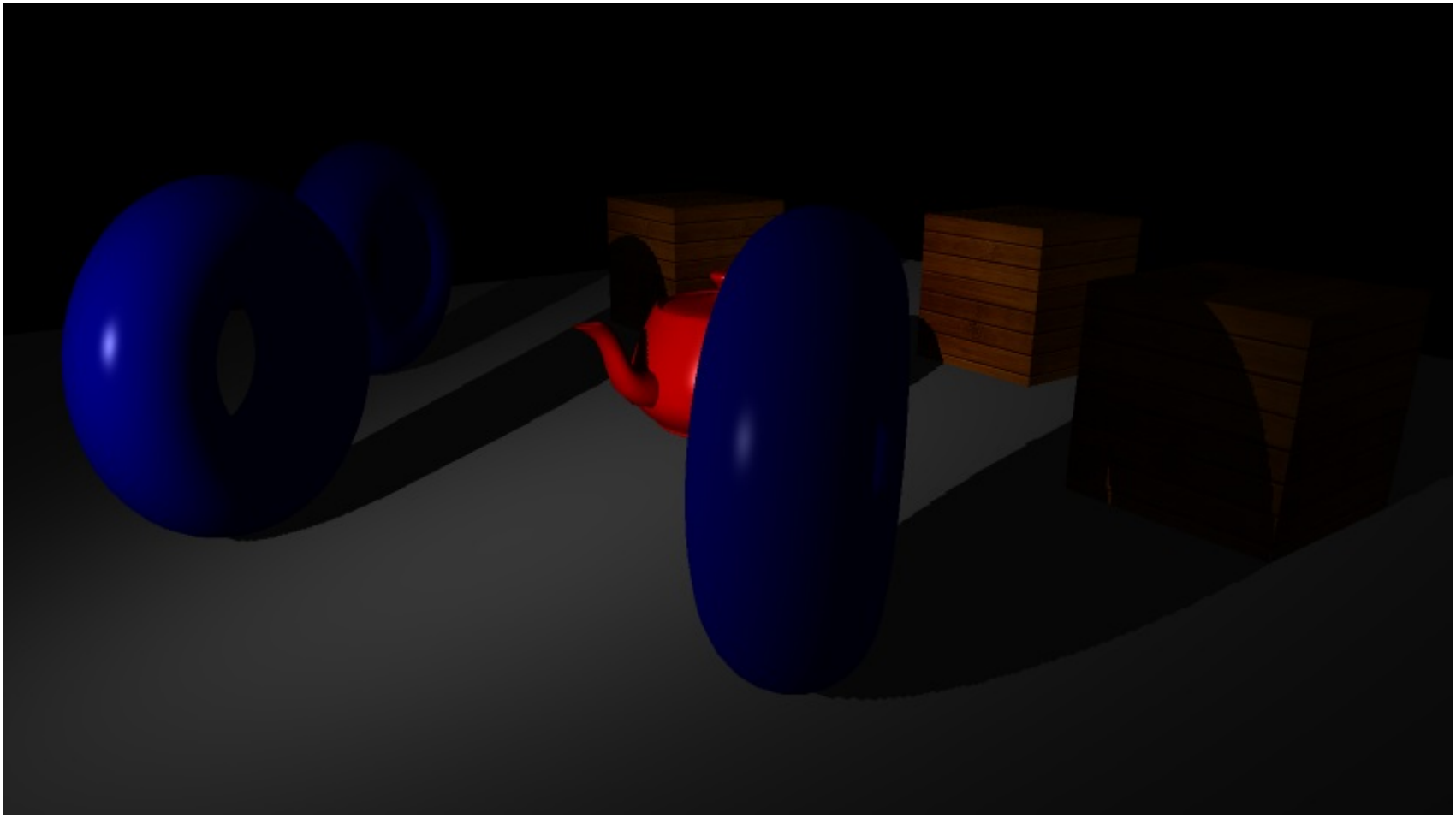
Task 7: Choose one of the following to remove the Shadow Acne artefact:

- Add an additional translation matrix to the shadow matrix
  Hint: offseting by -0.002 *in NDC seems to be OK*

