Praktikum: Echtzeit Computergrafik



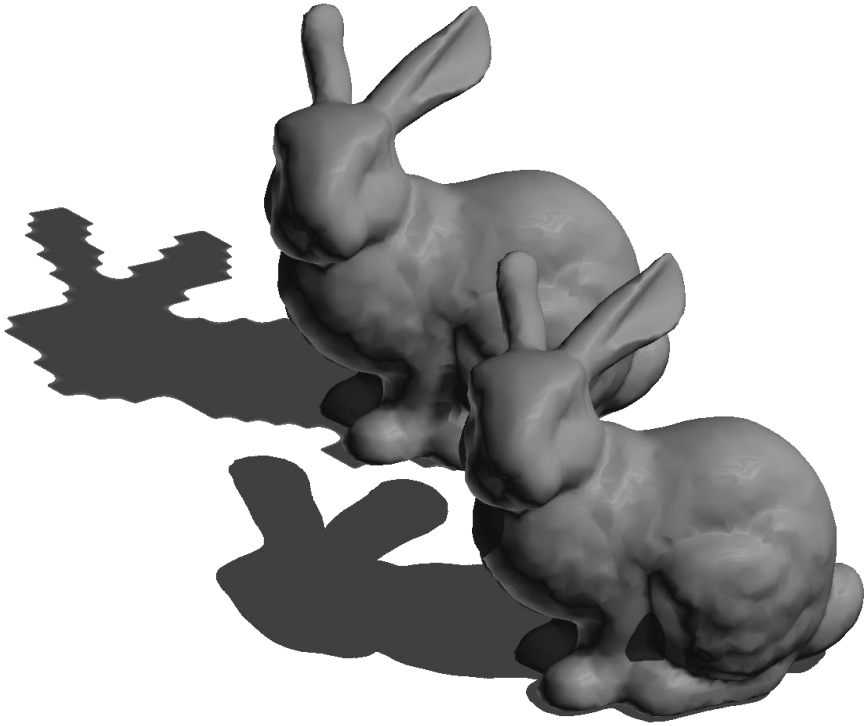
tum.3D

computer graphics & visualization

Bonus Assignment

Shadow mapping

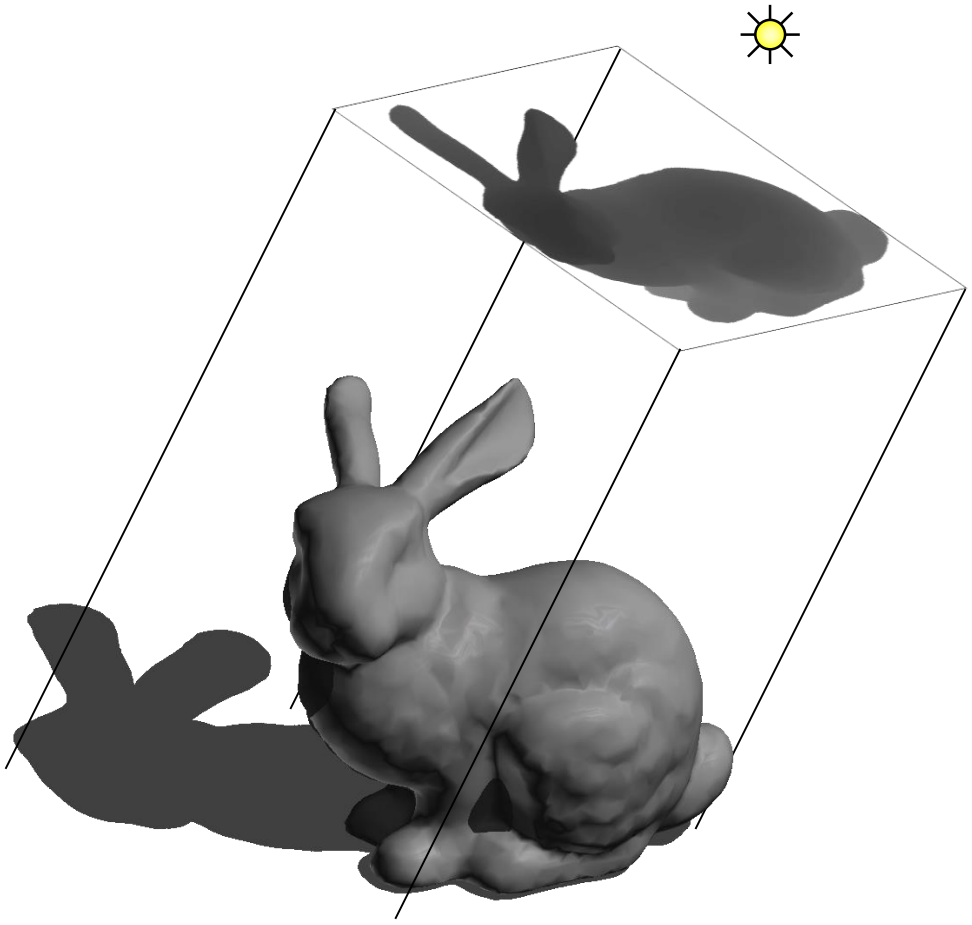
Methods



