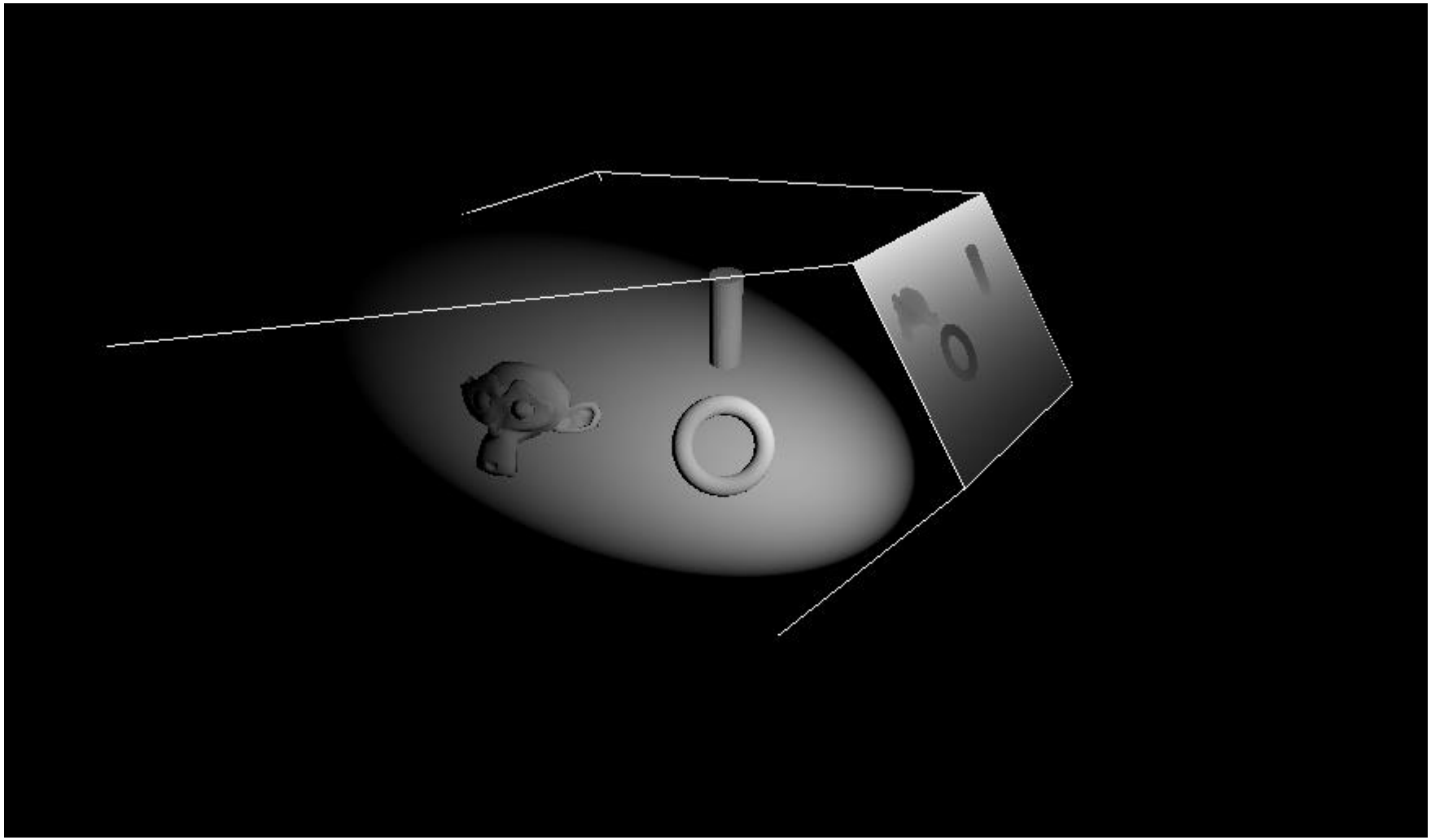
Computergraphik

Übung: Realtime Rendering

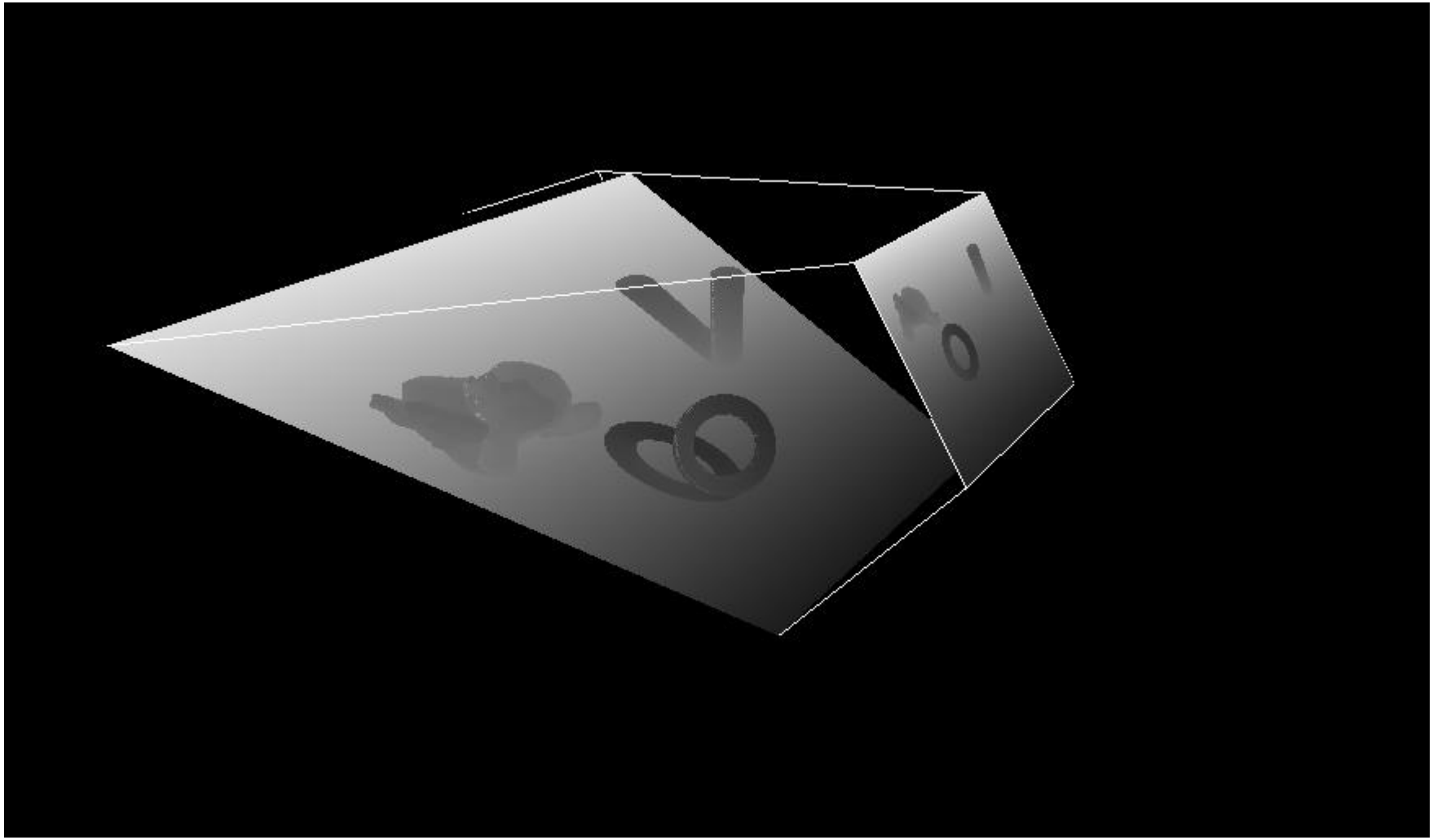
Prof. Dr.-Ing. Carsten Dachsbacher
Lehrstuhl für Computergrafik
Karlsruher Institut für Technologie

