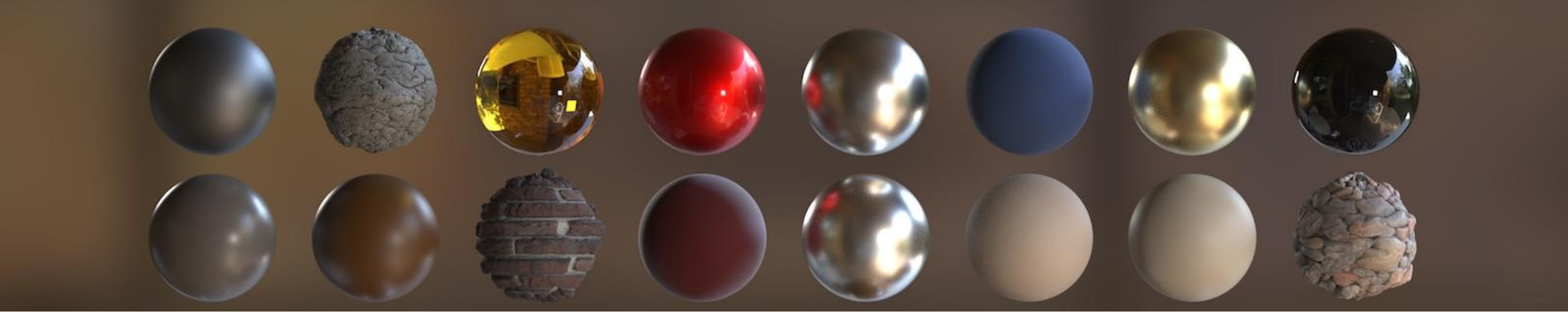
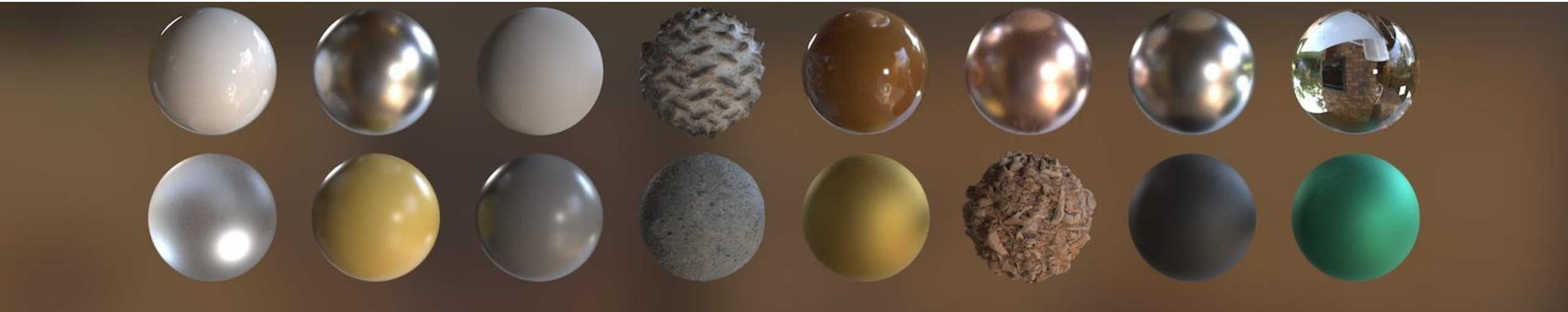
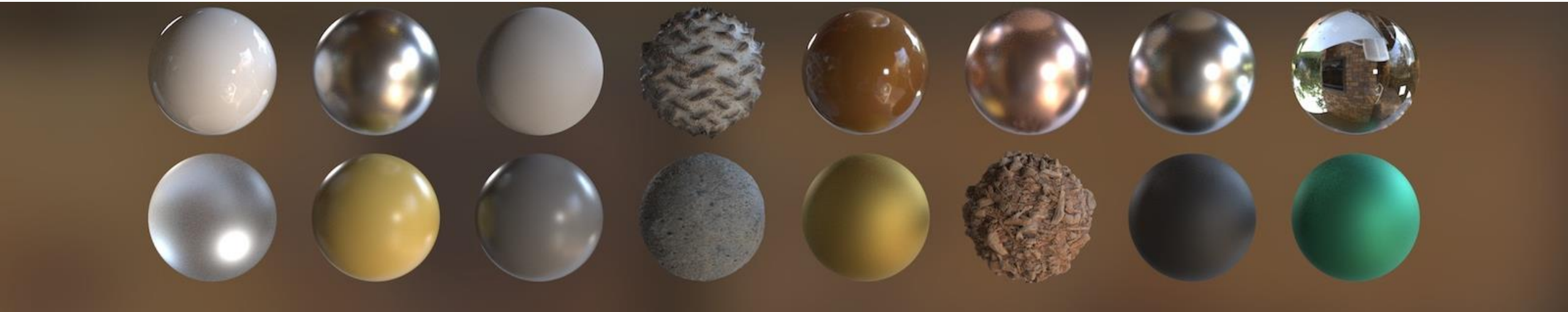