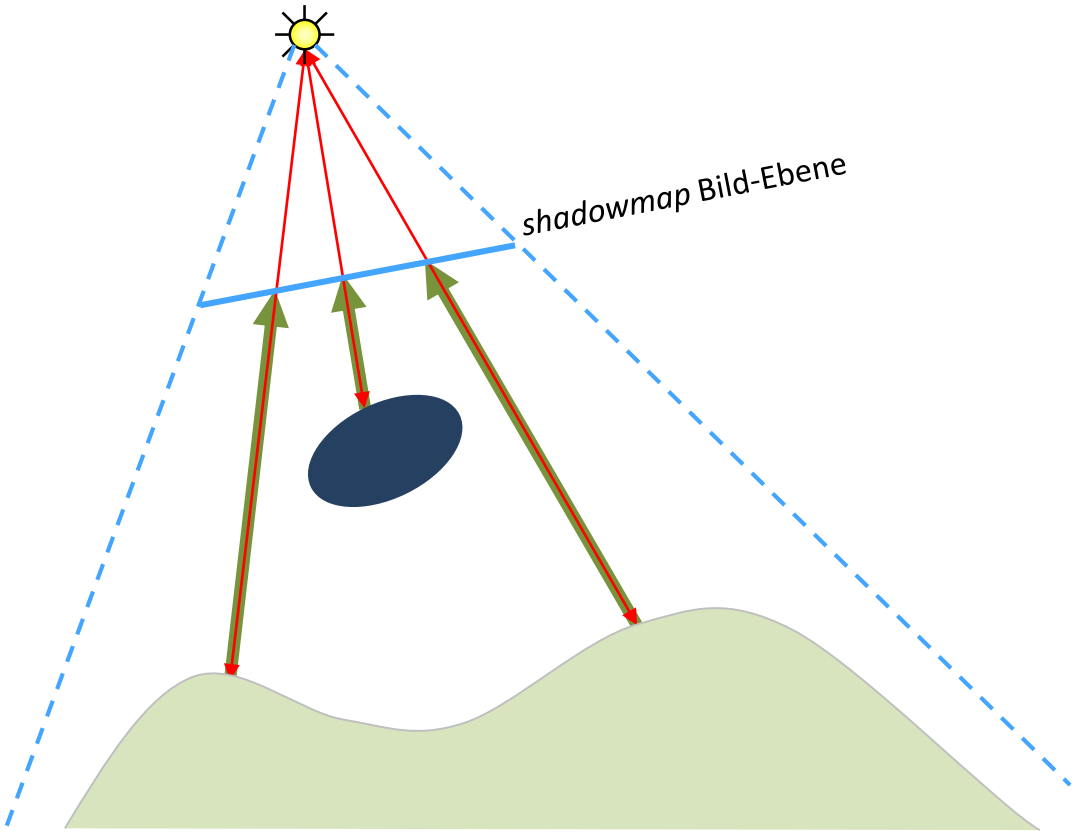
Shadow Map

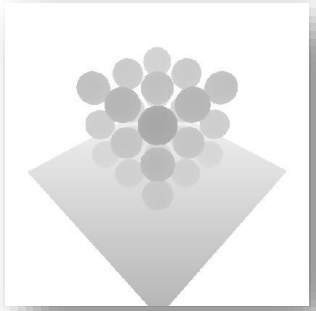
Shadow Volume

Shadow Map



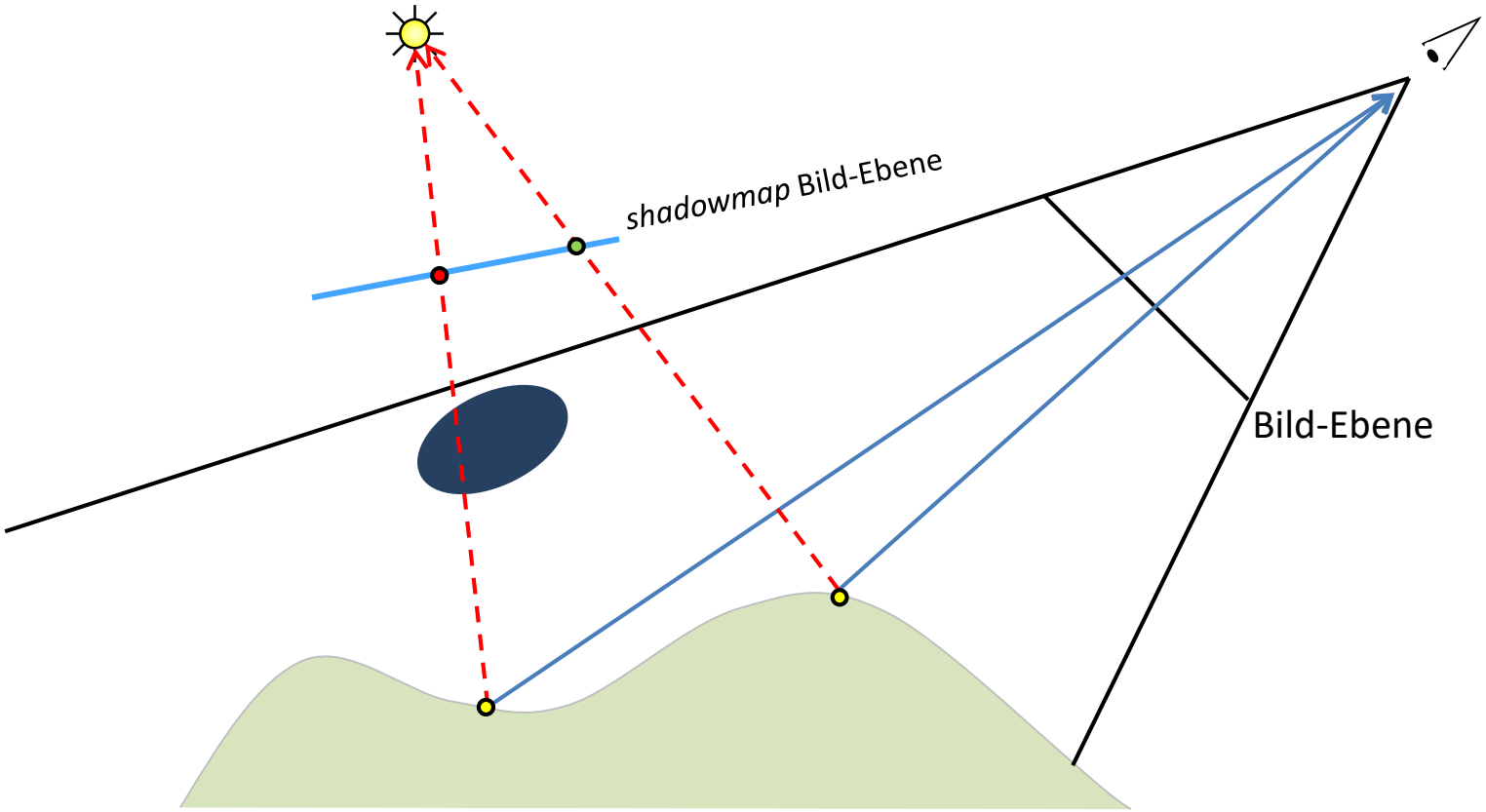
Shadow Maps: Creation