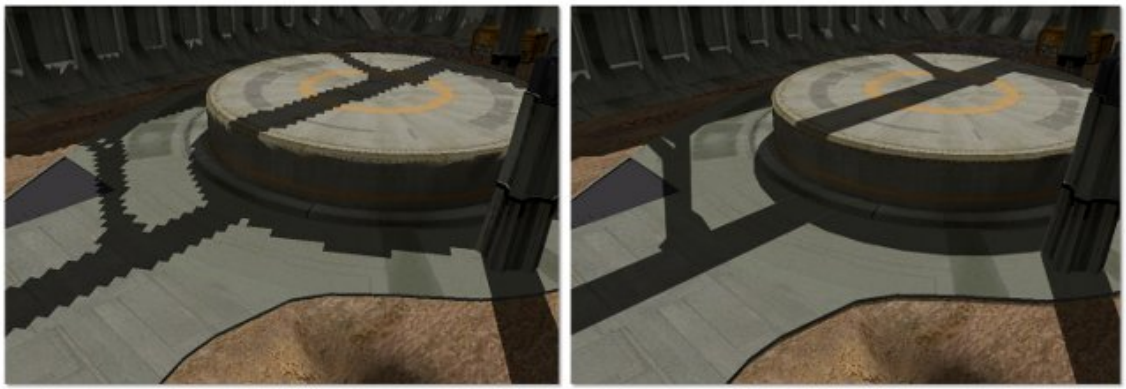
- Use glPolygonOffset when rendering into shadow texture
  Hint: parameters glPolygonOffset(2.0f, 0.0f) seem to be OK
  Hint: don't forget to enable/disable with glEnable(GL_POLYGON_OFFSET_FILL)

- Cull away front faces when rendering into the depth texture
  Hint: glEnable(GL_CULL_FACE), glCullFace(GL_FRONT)

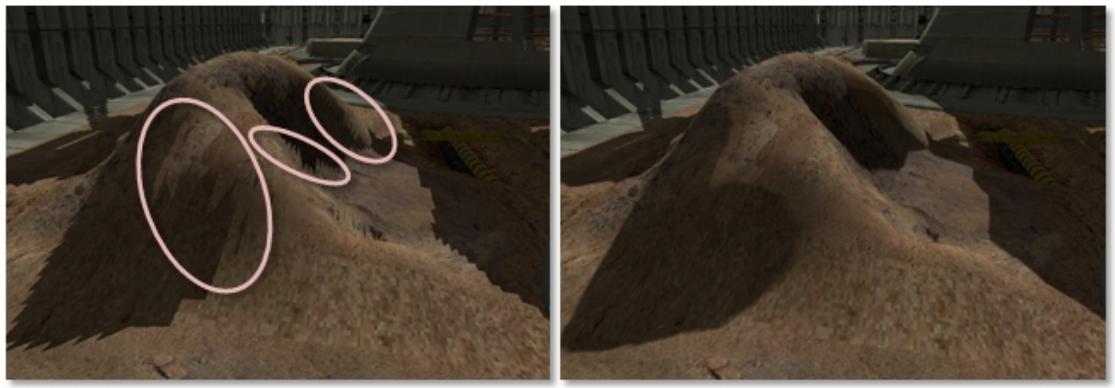
Task: Remove the artefacts



Other artifacts



Perspective aliasing



Projective aliasing

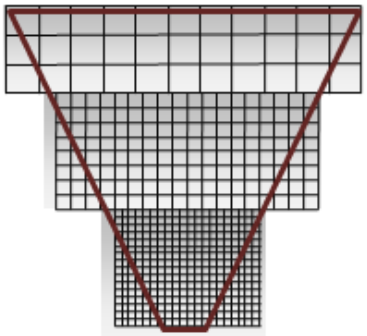
Other artifacts

Animation artifacts: Shimmering edges

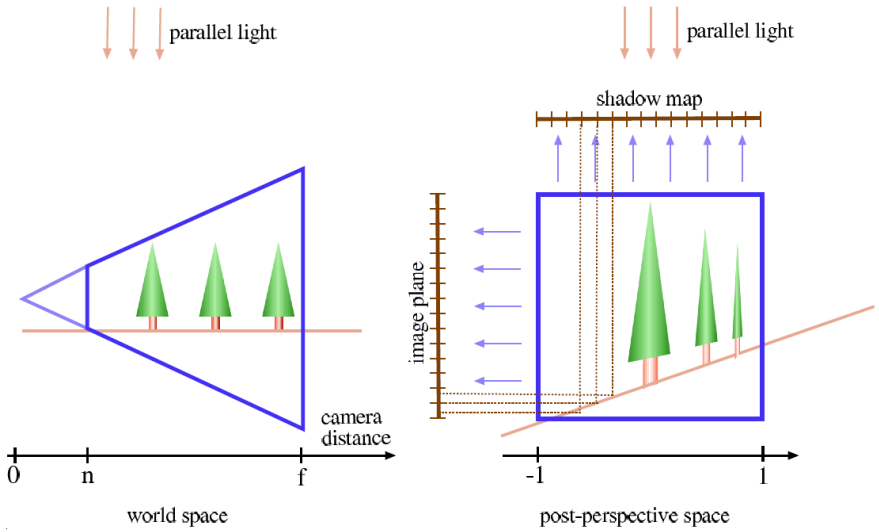
Images from: *Microsoft: Common Techniques to Improve Shadow Depth Maps*