Shader Praktikum

FH Wedel



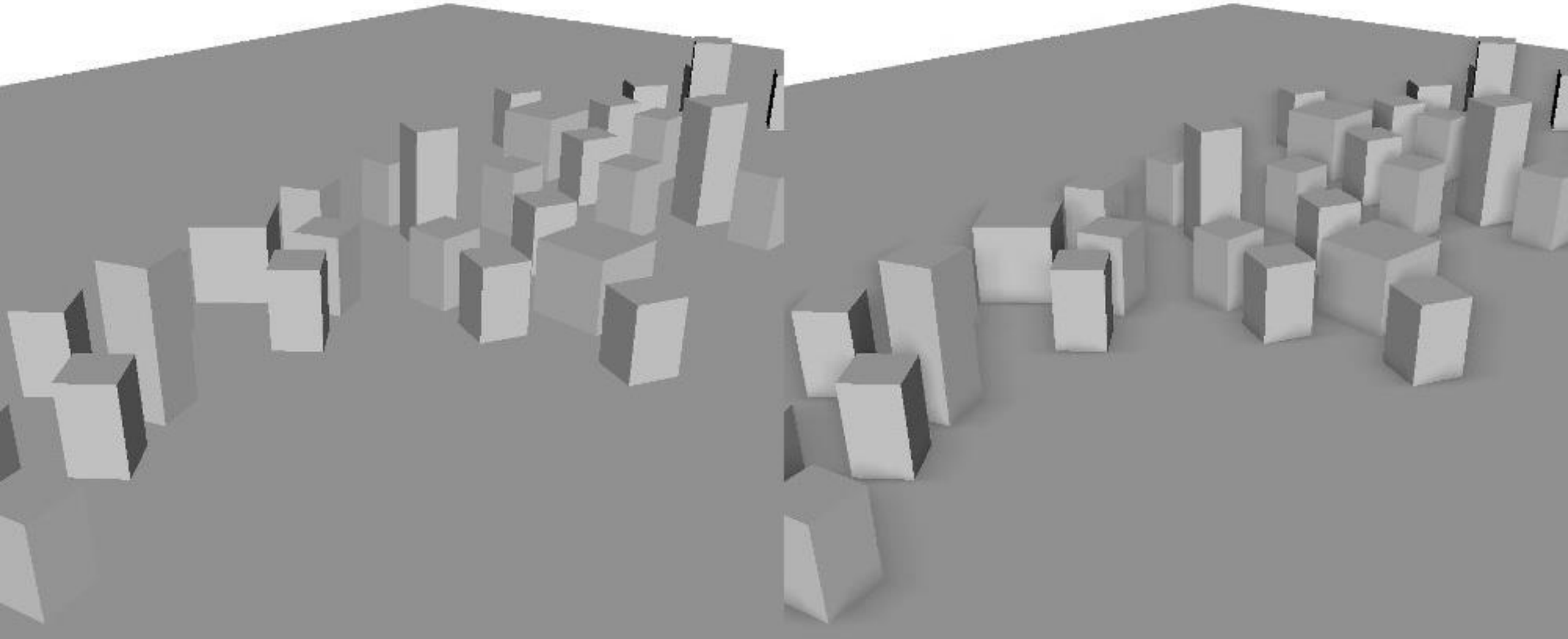
Shader – Aufgabe 4



Themen

- Screen Space Ambient Occlusion (SSAO)

Ambient Occlusion



Ambient Occlusion (AO)

- **Umgebungsbeleuchtung** (ambient Lighting)
  - fixe Licht Konstante
  - Simulation der Lichtstreuung
- Realität
  - Licht **streut** in alle Richtungen mit unterschiedlichen Intensitäten
  - Indirekt beleuchtete Teile der Szene sollten unterschiedliche Intensitäten besitzen

Ambient Occlusion (Umgebungsverdeckung)

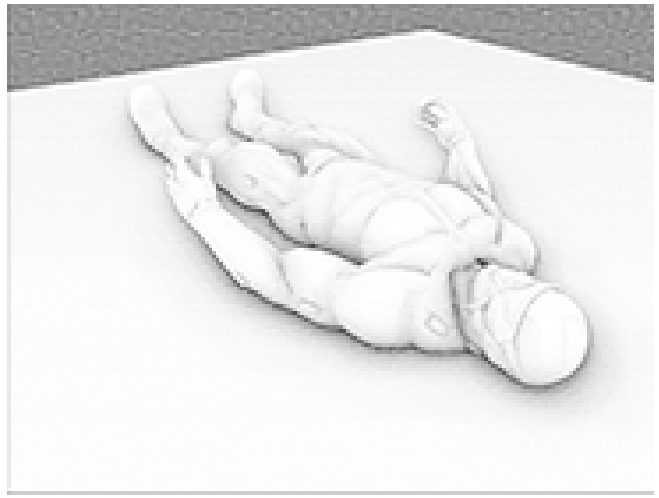
- Maß für die **Erreichbarkeit** eines **Oberflächen Punktes**
- Approximation **von indirekter Beleuchtung**
  - Abdunkeln von Falten, Löchern & Oberflächen die dicht beieinander sind
  - Diese Bereiche werden hauptsächlich von ihrer umgebenden Geometrien verdeckt
    - „teure“ Technik, da die Geometrie beachtet werden muss
- Optionen
  - **genaue** (**statische**) Berechnung **vor** der Laufzeit (Pre-Processed Ambient Occlusion)
  - **ungenaue** aber **dynamische** Berechnung **zur** Laufzeit (Screen Space Ambient Occlusion)

Screen Space Ambient Occlusion

https://learnopengl.com/Advanced-Lighting/SSAO

Screen-Space Ambient Occlusion

- Nutzung der **Szenetiefe in Screen-Space** (anstelle echter Geometrie) um die Verdeckung zu bestimmten
  - **Verdeckungsfaktor**-Berechnung [0.0,1.0] abhängig von den dem Fragment umgebenden Tiefenwerten
  - Verdeckungsfaktor wird in der Beleuchtungsberechnung benutzt um den Ambienten-Teil zu minimieren
  - Auf **Pixelebene**
- Die Technik ist *extrem schnell* im Vergleich zu realer Umgebungsverdeckung und ergibt trotzdem gute Ergebnisse
- Standard zur Approximation real-time Umgebungsverdeckung (**ambient Occlusion**)