Shadow Mapping



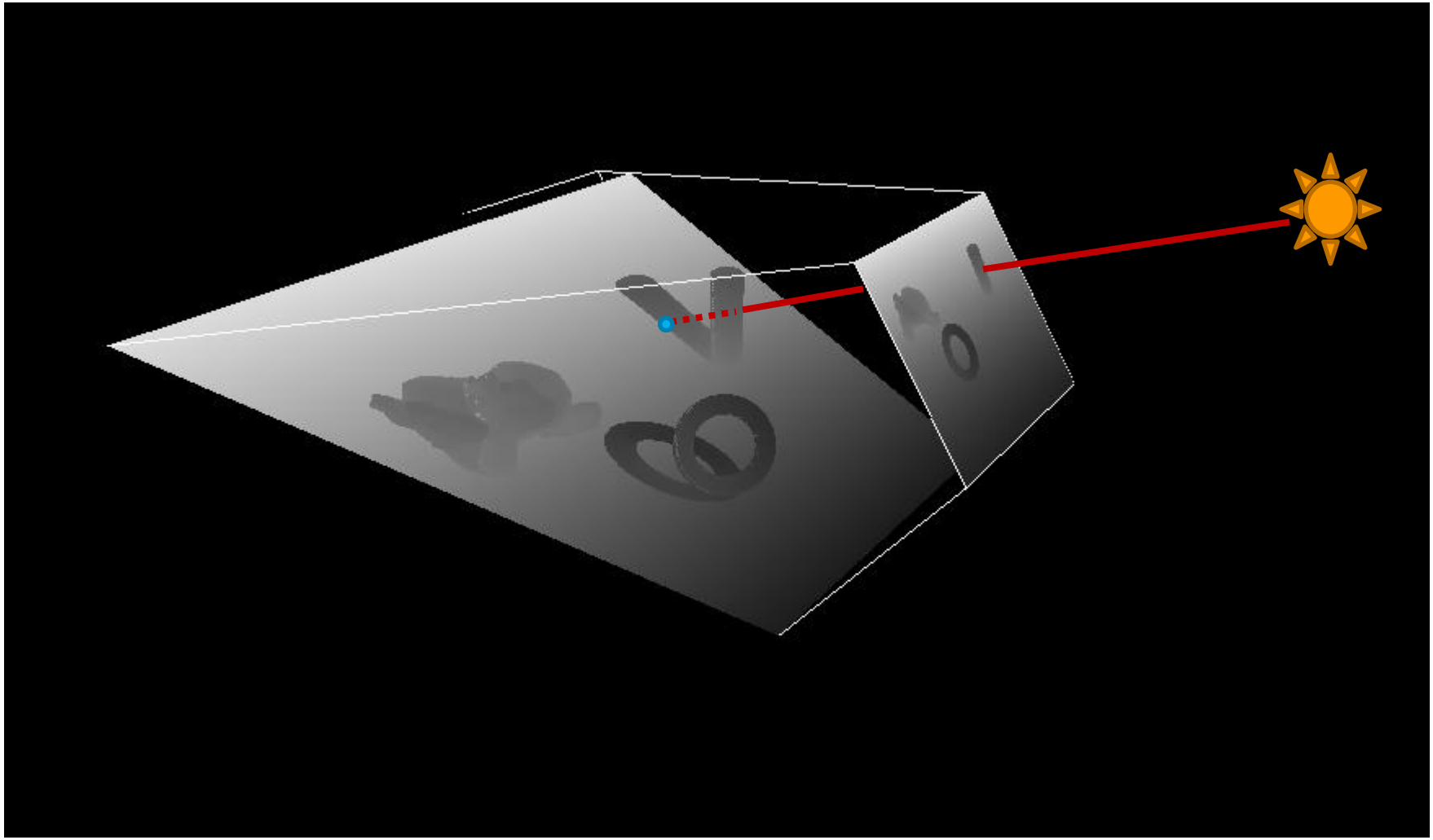
Kameraansicht ohne Schatten

Shadow Mapping



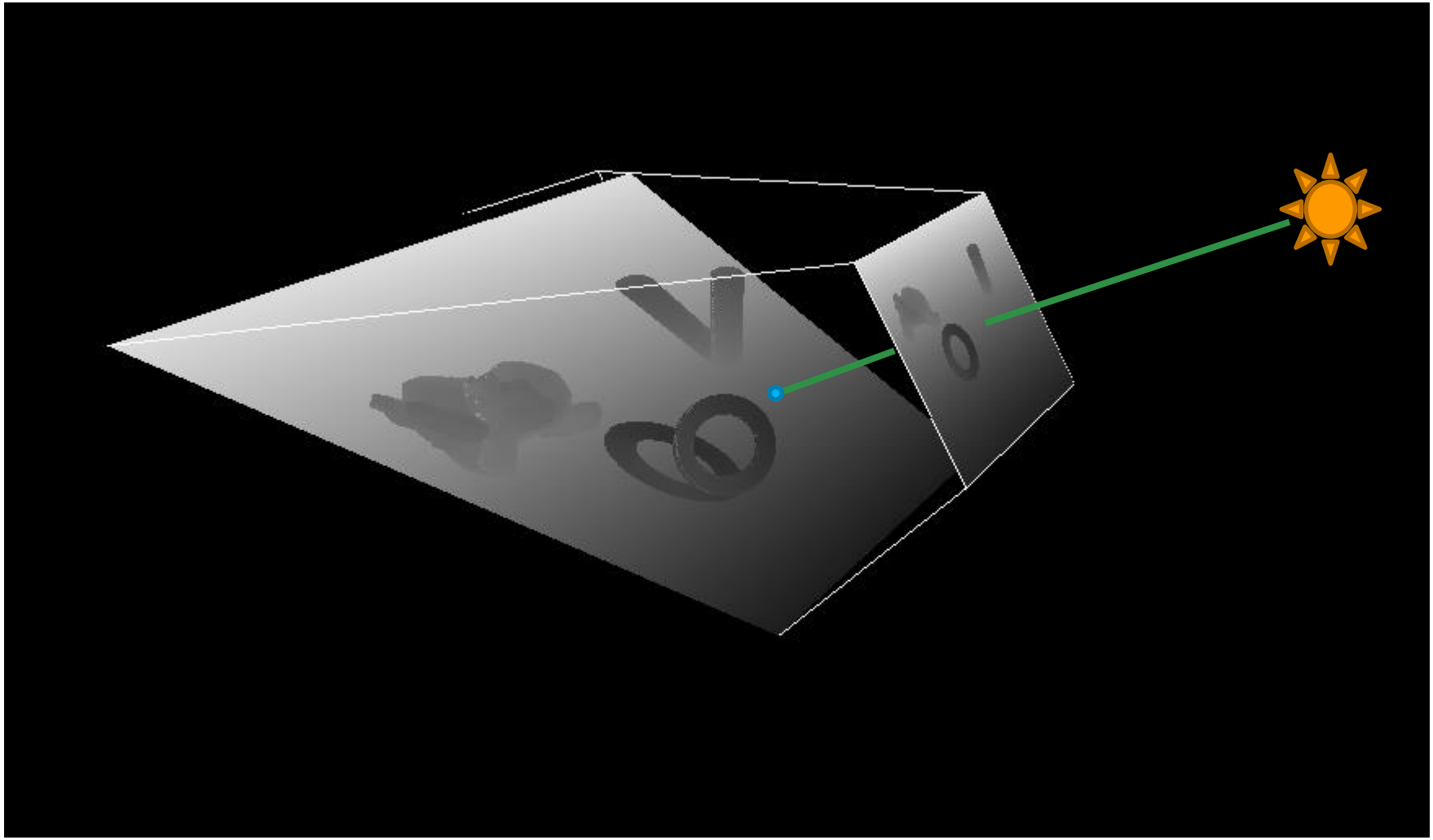
Tiefenpuffer projiziert auf Szene

Shadow Mapping



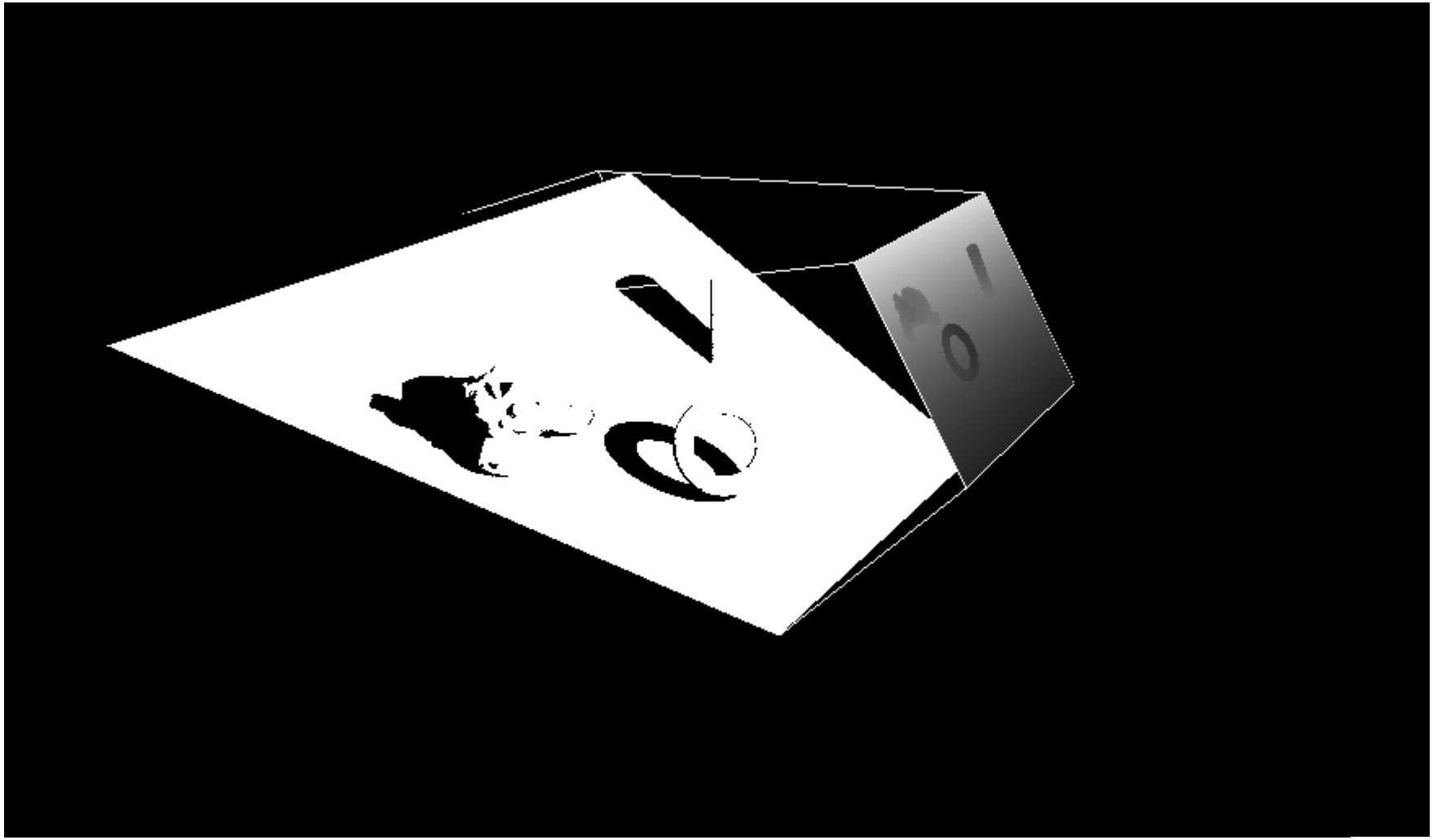
