Praktikum: Echtzeit Computergrafik



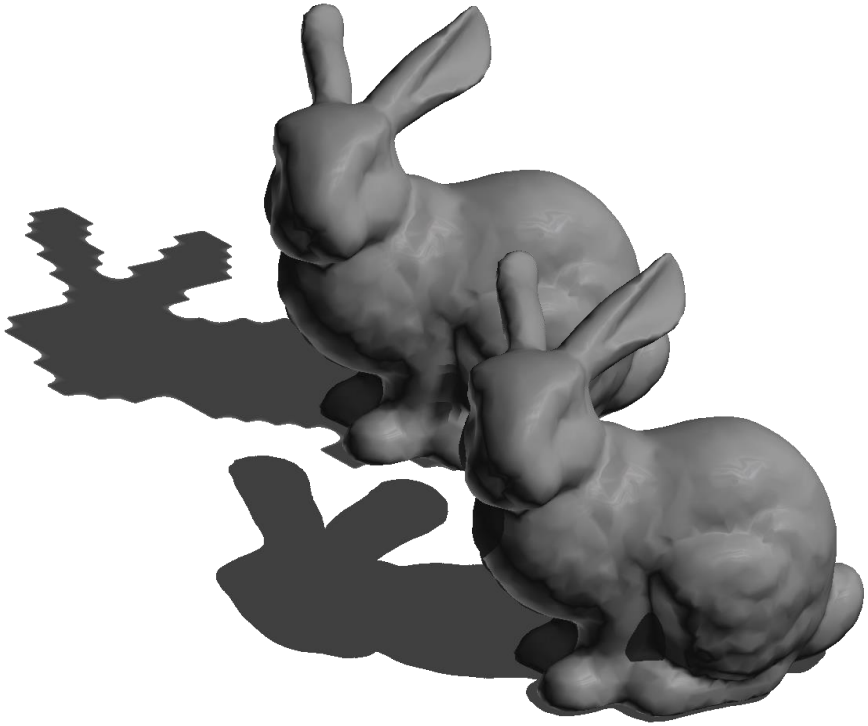
tum.3D

computer graphics & visualization

Bonus Assignment

Shadow mapping

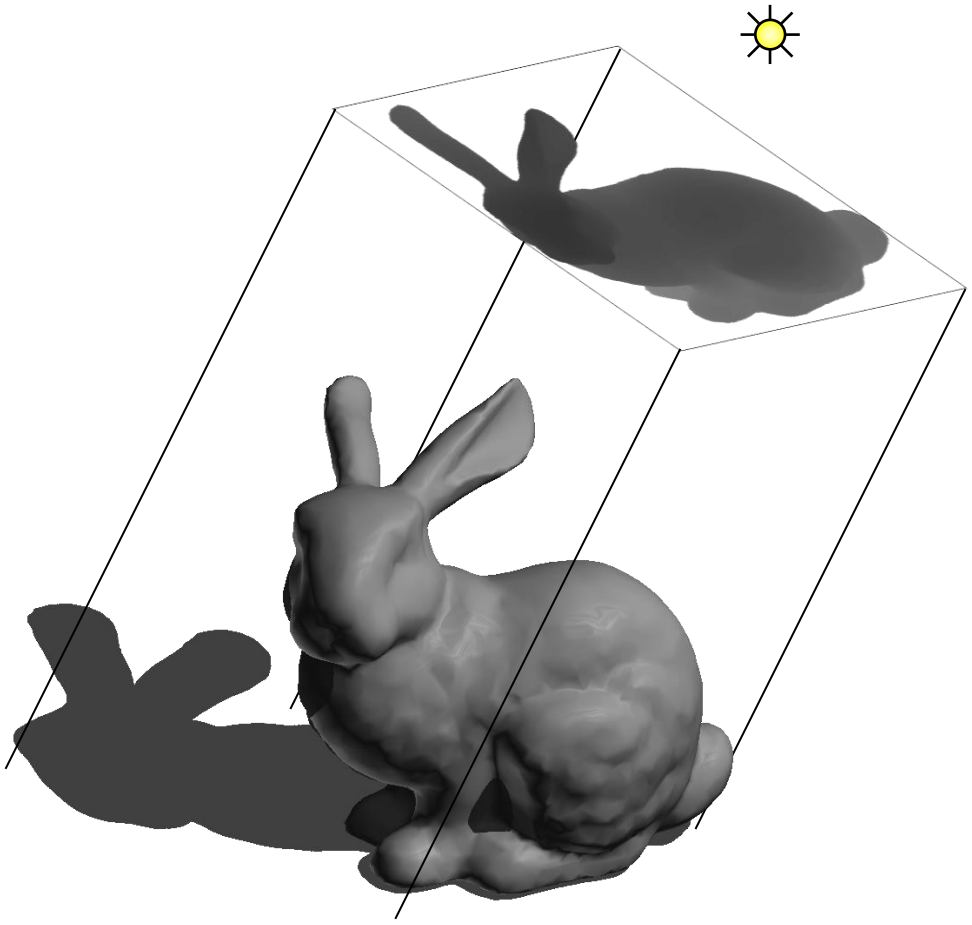