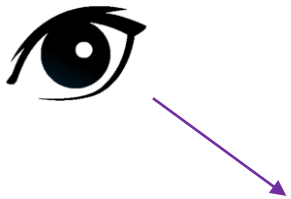


SSAO-Buffer

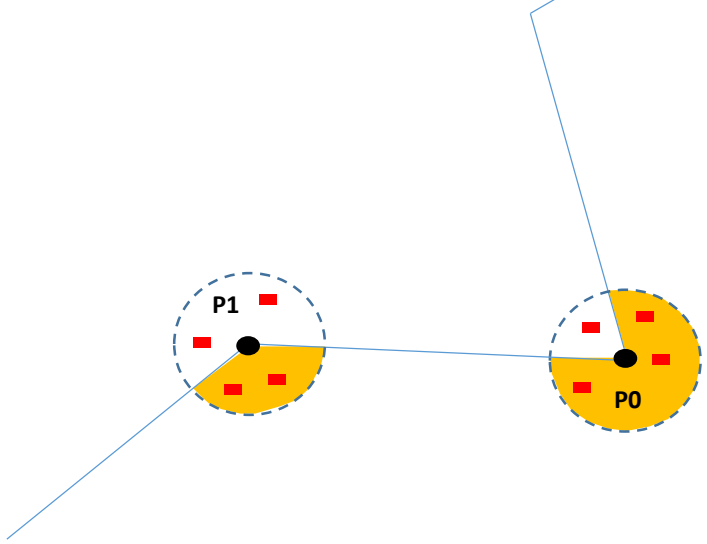
https://learnopengl.com/Advanced-Lighting/SSAO

Screen-Space Ambient Occlusion

View-Position & -Richtung



Hülle der Geometrie



Orangene Bereiche – Verdeckung des (schwarzen) Punkts Px

weiße Bereiche – keine Verdeckung des (schwarzen) Punkts Px

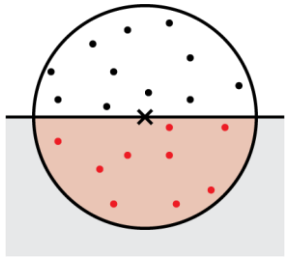
Rote Rechtecke – Samples um den (schwarzen) Punkts Px

Screen Space Ambient Occlusion

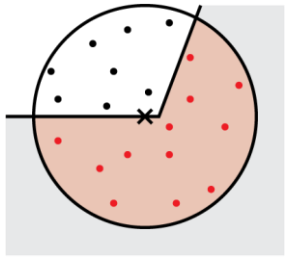
Sampling mit *Kugelförmigen* Kernel

Problem

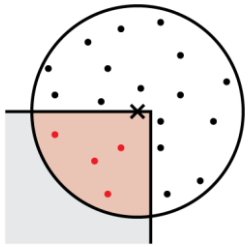
- In der Ebene zu dunkel und an Kanten zu hell



In der Ebene zu dunkel



Dunkler bei Verdeckung

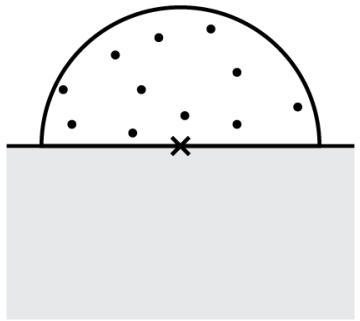


Zu hell an Kanten

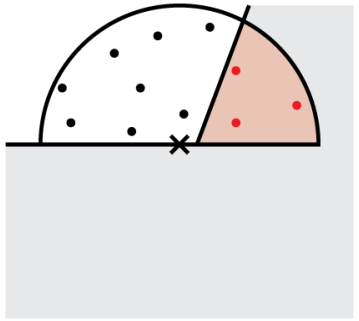
Rote „Samples“ als Verdeckungsfaktor

Screen-Space Ambient Occlusion

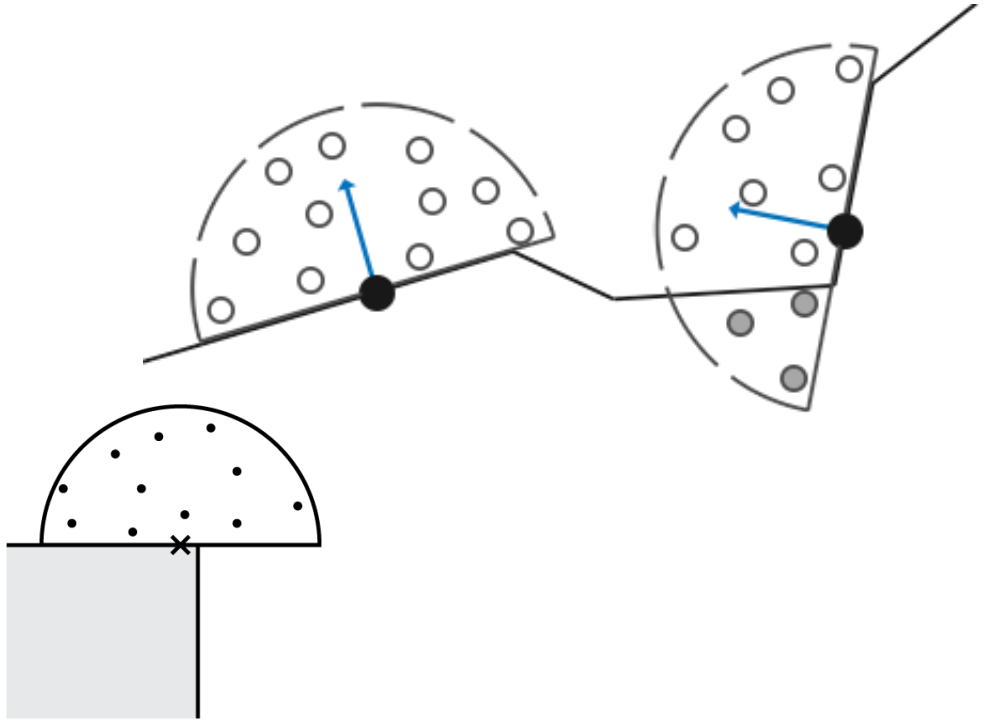
- Nutzung einer **Halbkugel (Hemisphäre)**, die entlang der **Normalen** orientiert ist (oberhalb des Horizontes)
  - Dadurch wird die Geometrie, die sich hinter dieser befindet nicht berücksichtig (verfälscht nicht den Verdeckungsfaktor)
  - Berechnung von Tangential-Koordinantensystem