Tiefenpuffer projiziert auf Szene

Shadow Mapping



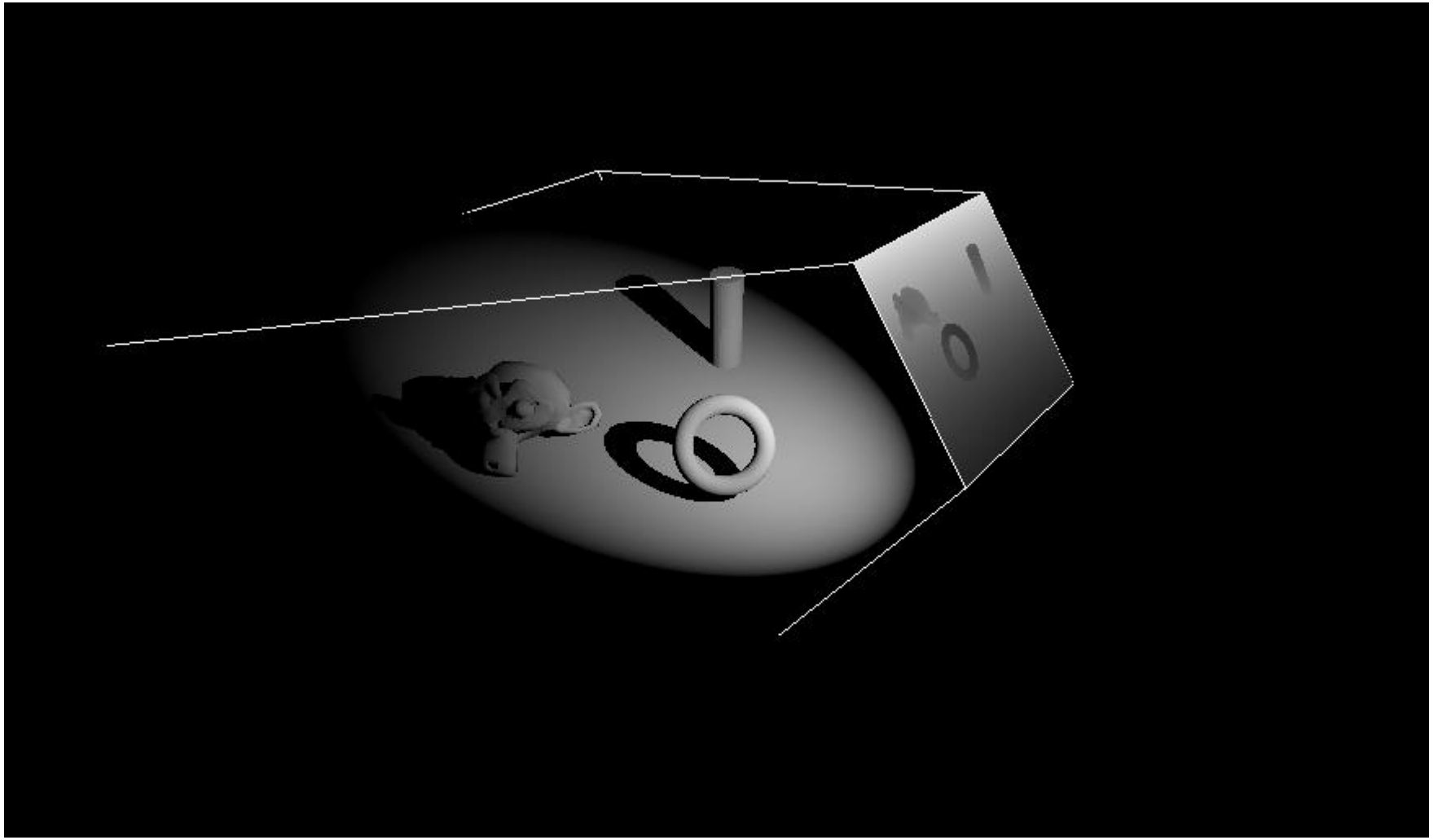
Tiefenpuffer projiziert auf Szene

Shadow Mapping



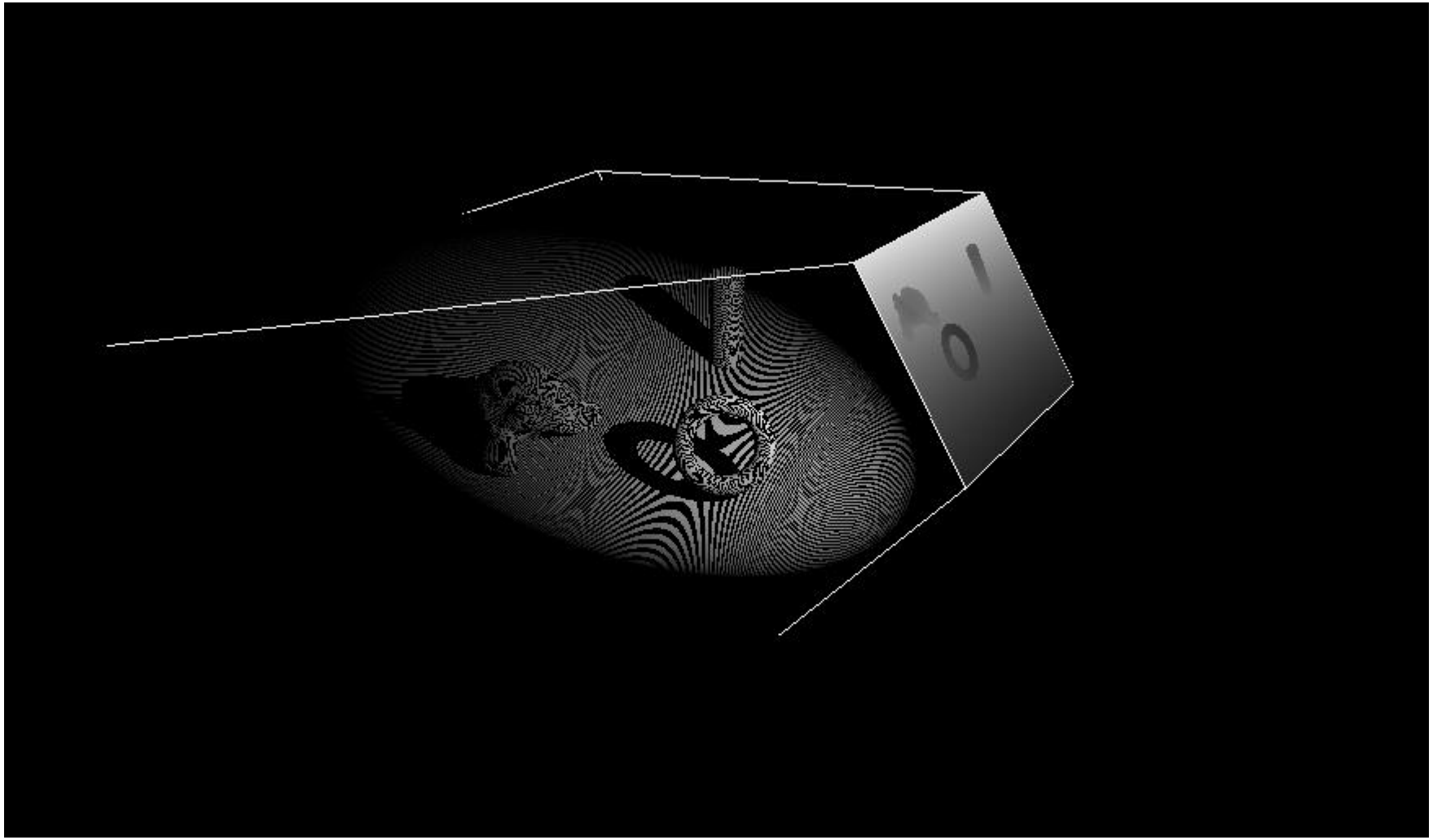
Resultat des Tiefenvergleichs (Shadow Mask)