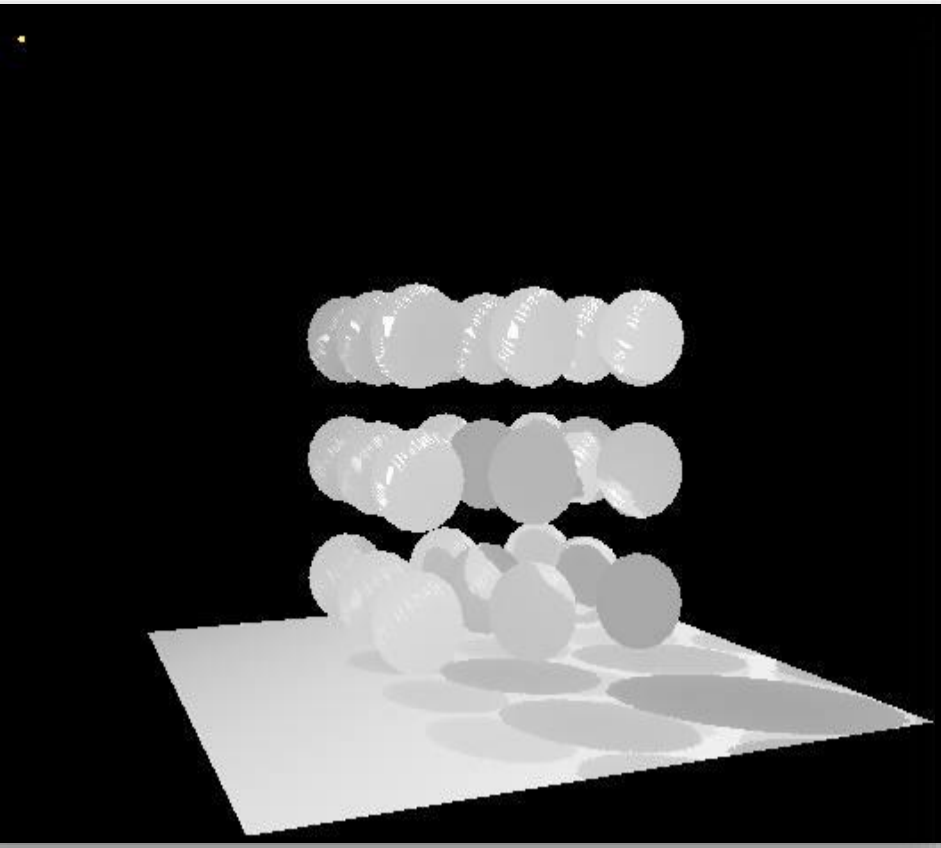
All objects are rendered from the view of the camera. The distance to the closest object is saved in a texture (the depth buffer).

Shadow Maps: Usage

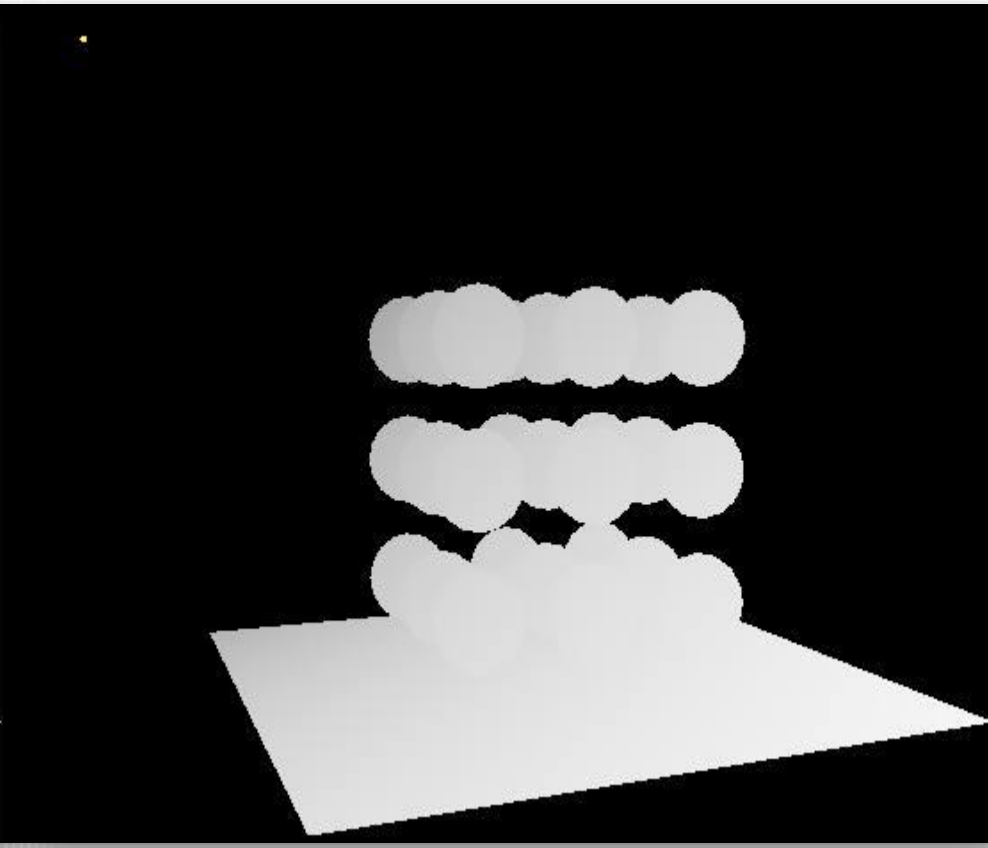


1. Renderer from the camera viewport.
2. For the regular rendering: project positions back into the viewport of the camera and compare the depth values with the values stored in the Shadow Map.