Keine Verdeckung in der Ebene



Verdeckung nur in einer Hemisphäre

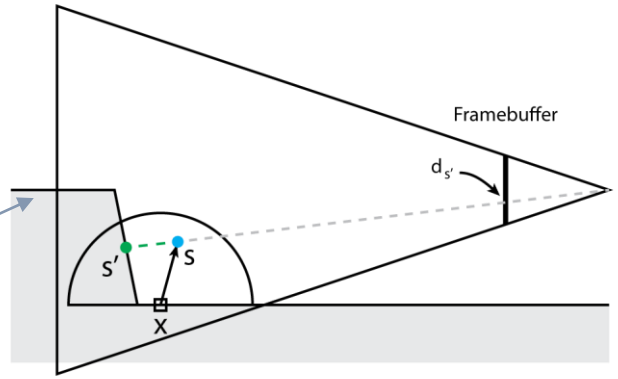


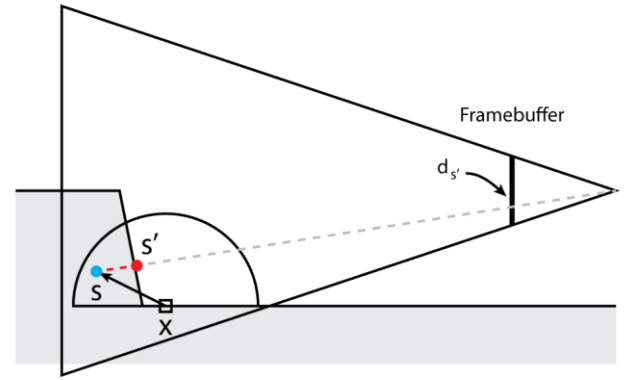
Keine Aufhellung an den Kanten

https://learnopengl.com/Advanced-Lighting/SSAO

Screen-Space Ambient Occlusion (Algorithmus)

- Sample *s* wird zufällig in der Hemisphäre gewählt
- Tiefe des Samples *s* wird mit dem Wert des Depth Buffers $d_{s'}$ an der korrespondierenden Position verglichen
- Vergleich mit: Tiefe $d_{s'}$ entspricht dem Fragment *s'*
  - Sample s liegt *vor* dem Fragment *s'* → **keine Verdeckung**
  - Sample s liegt *hinter* dem Fragment *s'* → **Verdeckung** bei diesem Sample
- Anteil der Samples ohne Verdeckung ist der SSAO-Wert



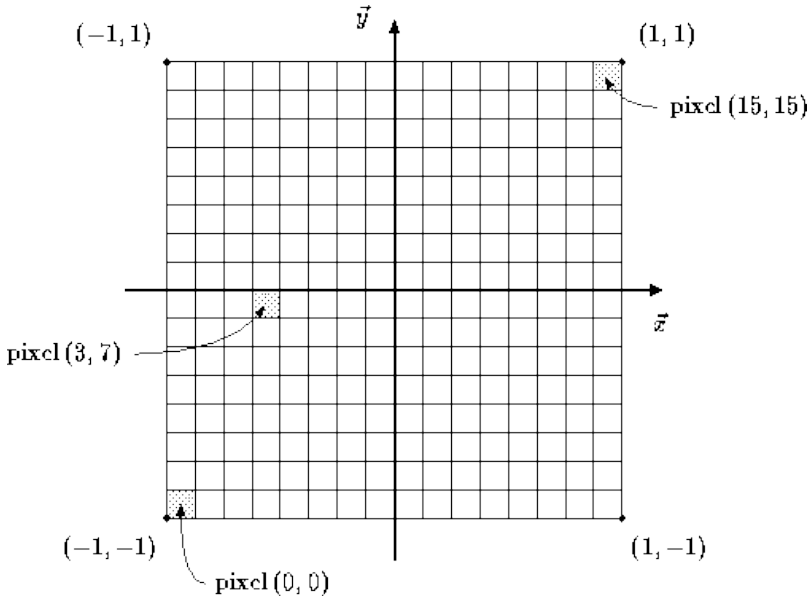


Screen-Space Ambient Occlusion

- Screen-Space-Technik
  - Berechnung auf Screen-füllendem Quad
  - Es liegen keine „echten“ geometrischen Daten vor
  - Samplen der Daten mittels einer Screen-Space Textur

Transformation-Algorithmus

- Konvertierung des Sample Vektors (vec3) in Vec4
  - Setzen der W-Komponente auf 1
- Multiplikation mit der **Projektions-Matrix** (um den Vektor vom view-space in den cliping-space zu bringen)
- (manuelle) **perspektivische Division** (teilen der xyz-komponenten durch die w-Komponente)
- Transformation der xyz-Komponenten von [-1.0, 1.0] in Werte zum Samplen [0.0, 1.0]
- sampleDepth = texture(gBufferPosition, transVek.xy).z;



http://www.idav.ucdavis.edu/education/GraphicsNotes/Screen-Space/Screen-Space.html