Methods



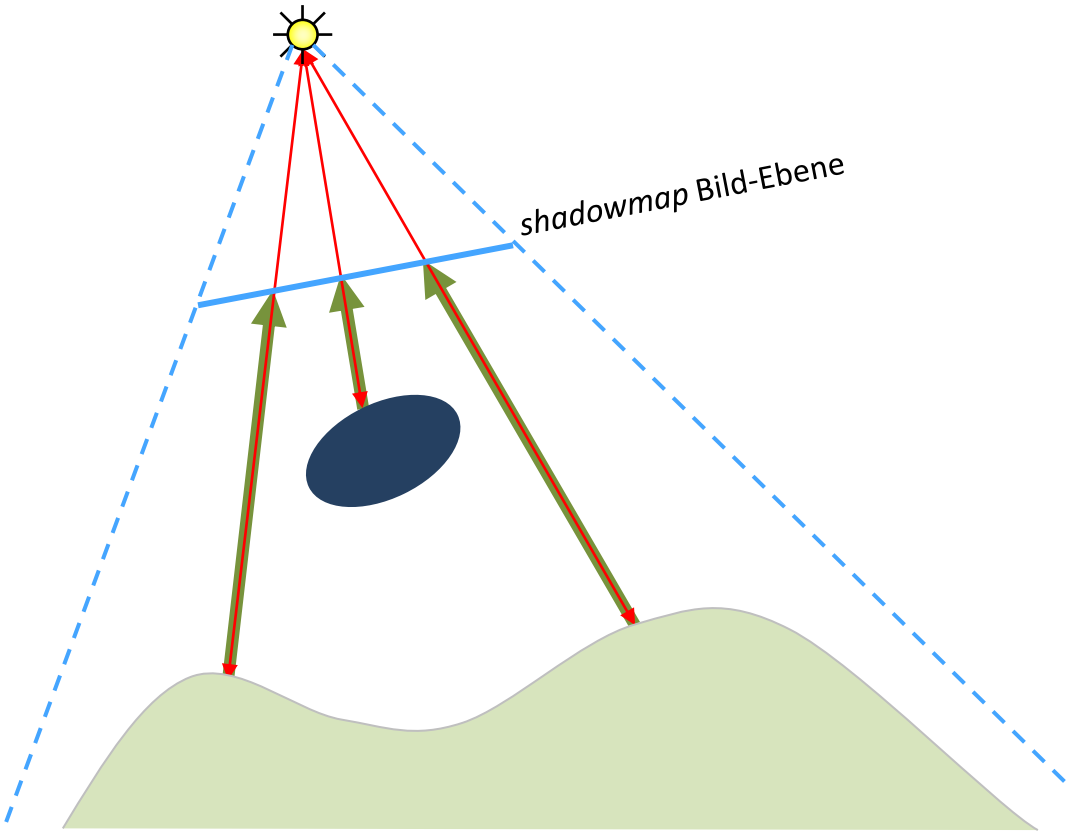
Shadow Map

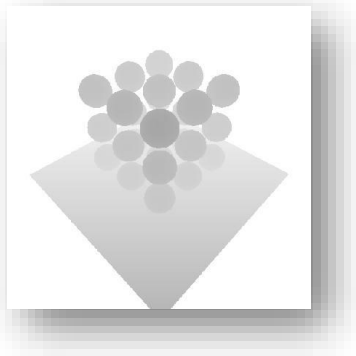
Shadow Volume

Shadow Map



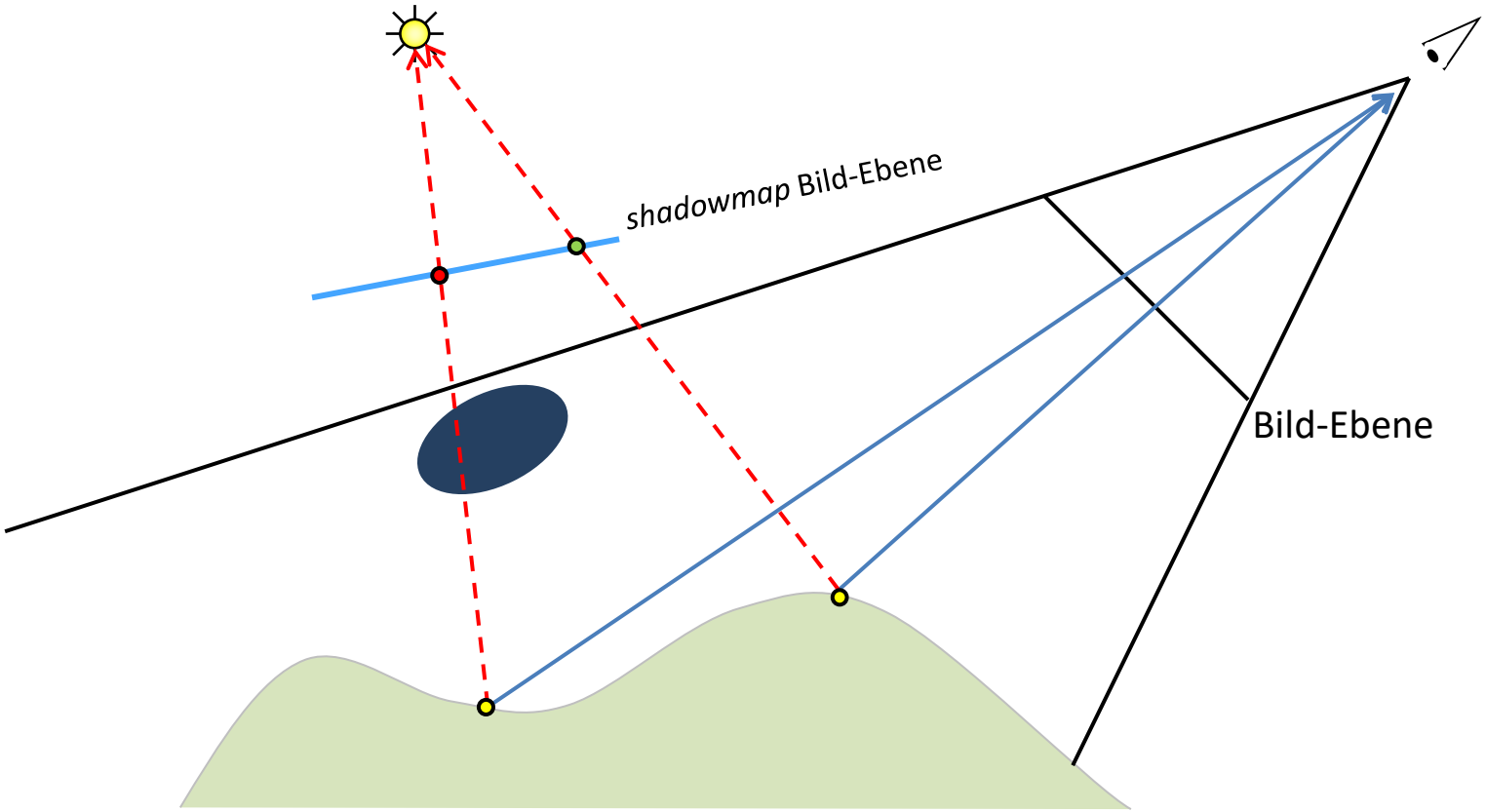
Shadow Maps: Erstellung