Shadow Mapping



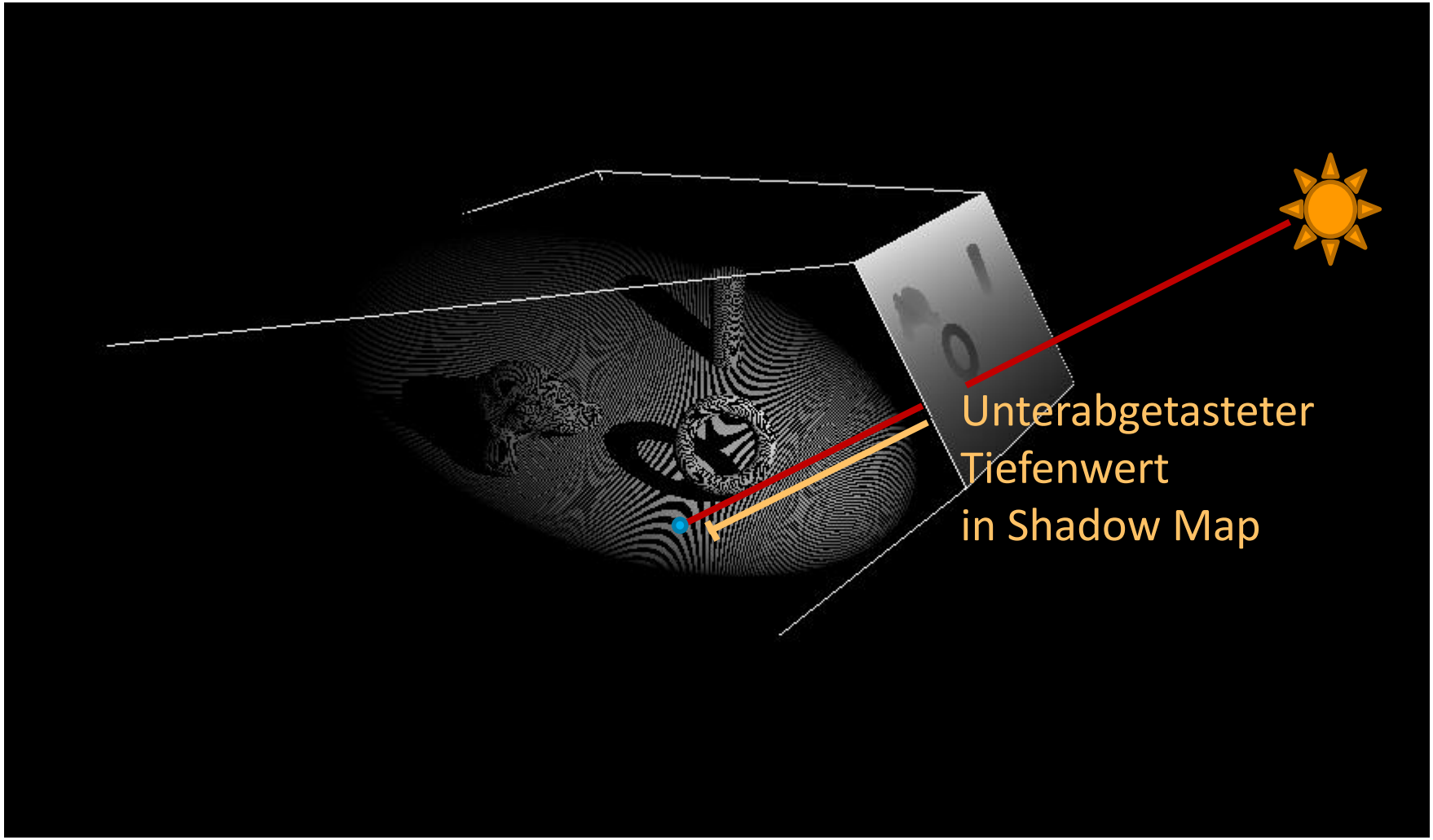
Kameraansicht mit Schatten (und Shading)

Shadow Bias



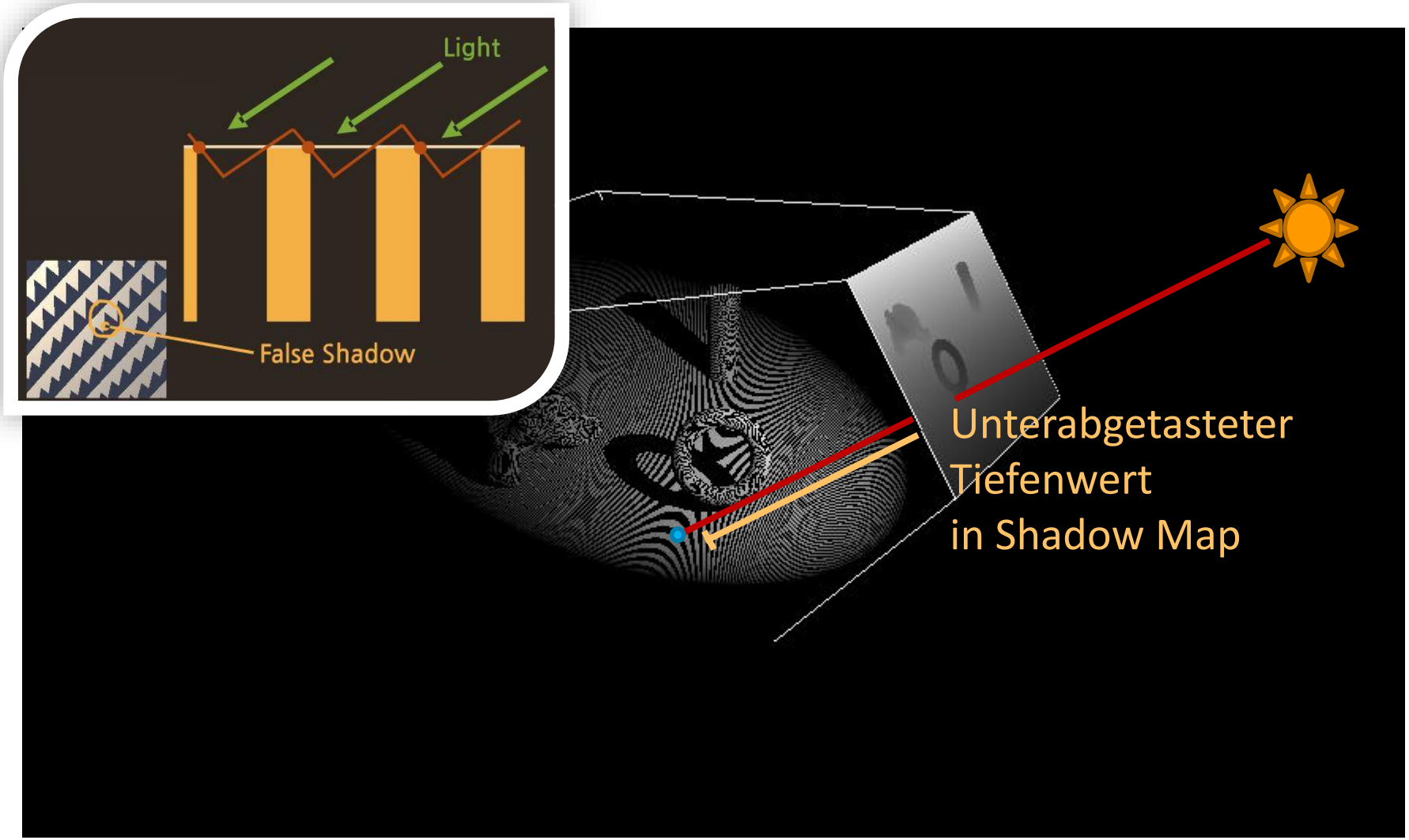
Kameraansicht mit Schatten, ohne Shadow Bias