Screen-Space Ambient Occlusion

Depth Buffer und Projektion

- Nach der Projektion (Multiplikation mit Projektionsmatrix)
  - Tiefenwerte im Intervall [-1, 1]
- Depth Range Transformation (glDepthRange)
  - Transformation in das Intervall [0,1]
- Schreiben auf Depth-Buffer

- Lesen von Tiefenwerten aus dem Depth-Buffer
  - (manuelle) **Rücktransformation in das Intervall [-1, 1]**
    - Damit diese korrekt mit der projizierten Sample-Position verglichen werden können

Screen-Space Ambient Occlusion

Pre-Pass (*Geometrie-Pass*)

- Szene wird mit einem G-Buffer FBO gerendert (vgl. Deferred Shading)

SSAO-Pass

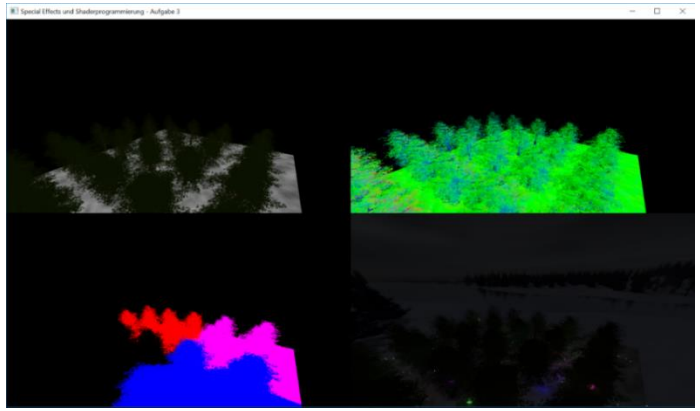
- Rendern eines Full-Screen Quads, damit für jeden Pixel ein Fragment erzeugt wird
- Berechnung **Ambient Occlusion Wert** auf den **SSAO-Buffer/-Texture im G-Buffer-Objekt***

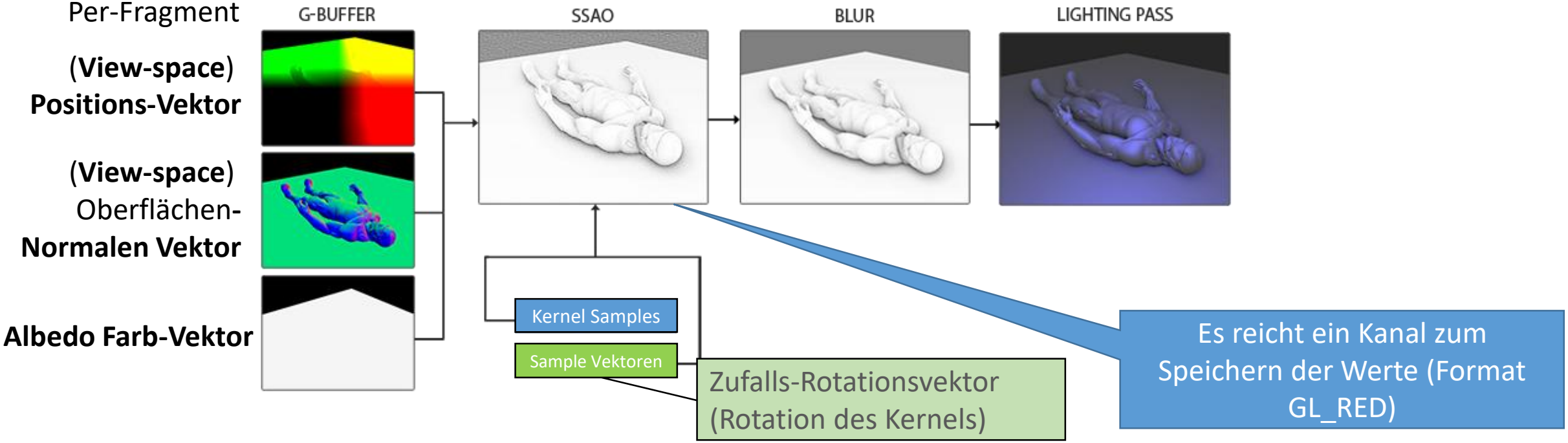
Main-Pass (*Lighting-Pass*)

- Zweites mal Render der Szene
- *Multiplikation Ambient Occlusion (-faktor) mit Umgebungsbeleuchtung* (ambient)
- [FragColor = (ambient * *ambientOcclusion*) + diffuse + specular]

* muss ergänzt werden

Screen-Space Ambient Occlusion





Screen-Space Ambient Occlusion

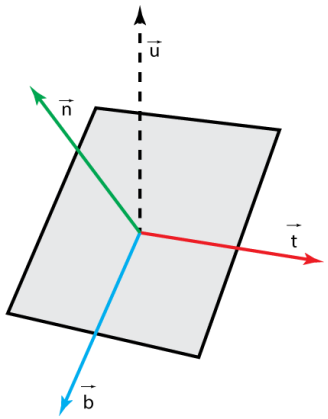
- Qualität und Genauigkeit hängen von der Anzahl an **Umgebungs-Samplen** ab
  - Bei zu *geringer* Anzahl, nimmt die *Genauigkeit* drastisch ab und es kommt zu einem Effekt namens *Banding*
  - Bei zu hoher Anzahl, geht es auf Kosten der *Performanz*
- Daher
  - Verringern der Anzahl von Samples durch das **zufällige rotieren des Sample Kernels** (geringere Anzahl an Samples → trotzdem hohes qualitatives Resultat)
    - Entstehung eines bemerkbaren **Rausch(Noise)-Patterns**
    - Dies kann mithilfe eines **Weichzeichnungs-(Blur)-Filters** minimiert werden
      - Nutzung des Ping-Pong Buffers zum Weichzeichnen