Shadow Maps

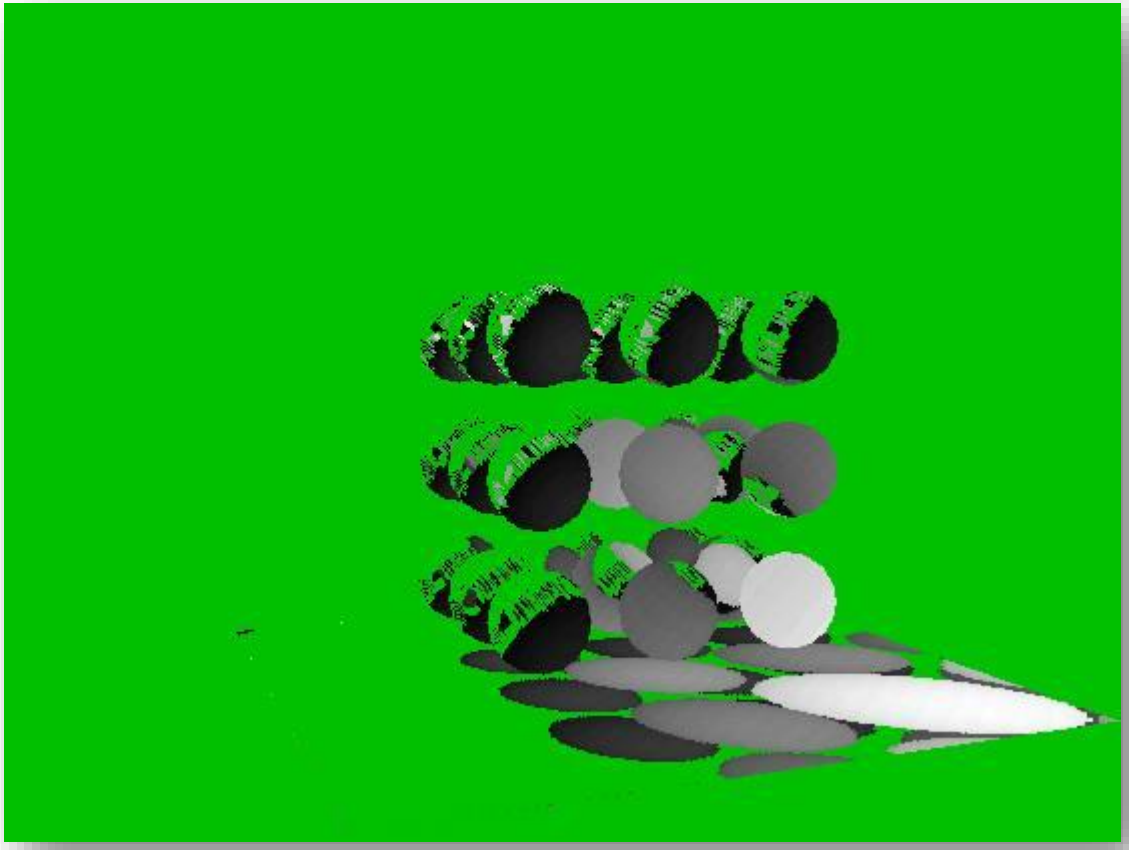


Depth values in the Shadow-Map



Depth values projected into light-space

Shadow Maps

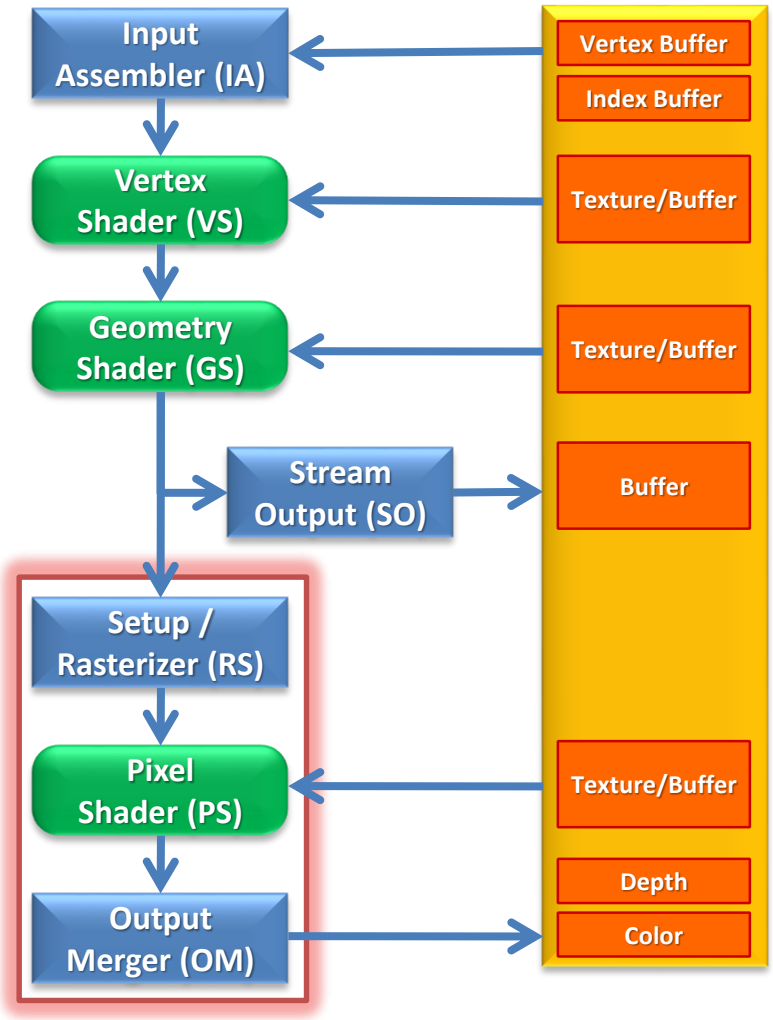


Difference of depth value from the texture / current depth

Green means smaller or equal

Render to texture

- GPUs allow to render into different render targets, instead of just the framebuffer (the screen)
- Up to 8 targets can be used at the same time
- But: only one depth buffer that is used for all render targets



Create render target:

- Create 2D texture
- Set **`BindFlag`** (in `D3D11_TEXTURE2D_DESC`) accordingly:
  - `D3D11_BIND_RENDER_TARGET` allows binding as color target
  - `D3D11_BIND_DEPTH_STENCIL` allows binding as depth-stencil target (requires texture format: `DXGI_FORMAT_D`*, more on that later)
- Additionally, a `RenderTargetView` / `DepthStencilView` is required to bind the texture as render target:
  - `CreateRenderTargetView()`
  - `CreateDepthStencilView()`

Render-Target binding

- The render targets are set with `OMSetRenderTargets()`
  - First save everything with `OMGetRenderTargets()`
  - The color targets as well as the depth-stencil target can be null
  - To create the shadow map we don‘t need color targets!
- Don‘t forget to clear
  - `ClearRenderTargetView`, `ClearDepthStencilView`
  - For best performance, clear the whole target early
  - If you use color+depth, clear both and not only Depth

Viewports/Rasterizer Stage