Shadow Bias



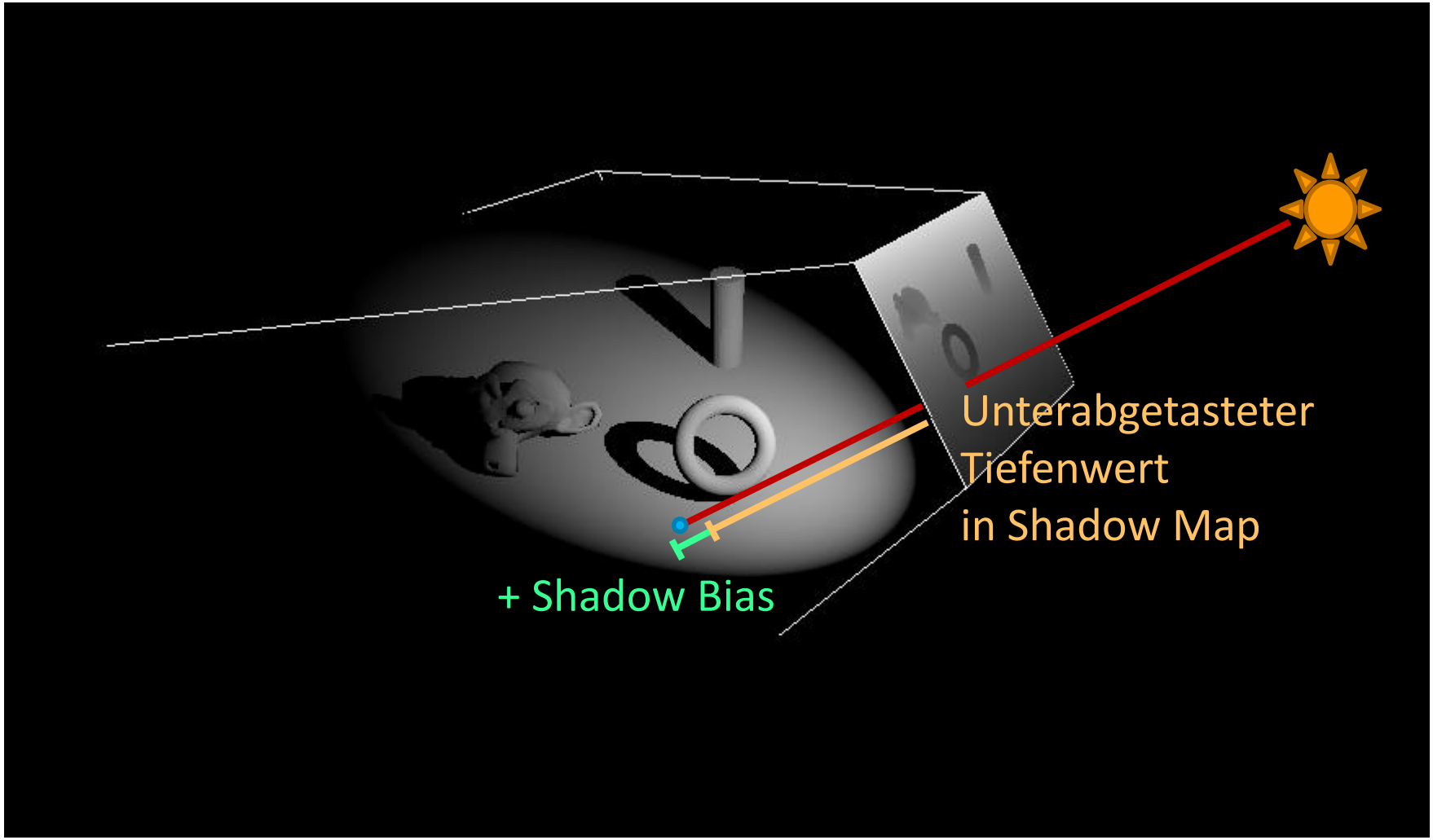
Kameraansicht mit Schatten, ohne Shadow Bias

Shadow Bias



Kameraansicht mit Schatten, ohne Shadow Bias

Shadow Bias

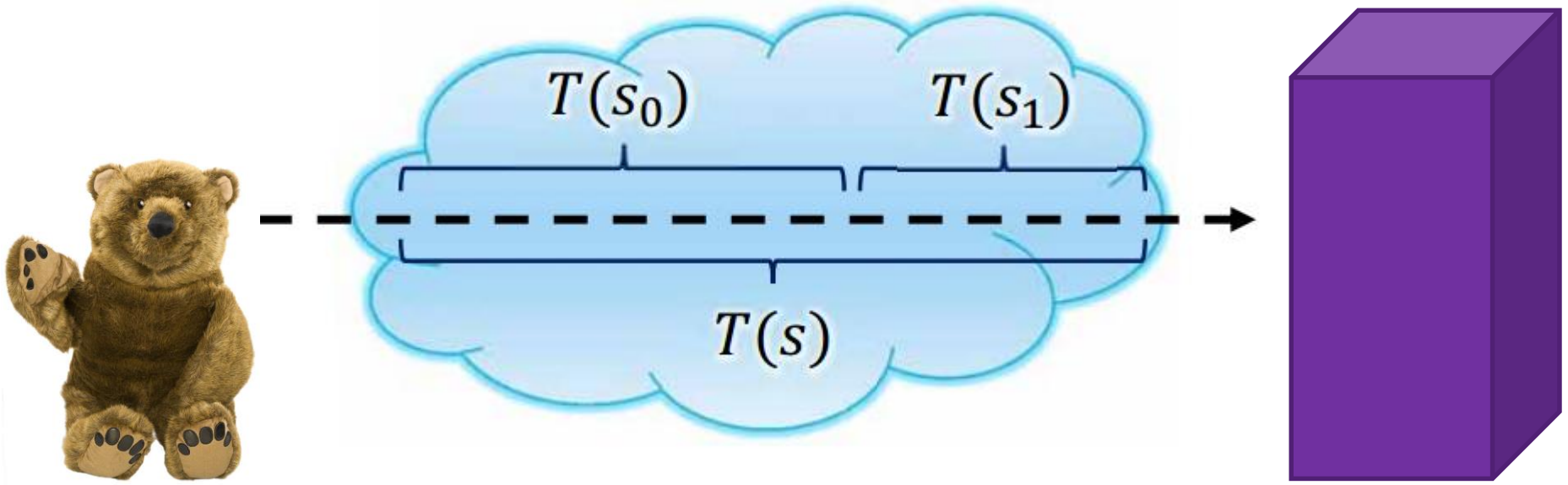


Kameraansicht mit Schatten und Shadow Bias

Absorptionsmodell

- Beersches Gesetz
  - Transmittance = $T(s) = e^{-\sigma_t s}$
  - σ_t Abschwächungskoeffizent, s Weglänge
  - σ_t wellenlängenabhängig (r, g, b)

- Transmittance verhält sich multiplikativ
  - $T(s) = T(s_0) \cdot T(s_1)$



Blending

- Draw Calls zeichnen die Szene in Schichten

- Blending ermöglicht Kombination von alter und neuer Farbe

Alpha Blending

Source Color (Fragmentfarbe)

Destination Color (Framebuffer bevor)

Source Alpha (Fragment Alpha)

Final Color (Framebuffer danach)

Final Color = Source Alpha * Source Color + (1 – Source Alpha) * Destination Color

Alpha Blending

Source Color (Fragmentfarbe)

Destination Color (Framebuffer bevor)

Source Alpha (Fragment Alpha)

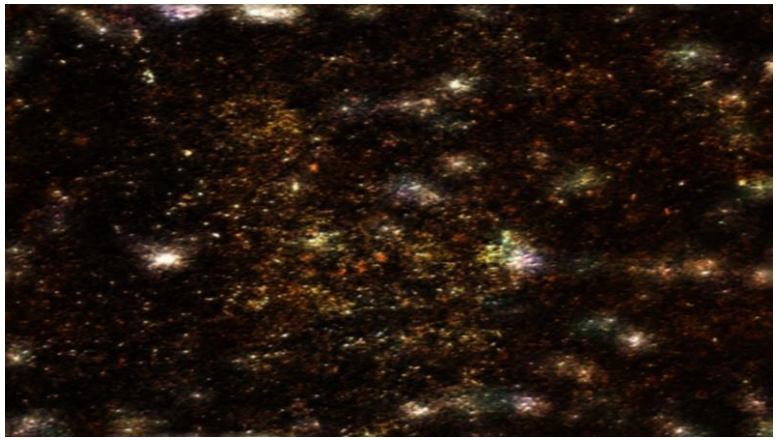
Final Color (Framebuffer danach)

```
glEnable(GL_BLEND);
glBlendEquation(GL_FUNC_ADD);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)
```

Final Color = Source Alpha * Source Color + (1 – Source Alpha) * Destination Color

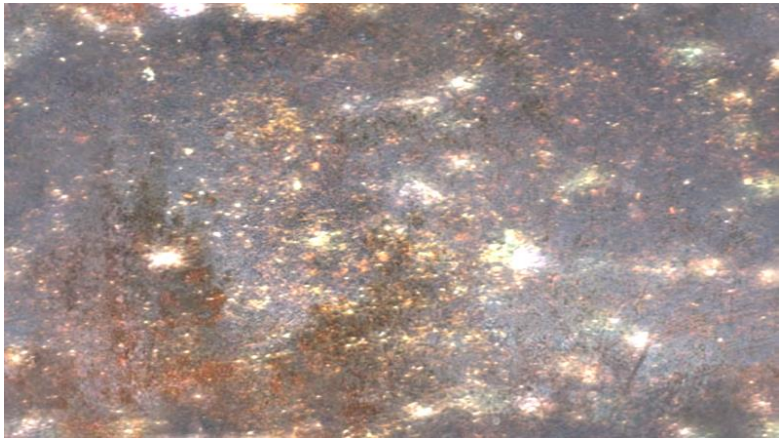
Additives Blending

Source Color (Fragmentfarbe)



Destination Color (Framebuffer bevor)

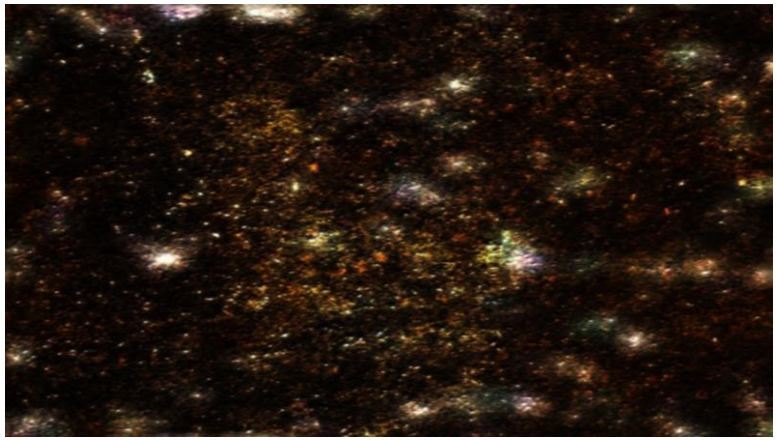
Final Color (Framebuffer danach)