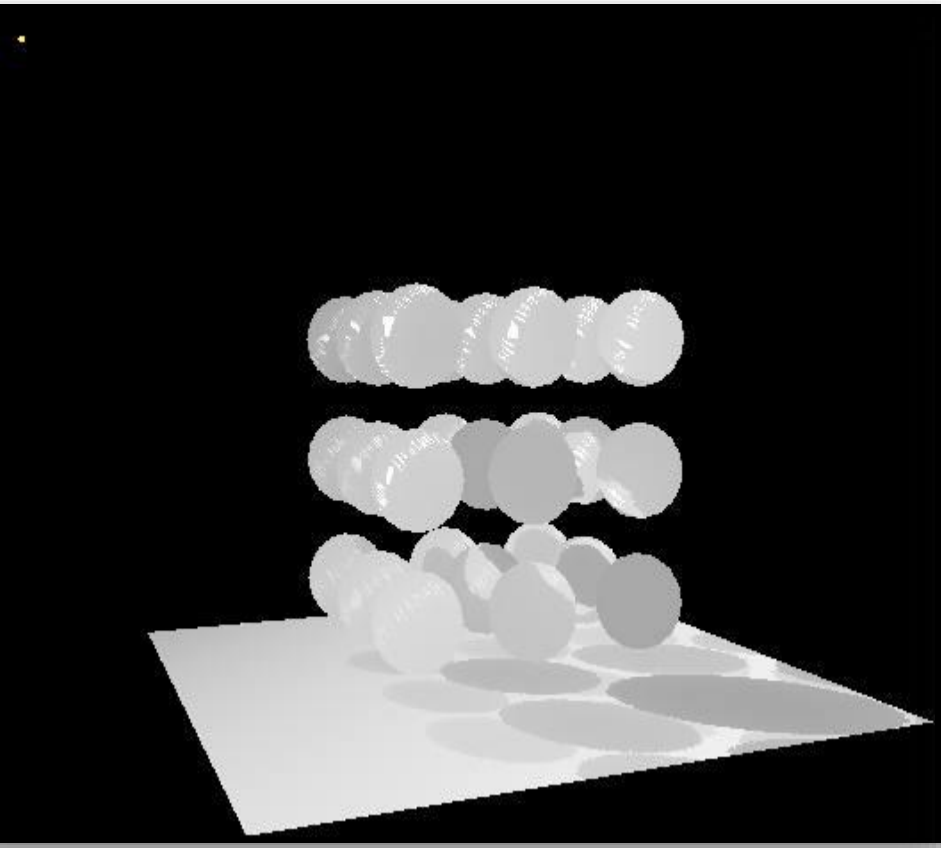
Alle Objekte aus Sicht der Kamera Rendern, und den Abstand zum nähesten Objekt in einer Textur speichern

Shadow Maps: Nutzung