Sample Position ermitteln

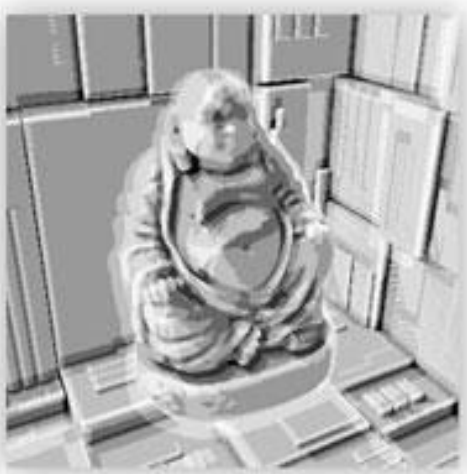
Tangent Space

- Verteilung **Samples** in der **Hemisphäre**
  - Pro Pixel ein Tangent Space
  - Orientierung der Tangente und Bi-Tangente in Ebene ist irrelevant

- Die Normale $\vec{n}$ eines Pixels kann aus dem **Pre-Pass** (Geometrie-Pass) ermittelt werden
- Tangente & Bi-Tangente können durch das Kreuzprodukt bestimmt werden
- Bestimmung 1. Achse unter Nutzung eines normierten zufälligen Vektors $\vec{u}$
  - Anwendung der Gram-Schmidt-Verfahrens
    - *Tangente t = normalize(u – n * dot(u, n))*
    - *Bitangente b = cross(n, t)*
- ACHTUNG Spezialfall: $\vec{n} = \vec{u}$

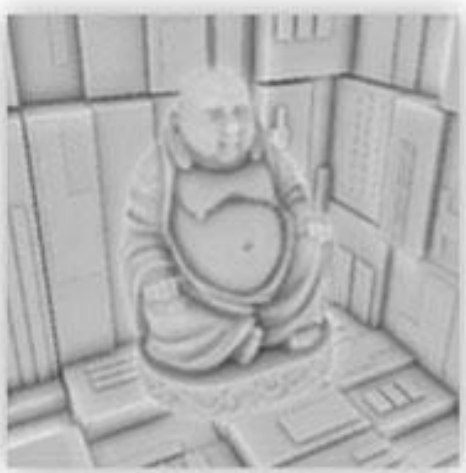


Screen-Space Ambient Occlusion

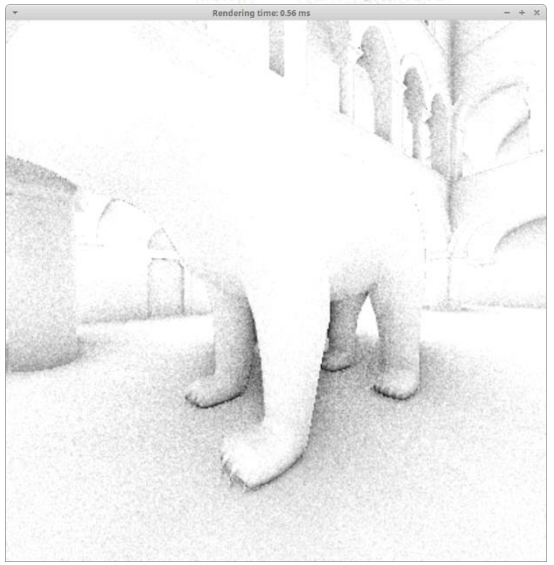


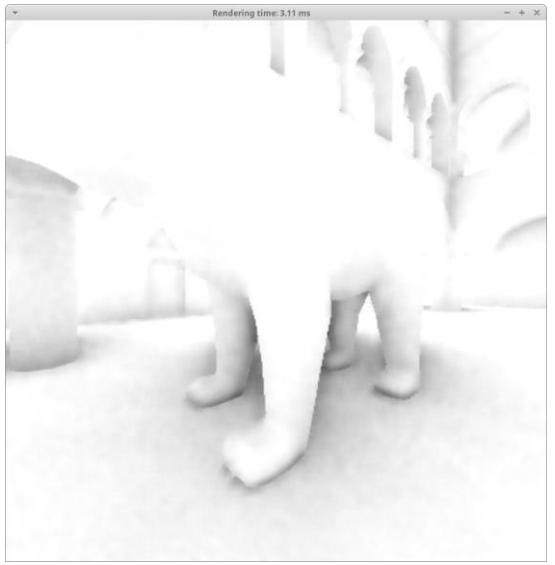
low sample 'banding'

random rotation = noise



+ blur = acceptable





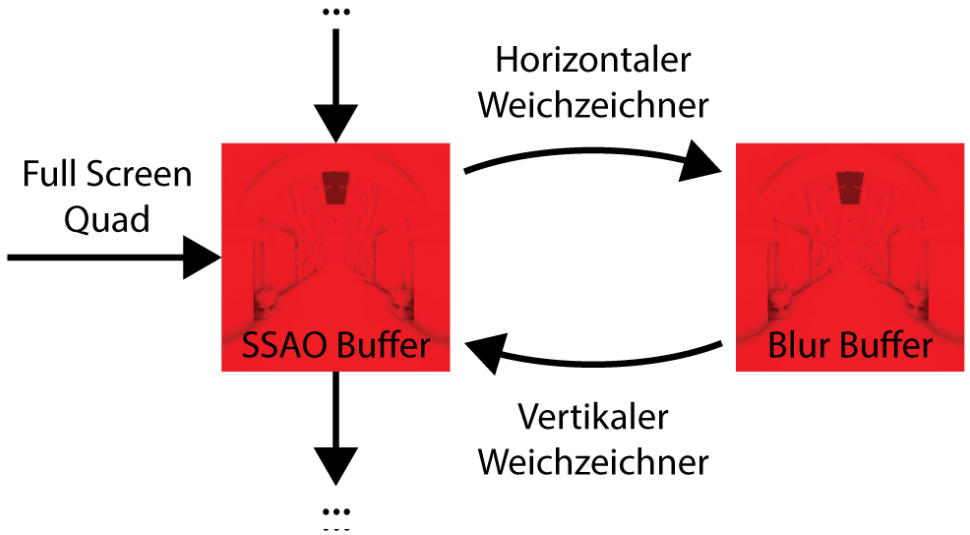
https://learnopengl.com/Advanced-Lighting/SSAO