Final Color = Source Color + Destination Color

Gewichtetes Additives Blending

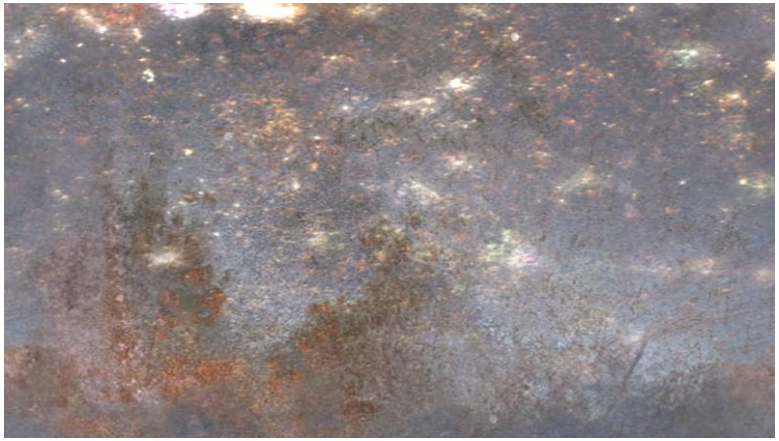
Source Color (Fragmentfarbe)



Destination Color (Framebuffer bevor)

Source Alpha (Fragment Alpha)

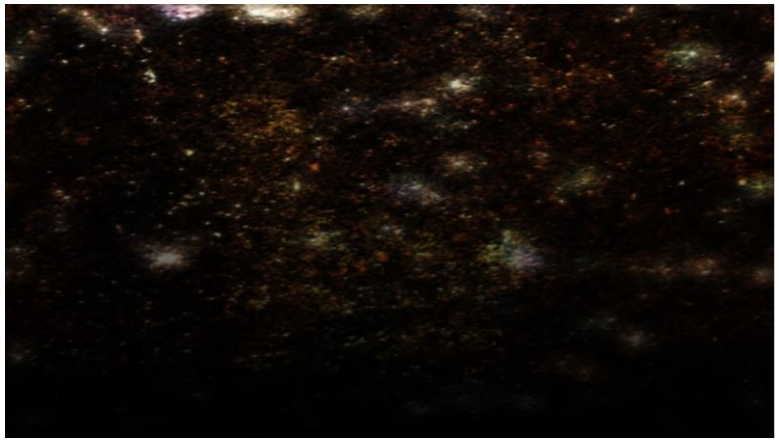
Final Color (Framebuffer danach)



Final Color = Source Alpha * Source Color + Destination Color

Vormultipliziertes Additives Blending

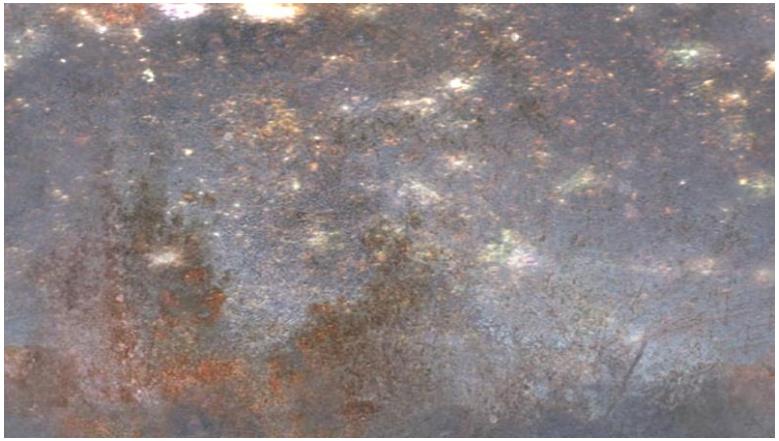
Source Color (Fragmentfarbe)



Destination Color (Framebuffer bevor)

Source Alpha (Fragment Alpha)

Final Color (Framebuffer danach)



Final Color = Source Color + Destination Color

Alpha Blending

Source Color (Fragmentfarbe)

Destination Color (Framebuffer bevor)

Source Alpha (Fragment Alpha)

Final Color (Framebuffer danach)

Final Color = Source Alpha * Source Color + (1 – Source Alpha) * Destination Color

Vormultipliziertes Alpha Blending

Source Color (Fragmentfarbe)

Destination Color (Framebuffer bevor)

Source Alpha (Fragment Alpha)

Final Color (Framebuffer danach)

Final Color = Source Color + (1 – Source Alpha) * Destination Color

Vormultipliziertes Alpha Blending

- Final Color = Source Color + (1 – Source Alpha) * Destination Color

- Fasst alle gängigen Blend Modes zusammen
  - **Additives Blending:** Source Alpha = 0
  - **Alpha Blending:** Source Color *= Source Alpha im Fragment Shader
  - **Gewichtetes Additives Blending:**
    Source Color *= Source Alpha, dann Source Alpha = 0

- Erlaubt Mischen verschiedener Effekte in einem Draw-Call
  - Warum wichtig?

Vormultipliziertes Alpha Blending

- Final Color = Source Color + (1 – Source Alpha) * Destination Color

- Fasst alle gängigen Blend Modes zusammen
  - **Additives Blending:** Source Alpha = 0
  - **Alpha Blending:** Source Color *= Source Alpha im Fragment Shader
  - **Gewichtetes Additives Blending:**
    Source Color *= Source Alpha, dann Source Alpha = 0

- Erlaubt Mischen verschiedener Effekte in einem Draw-Call
  - Warum wichtig?
    - Tiefensortierung! (Z-Buffer hilft nicht)

Blending

- Zuordnung im Fragment Shader:
  - Source Color festgelegt durch frag_color.rgb
  - Source Alpha festgelegt durch frag_color.a

- Auch Blending mit mehreren Source Colors möglich
  - `layout(location = 0, index = 0) out vec4 add_color;`
    - "Source Color 0": **GL_SRC_COLOR**
  - `layout(location = 0, index = 1) out vec4 mul_color;`
    - "Source Color 1": **GL_SRC1_COLOR**

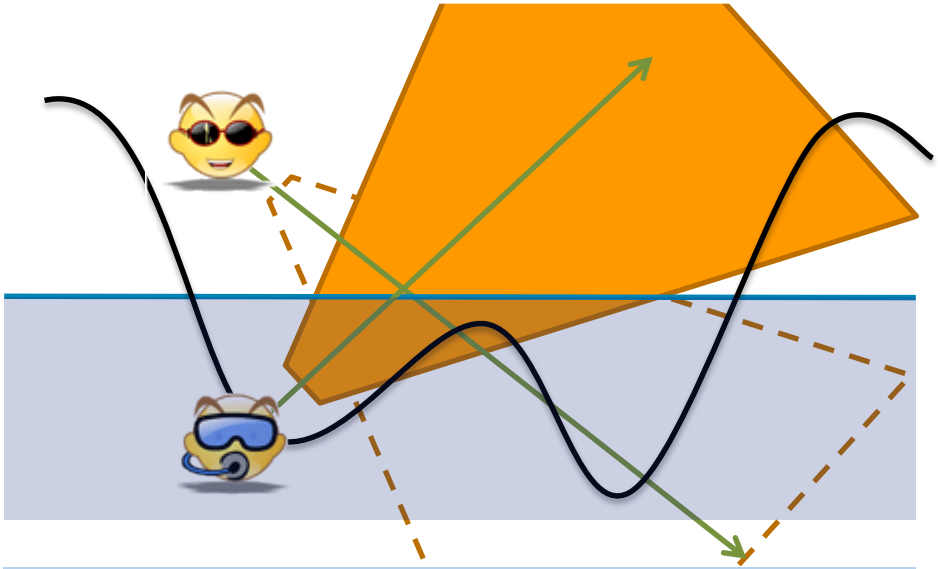
glBlendFunc(…)

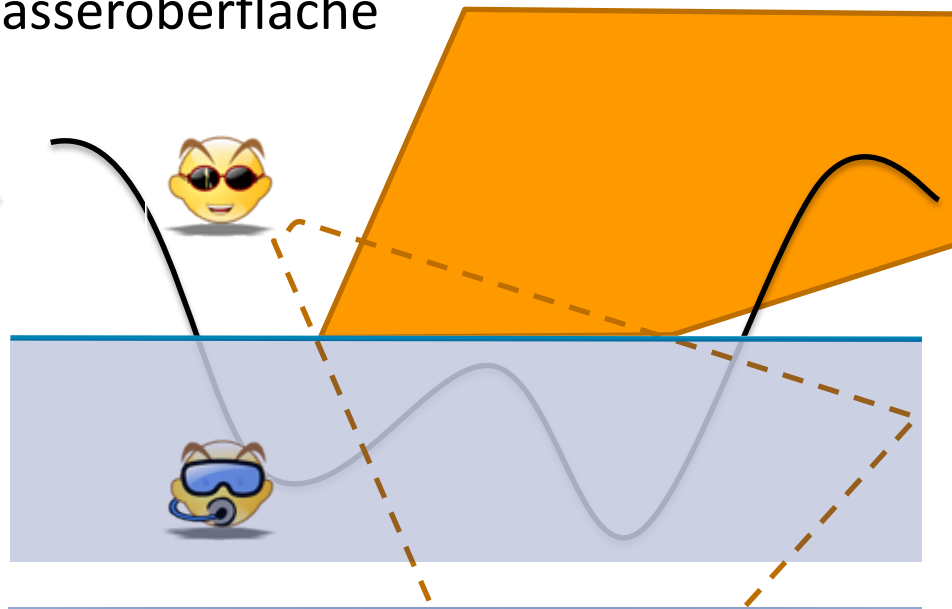
- Wasser:
  - Final Color = 1 * Source Color 0 + (Source Color 1) * Destination Color