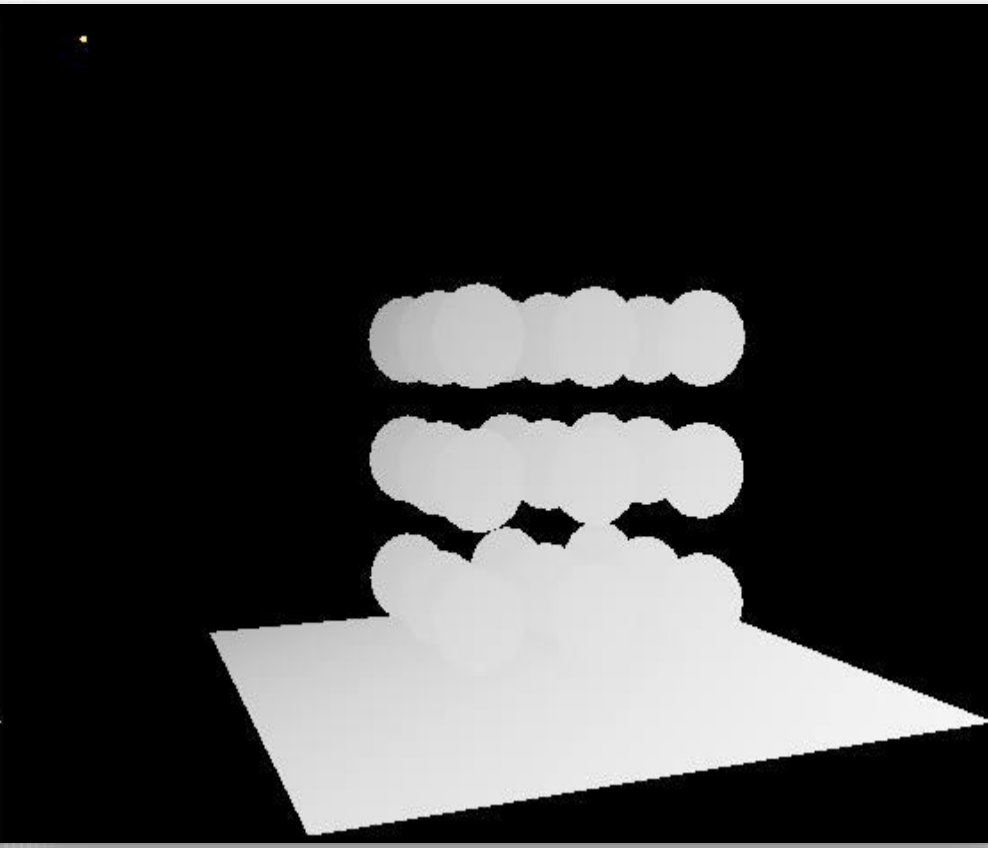


1. Rendern aus Sicht der Kamera
2. Positionen zurück in die Ansicht der Lichtquelle projizieren und die Tiefenwerte mit den in der Shadow Map gespeicherten vergleichen