- You can define a `D3D11_VIEWPORT` per render target:
  - The viewport transforms positions in clip-space (-1…1) to 2D pixel positions in the render target (0..N)
  - With viewports you can only write parts of a render target. This might be useful to save multiple shadow maps in one texture.
  - `ID3D11DeviceContext::RSSetViewports()`
  - Set min-depth to 0, max-depth to 1

HLSL and MRT

- DirectX uses „system values“ to specify, in which render target to write
  - `SV_Target`: color render target (`SV_Target0` .. `SV_Target7`)
  - `SV_Depth`: depth buffer

```
void PSMultipleRTs(float4 Position : SV_Position,
                   float4 ProjectedPosition : PROJPOS,
                   float3 Normal  : NORMAL,
                   float4 FooBar  : WHATEVER,
                   out float3 C1  : SV_Target0,  //output into first target
                   out float4 C2  : SV_Target1,  //output sent to second target
                   out float D0   : SV_Depth )  //custom output to depth buffer
{
    C1 = Normal;
    C2 = FooBar;
    D0 = ProjectedPos.z/ProjectedPos.w; //dehomogenize depthvalue after
    interpolation !!!
}
```

Generate Shadow Maps

- The shadow map stores the smallest depth as seen from the camera
- Hardware support: Define a custom `DepthStencilState` (with Depth-Test-Operation: `LESS`) and use it to render the shadow map
- You used that already (`LESS` is the default depth-test!)
  - Just keep your existing DepthStencilStates