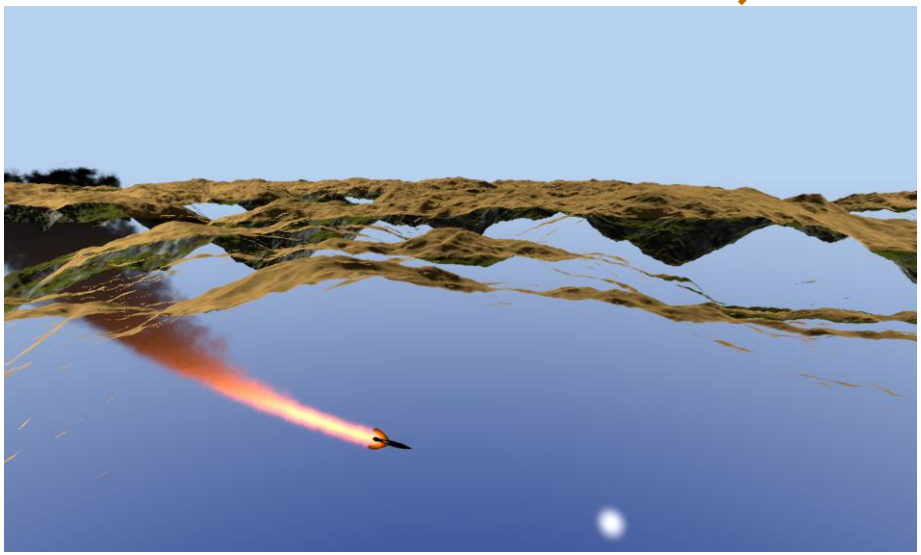
Wasser:
glBlendFunc(GL_ONE, GL_SRC1_COLOR)

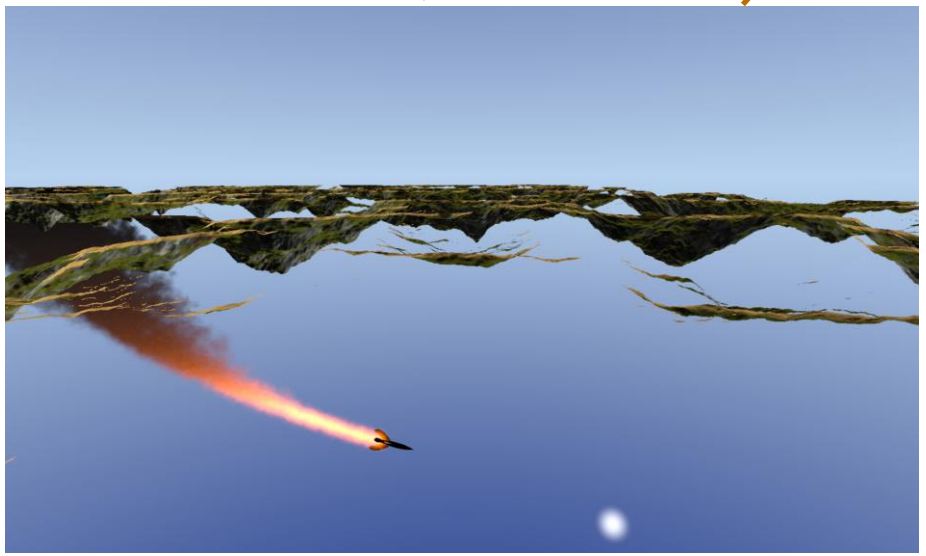
Planare Reflexion

- Rendere Szene gespiegelt an der Wasseroberfläche

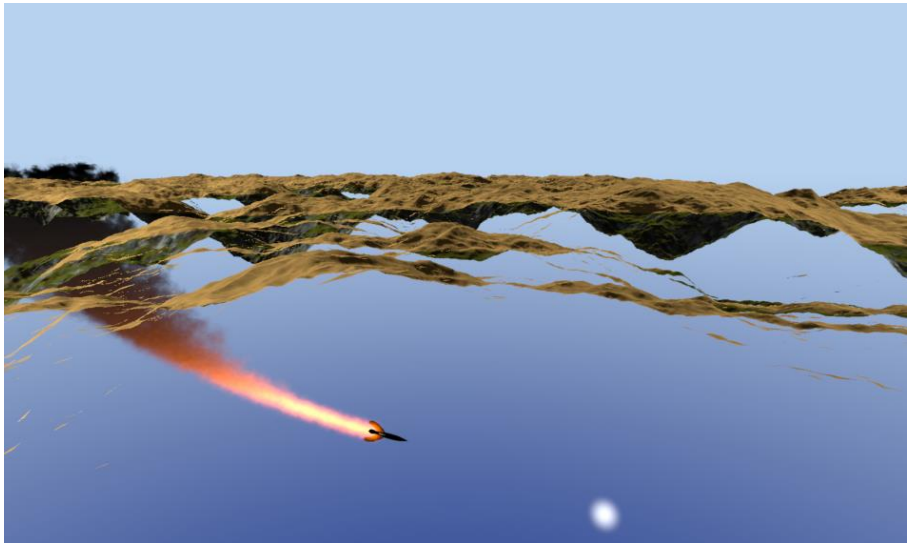


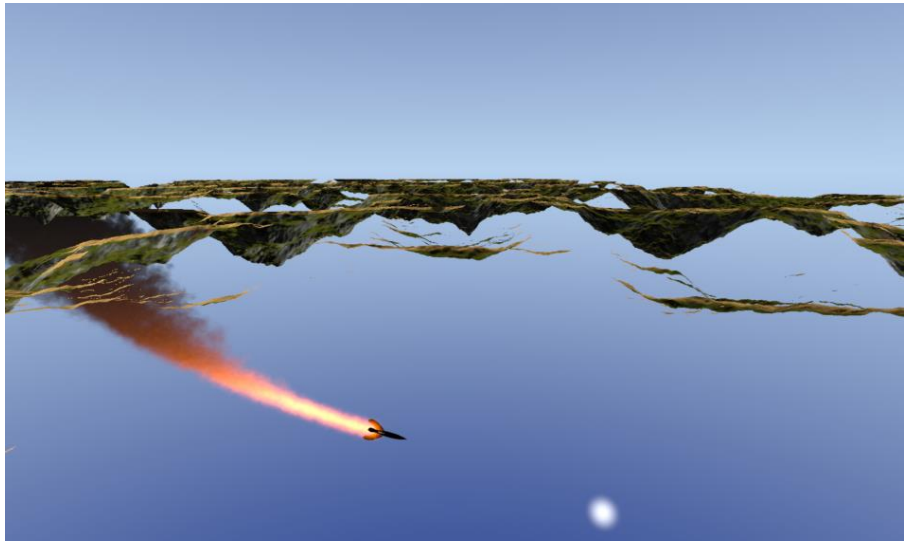






Ohne und mit Clipping an Wasseroberfläche





Clipping im Fragment Shader?

- Clipping ließe sich auch im Fragment Shader implementieren
  - Wie?

Clipping im Fragment Shader?

- Clipping ließe sich auch im Fragment Shader implementieren
  - Wie?

```
uniform vec4 water_plane;
...
in vec3 world_position;
...
out vec4 frag_color;

void main() {
  if (signed_distance[1](world_position, water_plane) < 0.0f)
    discard;

  <original shader code>
}
```

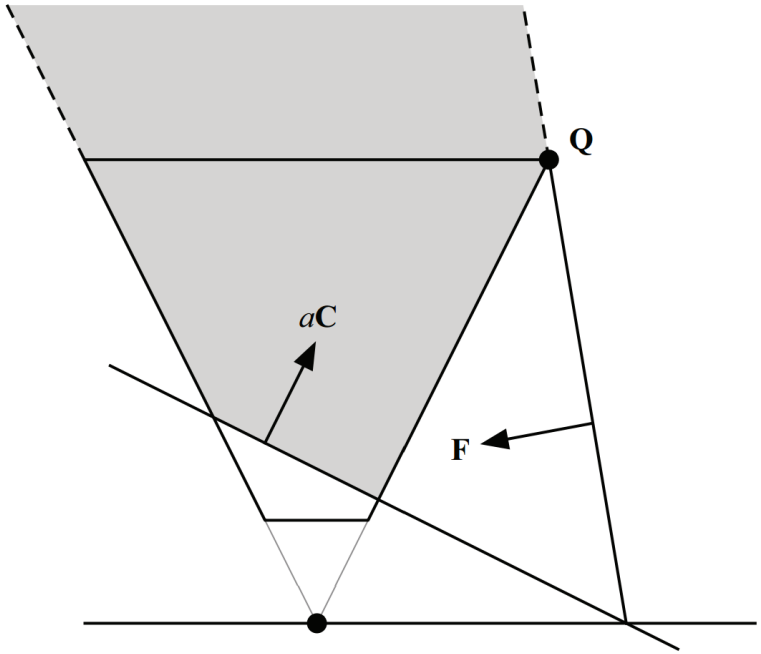
  - Warum lieber Clipping über modifizierte Near Plane?