Blur

- Nutzung eines 2D- bzw. (verbessert) 2x 1D-Filter
  - Unterteilung Horizontale Filterung
  - Unterteilung Vertikale Filterung

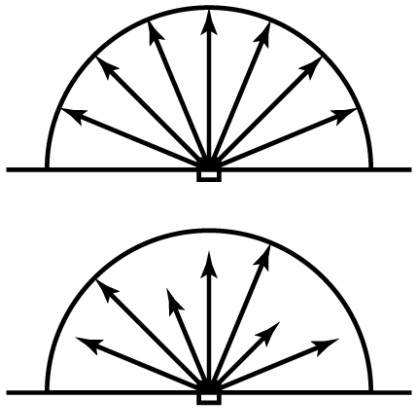
Weichzeichner

- Weichzeichner (z.B. mit Compute Shader)
- Weichzeichner in 2 Schritten
  - Horizontales Weichzeichnen (Lesen aus SSAO Framebuffer und Schreiben in Blur Buffer)
  - Vertikales Weichzeichnen (Lesen aus Blur Buffer und Schreiben in SSAO Framebuffer)

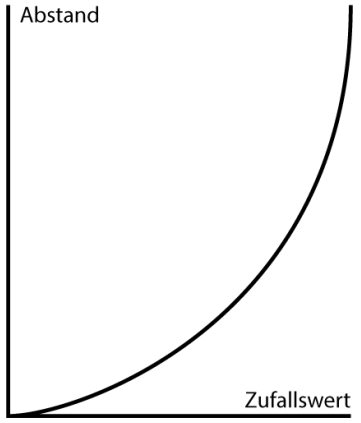


Sampling mit unterschiedlichem Abstand

- Die Samples sollen einen **unterschiedlichen Abstand** zum Ursprung haben aber so verteilt sein, dass **nahe dem Ursprung mehr Samples** erzeugt werden
- Unterschiedliche Abstande z.B. durch **pseudo-Zufallswerte**
- Verteilung durch Abbildung der Zufallswerte mit einer Potenzfunktion
  - z.B. **float** lerp (**float** a, **float** b, **float** f) { **return** a + f * (b - a) ; }



Unterschiedlicher Abstand



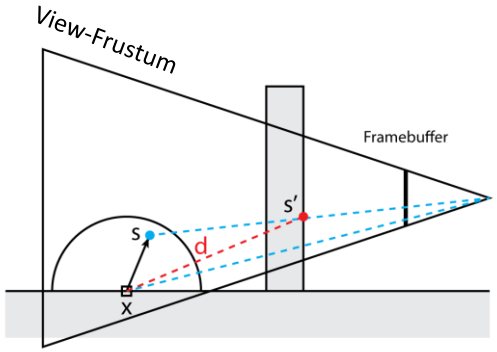
Verteilung der Abstände

Abstandsbeschränkung

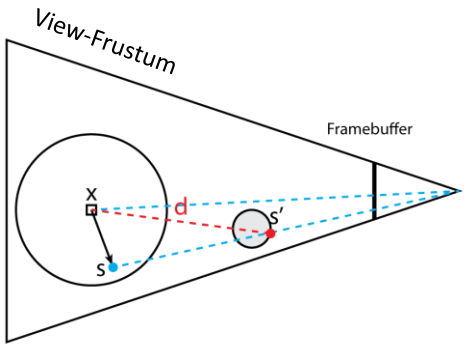
http://www.abload.de/img/screenshot2012-11-27a5up1z.png

Abstandsbeschränkung

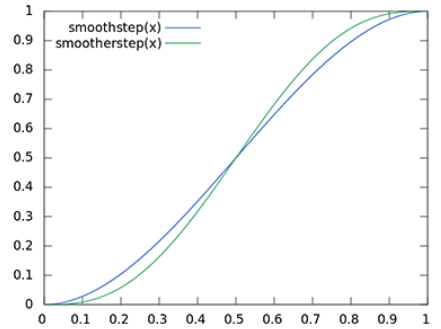
- Der zum Sample s korrespondierende Punkt *s´*
  ist zu weit vom Punkt x entfernt
  → Sample wird **nicht berücksichtig**
- **Abstandsvergleich im Betrachter- oder Welt-Koordinatensystem**, damit die perspektivische Verzerrung die Ergebnisse nicht beeinflusst
- *rangeCheck = smoothstep(0.0, 1.0, radius / distance)*
  - Hier range Check als Multiplikator für den Occlusion Wert
  - distance – Tiefen-Abstand zwischen der Fragment-Position & der gesampelten Tiefe