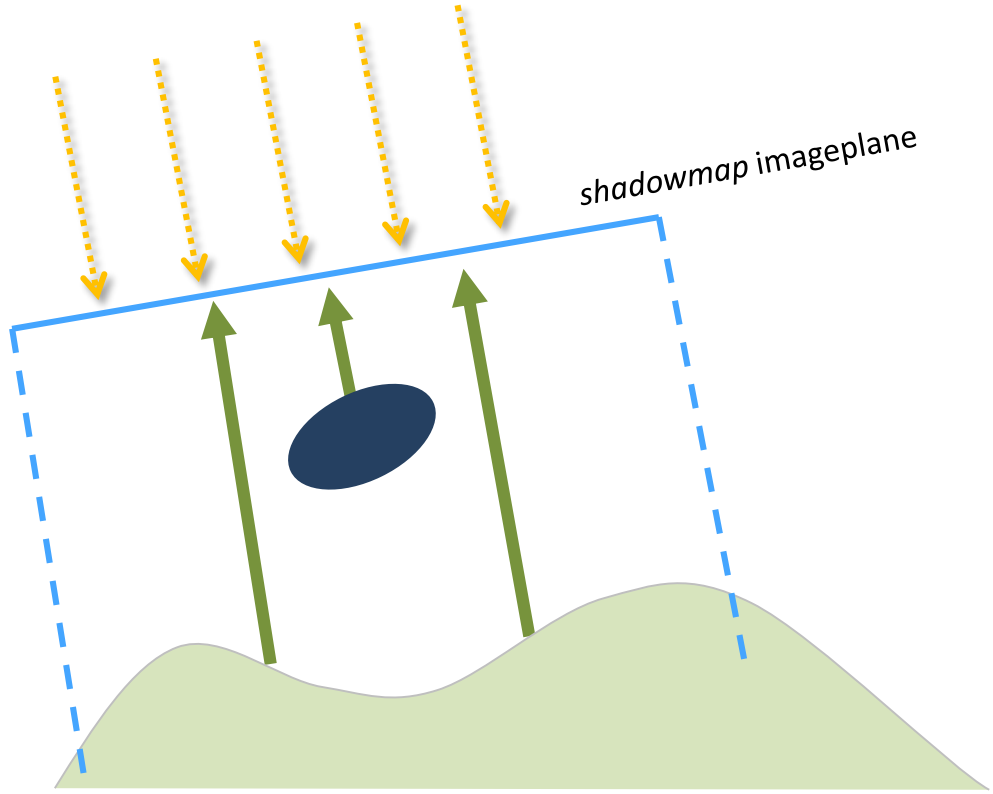
Generate Shadow Maps

- A shadow map stores only depth values. You can use that to improve performance by only generating depth values:
  - The output-merger should only write into the DepthStencil-Buffer, hence set all color targets to NULL
  - Deactivate the pixel-shader, set it to NULL in the effect file

Generate Shadow Maps

- D3D11 allows different “views” of a resource
- Use to format `DXGI_FORMAT_R32_TYPELESS` for the texture
- Set the Bind-Flags in the Texture-Descriptor to `D3D11_BIND_DEPTH_STENCIL | D3D11_BIND_SHADER_RESOURCE`
- DepthStencilView with `DXGI_FORMAT_D32_FLOAT`
- ShaderResourceView with `DXGI_FORMAT_R32_FLOAT`

Shadow Map: Directional light



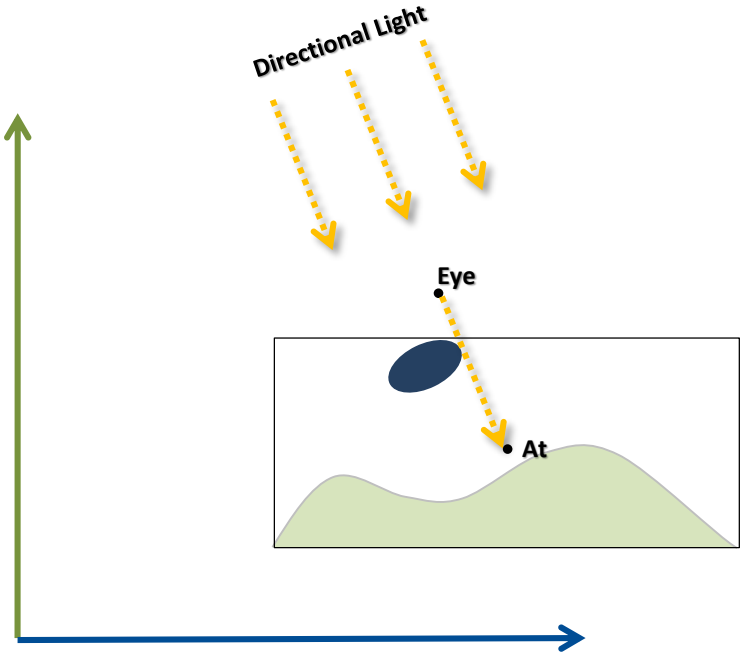
Light rays are parallel → orthographic projection