Shadow Maps

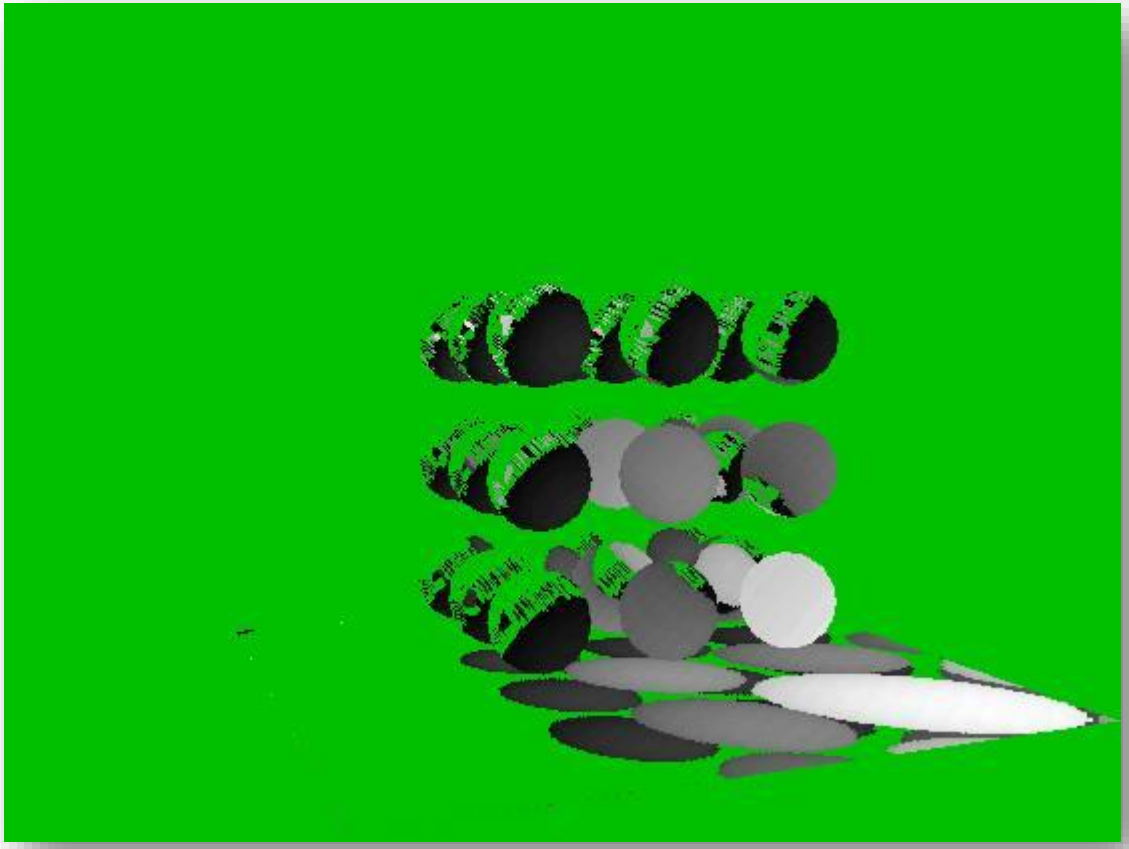


Tiefenwerte in der Shadow-Map



Tiefenwerte projiziert in den Light-Space

Shadow Maps

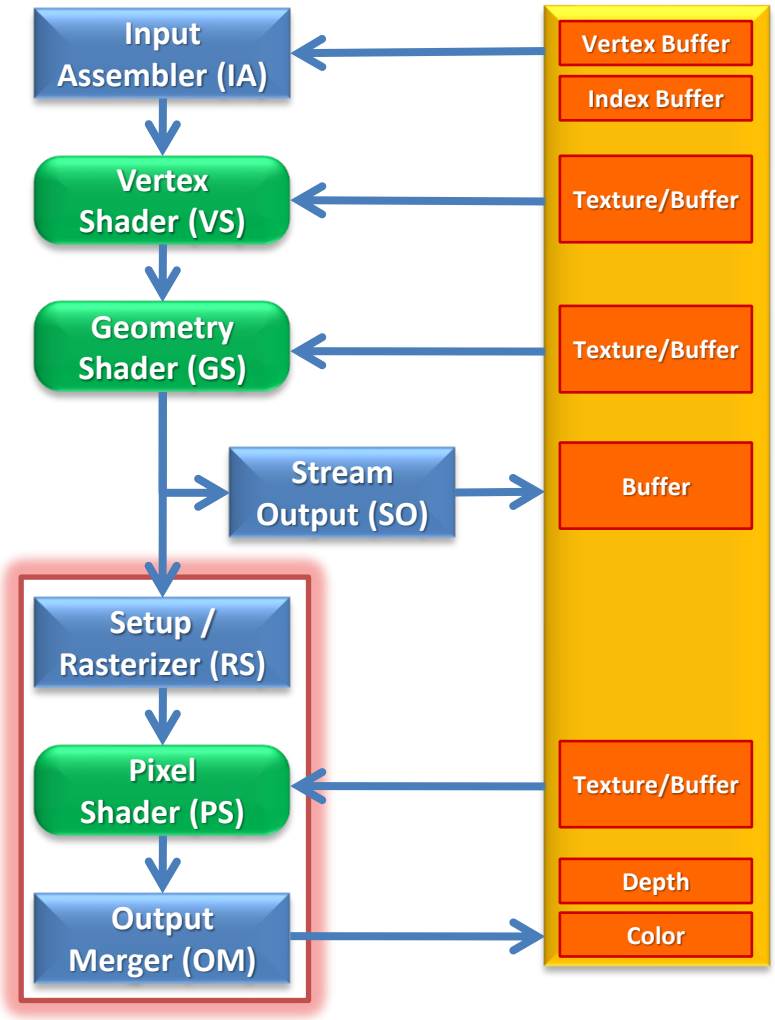


Differenz Tiefenwerte/Aktuelle Werte

Grün entspricht kleiner oder gleich

Rendern in eine Textur

- Grafikkarten erlauben es, in verschiedene Render-Targets zu rendern (außer dem Framebuffer/Bildschirm)
- Bis zu 8 Targets können gleichzeitig benutzt werden
- Allerdings: Nur ein Tiefen-Buffer, der für alle Render-Targets verwendet wird



Render-Target erstellen:

- 2D-Textur erstellen
- **`BindFlag`** (in der `D3D11_TEXTURE2D_DESC`) entsprechend setzen:
  - `D3D11_BIND_RENDER_TARGET` ermöglicht das Binden als Color-Render-Target.
  - `D3D11_BIND_DEPTH_STENCIL` ermöglicht das Binden als Depth-Stencil-Target (benötigt ein Format der Art: `DXGI_FORMAT_D*`, dazu später mehr)
- Zusätzlich ist eine `RenderTargetView` / `DepthStencilView` notwendig, um die Textur auch tatsächlich als Render-Target binden zu können:
  - `CreateRenderTargetView()`
  - `CreateDepthStencilView()`

Render-Target binden

- Mittels `OMSetRenderTargets()` können die Render-Targets gesetzt werden
  - Vorher alles mit `OMGetRenderTargets()` sichern
  - Sowohl die Color-Targets als auch das Depth-Stencil Target kann null sein
  - Beim Erstellen der Shadow Map brauchen wir keine Color-Targets!
- Clear nicht vergessen
  - `ClearRenderTargetView`, `ClearDepthStencilView`
  - Für beste Performance frühzeitig das gesamte Target leeren
  - Wenn man Color+Depth nutzt, sollte man auch beide leeren und nicht nur Depth

Viewports/Rasterizer Stage

- Pro Render-Target kann ein `D3D11_VIEWPORT` definiert werden:
  - Der Viewport transformiert Positionen im Clip-Space (-1..1) auf 2D Pixel Positionen innerhalb des Render-Targets (0..N)
  - Mit Viewports kann man z.B. in Teile eines Render-Targets schreiben (nützlich um mehrere Shadow-Maps in einer Textur zu speichern)
  - `ID3D11DeviceContext::RSSetViewports()`
  - Min-Depth auf 0 und Max-Depth auf 1 setzen!