[1] Vorzeichenbehafteter Abstand von Punkt zu Ebene (x_n, y_n, z_n, -d) entspricht 4D Skalarprodukt
`dot(vec4(world_position, 1), water_plane)`

Near-Plane Clipping

- Warum lieber Clipping über modifizierte Near Plane?
  - Effizienter: Bereits vor Ausführung des Fragment Shaders

- **Oblique Near-Plane Clipping Projection Matrix**
  - Verändere Projektionsmatrix so, dass Near Plane = Clipping Plane **C**

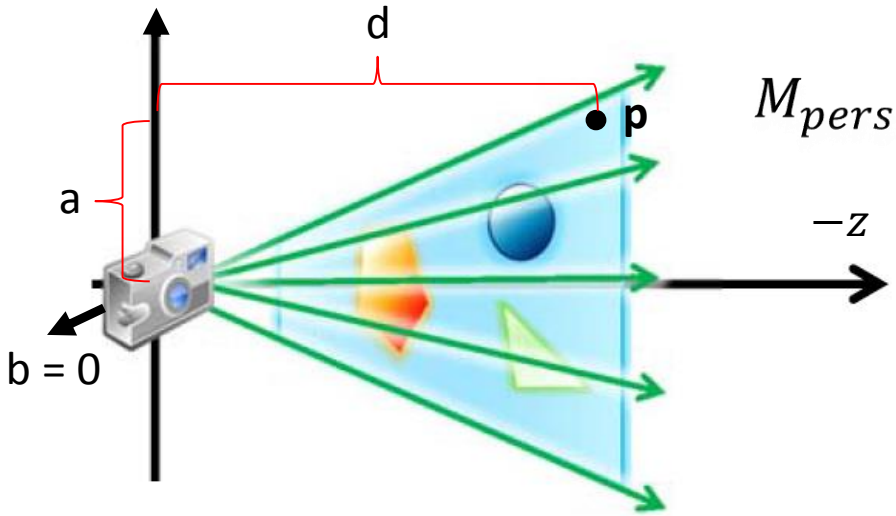


Erklärung und Code: http://www.terathon.com/code/oblique.html, http://www.terathon.com/gdc07_lengyel.pdf

Transformationen im Clip Space

- Erinnerung Koordinatenräume:
  - $p_{viewspace} = (a, b, -d)^T$
  - $p_{clipspace} = (a', b', d', d)^T$
  - $p_{ndc} = (a', b', d', d)^T/d = (a'/d, b'/d, d'/d, 1)^T$

$$M_{pers} \begin{pmatrix} a \\ b \\ -d \\ 1 \end{pmatrix} = \begin{pmatrix} a' \\ b' \\ d' \\ d \end{pmatrix}$$



$$M_{pers} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2nf}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Transformationen im Clip Space

- Translation im Clip Space:

$$\begin{pmatrix} 1 & 0 & 0 & u \\ 0 & 1 & 0 & v \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a' \\ b' \\ d' \\ d \end{pmatrix} = \begin{pmatrix} a' + u \cdot \mathrm{d} \\ b' + v \cdot d \\ d' \\ d \end{pmatrix}$$

- Entspricht Verschiebung in NDC!

$$\begin{pmatrix} a' + u \cdot d \\ b' + v \cdot d \\ d' \\ d \end{pmatrix} \Big/ d = \begin{pmatrix} a'/d + u \\ b'/d + v \\ d'/d \\ 1 \end{pmatrix}$$

Transformationen im Clip Space

- Skalierung im Clip Space:

$$\begin{pmatrix} s & 0 & 0 & 0 \\ 0 & t & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a' \\ b' \\ d' \\ d \end{pmatrix} = \begin{pmatrix} s \cdot a' \\ t \cdot b' \\ d' \\ d \end{pmatrix}$$

- Entspricht Skalierung in NDC!

$$\begin{pmatrix} s \cdot a' \\ t \cdot b' \\ d' \\ d \end{pmatrix} / d = \begin{pmatrix} s \cdot a'/d \\ t \cdot b'/d \\ d'/d \\ 1 \end{pmatrix}$$

Homogener Texturraum

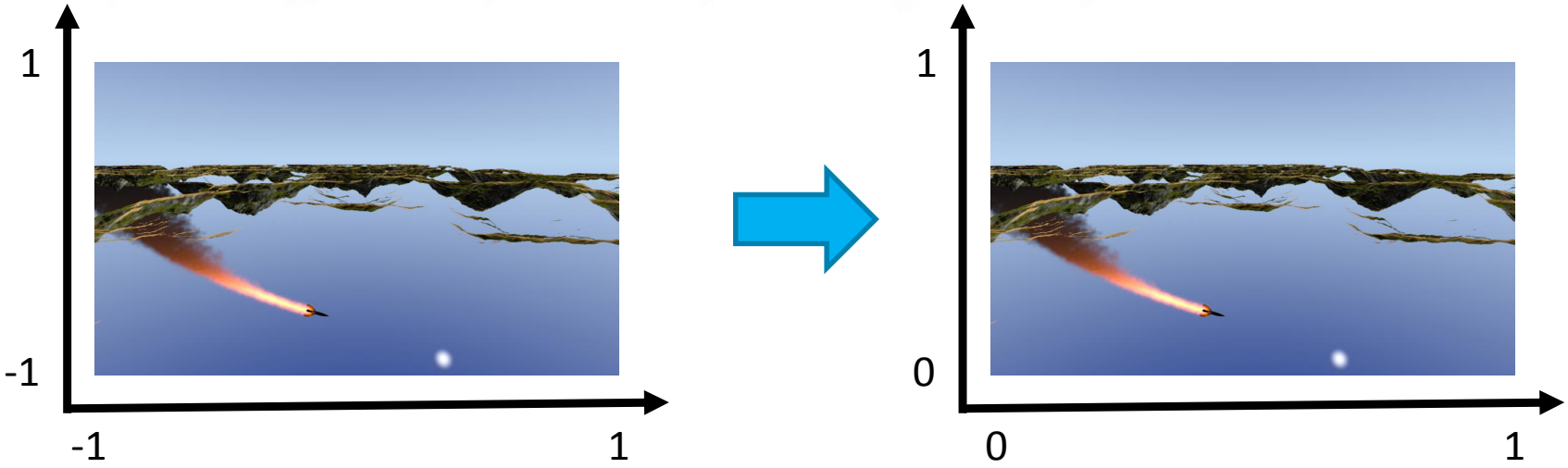
- Skalierung und Translation im Clip Space:

$$\begin{pmatrix} 0.5 & 0 & 0 & 0.5 \\ 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a' \\ b' \\ d' \\ d \end{pmatrix} = \begin{pmatrix} 0.5 \cdot a' + 0.5 \cdot d \\ 0.5 \cdot b' + 0.5 \cdot d \\ d' \\ d \end{pmatrix}$$

- Verschiebt NDC zu Texturkoordinaten:

$$\begin{pmatrix} 0.5 \cdot a' + 0.5 \cdot d \\ 0.5 \cdot b' + 0.5 \cdot d \\ d' \\ d \end{pmatrix} / d = \begin{pmatrix} 0.5 \cdot a'/d + 0.5 \\ 0.5 \cdot b'/d + 0.5 \\ d'/d \\ 1 \end{pmatrix}$$

Bereits Teil der **`RVP_to_tex`** Matrix in Aufgabe 3!



Ebenentransformation

- Ebenen (x_n, y_n, z_n, -d) (Hesse-Normalform) lassen sich wie Normalen mit homogenen Koordinaten transformieren!
  - Normale: d = 0, entspricht Ebene durch den Ursprung

- Was ist bei der Transformation von Normalen zu beachten?

Ebenentransformation

- Ebenen (x_n, y_n, z_n, -d) (Hesse-Normalform) lassen sich wie Normalen mit homogenen Koordinaten transformieren!
  - Normale: d = 0, entspricht Ebene durch den Ursprung

- Was ist bei der Transformation von Ebenen zu beachten?
  - Transformiere mit **transponierter inverser** Matrix!

$$\left((M^{-1})^T \begin{pmatrix} x_n \\ y_n \\ z_n \\ -d \end{pmatrix} \right)^T \cdot M \begin{pmatrix} x_p \\ y_p \\ z_p \\ 1 \end{pmatrix} = \begin{pmatrix} x_n \\ y_n \\ z_n \\ -d \end{pmatrix}^T M^{-1} \cdot M \begin{pmatrix} x_p \\ y_p \\ z_p \\ 1 \end{pmatrix}$$

$$= \begin{pmatrix} x_n \\ y_n \\ z_n \\ -d \end{pmatrix}^T \cdot \begin{pmatrix} x_p \\ y_p \\ z_p \\ 1 \end{pmatrix} = 0$$

p in Ebene (**n**, -d) ⇔ M**p** in Ebene $(M^{-1})^T$(**n**, -d)