Shadow Map: Directional light

- Orthographic projection
  - `D3DXMatrixOrthoLH`
  - Use the length of the bounding box diagonal as w, h and zFar
- Additionally: light view matrix
  - The camera (view matrix) must place the world correctly (`D3DXMatrixLookAtLH`)
  - But the look-at function requires an up-vector…

Direction Light: View Transformation

- Set the at-point to the center of the terrain (0,0,0)
- Set the eye-point using the light direction and size of the bounding box
- Use a fixed up-direction (e.g. 0,1,0)
  - Warning: must not be parallel to the light direction!



Use the Shadow-Map

1. Pass the light view-projection-matrix to the shader (World->Light-Space)
2. In the vertex shader: additionally transform the vertices into light space (`SV_Position` stays in camera-space)
3. In the pixel shader: dehomogenize by division through w, then transform the coordinates from NDC (-1…1) into texture space (0..1), invert y-axis
4. Compare depth values with the shadow map
5. Use lighting only if the test was successfull, i.e. the fragment was closer to the light source than the value in the shadow map

Problem: z-Fighting

- Different projections from the different views lead to precision problems

Solution

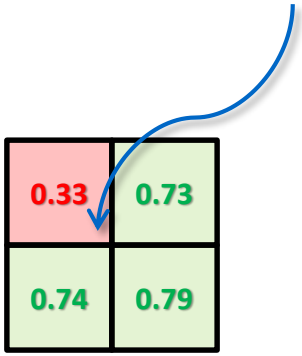
- Add bias: decrase z a bit before the test
  - Better: bias dependent on the distance
  - Or render only the back-faces, then there won‘t be issues at the front faces. And on the back face you shouldn‘t need to test for shadows as it can‘t get lit either way
- Render only objects that actually cast shadows

Problem: Aliasing

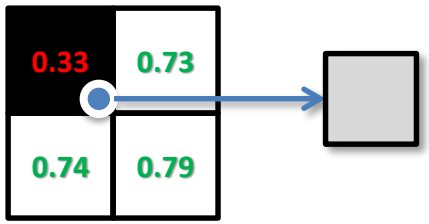
Percentage closer filtering

- Sample neighboring pixels in the shadow map, e.g. 2x2
- Filtering of the shadow test results (not of the depth!)
  - Hardware support: interpolation of the result

    Example: Fragment-Depth 0.49



Value in the shadow map is smaller than the projected depth: fragment is shadowed



Interpolation of the resulting black-and-white texture at the sample position

Percentage closer filtering

- Supported in hardware
  - HLSL: create SamplerComparisonState
    - Ste filter to `COMPARISON_*`
    - Set ComparisonFunc to LESS
  - Sampling: `SampleCmpLevelZero`
    - Rreturns the „fraction of shadow“
  - Hardware performs four comparisons automatically
    - No performance difference to a single fetch!

Bonus Assignment - Summary

- Add shadow mapping for the sun
  - Render in a depth-only render target
    - Calculate the bounding box of the terrain, adjust projection
  - Shadow test for all objects and the terrain
  - Hardware Percentage Closer Filtering

Advanced shadowmap techniques

- „ShadowMap“ Example in the DXSDK Sample Browser (DX9)
- PCSS: Percentage Closer Soft Shadows
  - http://news.developer.nvidia.com/2008/02/integrating-rea.html
- TSM: Trapezoidal Shadow Maps (PSM, LiPSM)
  - http://www.comp.nus.edu.sg/~tants/tsm.html
  - http://www.comp.nus.edu.sg/~tants/tsm/TSM_recipe.html
- CSM: Cascaded Shadow Maps
  - http://developer.download.nvidia.com/SDK/10.5/opengl/src/cascaded_shadow_maps/doc/cascaded_shadow_maps.pdf

Questions?