HLSL und MRT

- DirectX nutzt „System Values“, um festzulegen, in welches Render Target geschrieben werden soll
  - `SV_Target`: Farb-Render-Target (`SV_Target0` .. `SV_Target7`)
  - `SV_Depth`: Tiefen-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 !!!
}
```

Generieren von Shadow Maps

- Die Shadow Map speichert die kleinste Tiefe aus der Sicht der Kamera
- Hardware-Unterstützung: Ein eigenen `DepthStencilState` (mit Depth-Test-Operation: `LESS`) definieren und zum Rendern der Shadow Map benutzen
- Bisher auch schon immer benutzt (`LESS` ist der Standard-Test!)
  - Übernehmt eure bestehenden DepthStencilStates

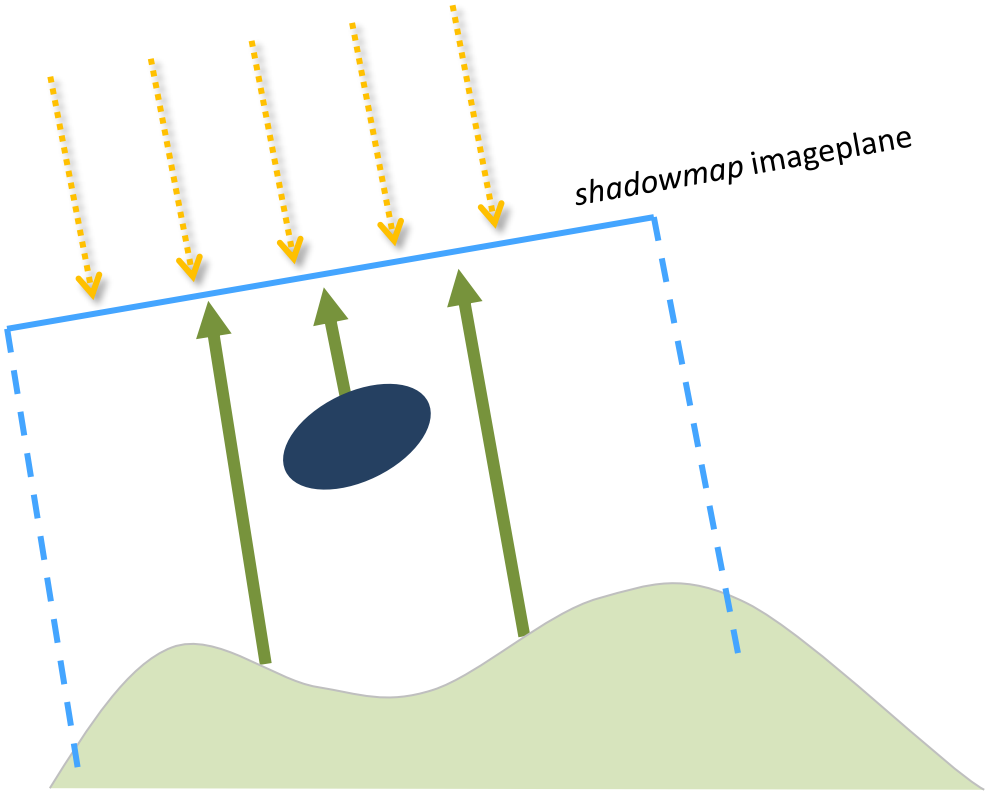
Generieren von Shadow Maps

- Eine Shadow-Map speichert **nur** Tiefen-Werte. Das kann man ausnutzen, indem man den State so aufsetzt, dass nur Tiefe generiert wird:
  - Der Output-Merger sollte nur in einen DepthStencil-Buffer schreiben; d.h. die Color-Targets auf `nullptr` setzen
  - Pixel-Shader deaktivieren (indem man ihn auf NULL setzt)

Generieren von Shadow Maps

- D3D11 erlaubt unterschiedliche “Views” auf eine Ressource
- Bei der Textur als Format `DXGI_FORMAT_R32_TYPELESS` angeben
- Im Texture-Descriptor die Bind-Flags auf `D3D11_BIND_DEPTH_STENCIL | D3D11_BIND_SHADER_RESOURCE` setzen
- DepthStencilView mit `DXGI_FORMAT_D32_FLOAT`
- ShaderResourceView mit `DXGI_FORMAT_R32_FLOAT`

Shadow Map: Directional light



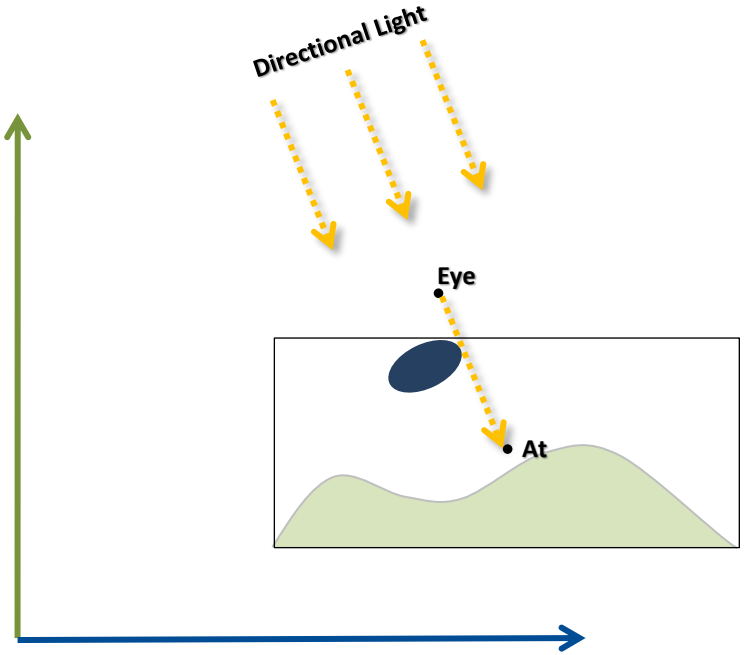
Lichtstrahlen sind parallel → Orthographische Projektion

Shadow Map: Directional light

- Orthographische Projektion
  - `D3DXMatrixOrthoLH`
  - Länge der Bounding Box Diagonale für w, h und zFar verwenden
- Zusätzlich: Light view matrix
  - Kamera (View-Matrix) muss die Welt korrekt platzieren (`D3DXMatrixLookAtLH`)
  - Allerdings erwartet die Look-At Matrix einen up-Vektor …

Direction Light: View Transformation

- At-Punkt auf die Mitte des Terrains (0,0,0) festsetzen
- Eye-Punkt über Lichtrichtung und Größe der Bounding Box (sicher: halbe Länge der Diagonale) berechnen
- Feste Up-Richtung (z.B. 0,1,0) verwenden
  - Achtung: Darf nicht parallel zur Lichtrichtung sein!



Shadow-Map verwenden

1. Light View-Projection-Matrix an den Shader übergeben (World->Light-Space)