Techniques improving shadows

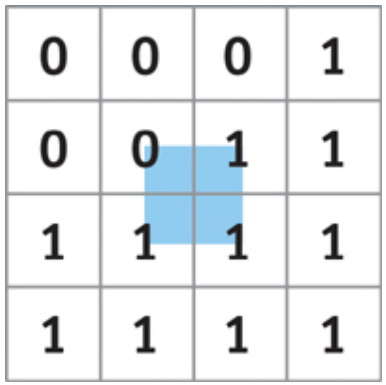
- Reducing aliasing:
  - Cascaded Shadow Maps
  - (Light Space) Perspective Shadow Maps
- Smoothing edges:
  - Percentage Closer Filtering



CSM



PSM



PCF

OpenGL support for shadow maps

OpenGL and GLSL provides additional functions for shadow maps:

- GLSL function *textureProj*
  - divides by w before performing the texture lookup:

    texture(shadow_tex, tex_coord.xy / tex_coord.w);

    is the same as

    textureProj(shadow_tex, tex_coord);

OpenGL support for shadow maps

- Shadow textures
  - *glTexParameteri(..., GL_TEXTURE_COMPARE_MODE, ...)*
    - ⋆ GL_COMPARE_REF_TO_TEXTURE compares value in texture with the reference value
    - ⋆ GL_NONE no comparison is done
  - *glTexParameteri(..., GL_TEXTURE_COMPARE_FUNC, ...)*
    - ⋆ GL_LEQUAL comparison returns 1.0 when the reference value is less or equal that the value in the texture.
  - Special GLSL sampler *sampler2DShadow*
    - ⋆ *float factor = texture(shadow_tex, tex_coord.xyz / tex_coord.w);* is the same as *float factor = textureProj(shadow_tex, tex_coord);*
  - Support for linear filters and mipmapping, which is done **after** the comparison.

Task: Use OpenGL functions

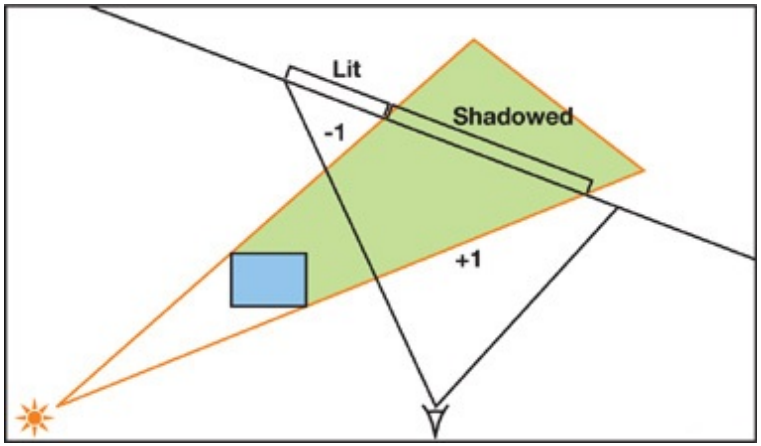
Task 8: Use *textureProj* and comparison.

- Use *textureProj*.
- Use *GL_COMPARE_REF_TO_TEXTURE* and *sampler2DShadow* for shadows.
- Use *GL_NONE* and *sampler2D* for displaying the shadow texture.

Foods for thought

Other techniques for shadows:

- Stencil shadows



Stencil volume

- RayTracing
- Shadows for ambient light? (Look up ambient occlusion)

Foods for thought

- How to solve transparent objects? What about semi-transparent objects? What about colored semi-transparent objects?
- What about multiple lights? (Optimization: not all of them needs shadows.)
- What about point lights? What about directional lights like the sun?

Further reading

- Microsoft: Common Techniques to Improve Shadow Depth Maps
  `https://msdn.microsoft.com/en-us/library/windows/desktop/ee416324(v=vs.85).aspx`
- Microsoft: Cascaded Shadow Maps
  `https://msdn.microsoft.com/en-us/library/windows/desktop/ee416307(v=vs.85).aspx`
- Wikipedia: Shadow mapping, section “Shadow mapping techniques”
  `https://en.wikipedia.org/wiki/Shadow_mapping`
- GPU Gems, GPU Gems 2, GPU Gems 3