2. Im Vertex-Shader: Vertices zusätzlich in den Light-Space transformieren (`SV_Position` bleibt aber im Camera-Space)
3. Im Pixel-Shader: Dehomogenisierung über Division durch `w`, danach die Koordinaten vom NDC (-1..1) in den Texture-Space transformieren (0..1), y-Achse umdrehen
4. Tiefen-Werte mit denen in der Shadow-Map vergleichen
5. Beleuchtung nur dann ausführen, wenn der Test erfolgreich war (d.h. das Fragment war näher an der Lichtquelle als der Wert in der Shadow-Map)

Problem: z-Fighting

- Verschiedene Projektionen in den verschiedenen Ansichten führen zu Präzisionsproblemen

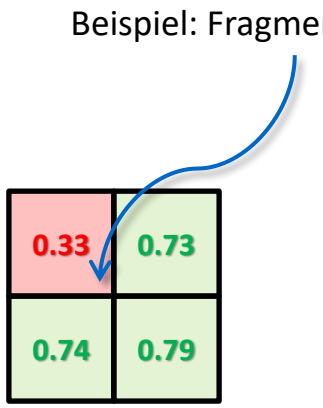
Lösung

- Bias hinzufügen: z ein wenig verkleinern vor dem Test
  - Oder ein Epsilon hinzufügen
  - Besser: Epsilon/Bias in Abhängigkeit von der Entfernung
  - Oder Back-Faces rendern, damit gibt es auf der Vorderseite keine Probleme mehr, und auf der Rückseite sollte i.d.R. kein Schatten mehr getestet werden da die Rückseite nicht beleuchtet sein kann
- Nur Objekte rendern, die auch tatsächlich Schatten werfen können

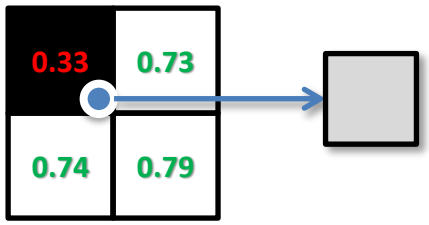
Problem: Aliasing / Kanten

Percentage closer filtering

- Samplen der Nachbarpixel in der Shadowmap (z.B. 2x2)
- Filtern des Shadow-Test-Ergebnisses (nicht der Tiefe!)
  - Hardwareunterstützung: Interpolation des Ergebnisses



Wert in der Shadow Map ist kleiner als die projizierte Tiefe: Fragment ist im Schatten



Interpolation einer resultierenden schwarz / weiß Texture an der Sample-Position

Percentage closer filtering

- In HW unterstützt
  - HLSL: SamplerComparisonState anlegen
    - Filter auf `COMPARISON_*` setzen
    - ComparisonFunc auf `LESS` setzen
  - Samplen: `SampleCmpLevelZero`
    - Liefert „Verschattungsfaktor“
  - HW führt automatisch 4 Vergleiche durch
    - Kein Performance-Unterschied zu einem einzelnen Fetch!

Assignment 10

- Shadow Mapping für die Sonne hinzufügen
  - Rendern in ein Depth-Only Render-Target
    - Bounding-Box des Terrains ausrechnen, Projektion anpassen
  - Schattentest für alle Objekte und das Terrain
  - Hardware Percentage Closer Filtering

Advanced shadowmap techniques

- „ShadowMap“ Example im 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?