Seitenansicht



Topansicht

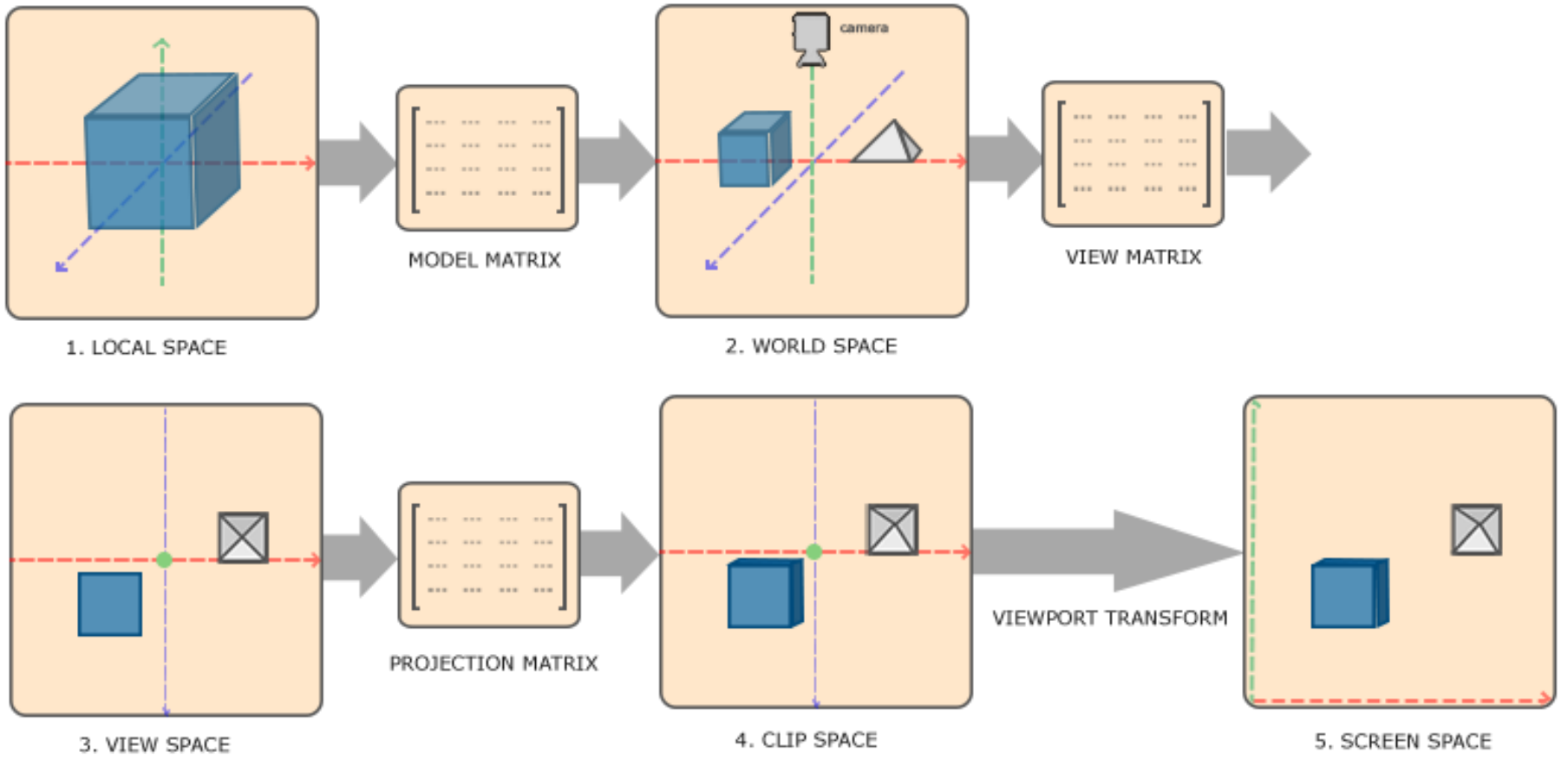


smoothstep-Funktion

Grundlegender Algorithmus

- Berechnen der **Welt-** oder **Betrachter-Position** & **-Normale** des Fragments
- Berechnen von Basisvektoren für ein **Tangential-Koordinatensystem** (TBN-Matrix)
- Für jedes Sample (*kernelSize*)
  - **Hammersley-Koordinate** berechnen (Funktion hammersley)
  - **Sampling Vektoren im Tangential-Koordinatensystem berechnen** (Funktion sampleHemisphere)
    - Alternativ können diese im Host vorberechnet werden und
      - Als Noise-Textur übergeben werden
      - (oder) als uniform vec3[] übergeben werden
  - Sampling Vektoren ins Welt- oder Betrachter-Koordinatensystem **transformieren**
    - *sampleViewSpace = TBN * sampleTangentSpace* (Hier TBN → im View-Space)
  - **Sample-Positionen** im Screen-Space berechnen
  - Tiefe des Samples mit dem Wert des Depth-Buffers an der projizierten Position des Samples vergleichen unter Beachtung des Abstandes
    - wenn *samplerDeepth >= sample.z + bias* → *occlusion += rangeCheck*
    - Sonst *occlusion += 0*
- Gesammt Occlusion durch Anzahl der Samples teilen (*occlusion = 1.0 – occlusion / KernelSize)*

Positionsberechnung aus Tiefenbild



Positionsberechnung aus Tiefenbild

- Vertex Post-Processing

$Vertices\ im\ ViewSpace$ → Multiplikation mit **Projectionmatrix** (Clipp-Space)
→ W-Division → **Viewport** Multiplikation

$$\begin{pmatrix} \frac{1}{\text{aspect} * \tan\left(\frac{FOV}{2}\right)} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{1}{\tan\left(\frac{FOV}{2}\right)} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-n-f}{n-f} & \frac{2fn}{n-f} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Projektionsmatrix*

$$\begin{pmatrix} x_{ndc} \\ y_{ndc} \\ z_{ndc} \end{pmatrix} = \begin{pmatrix} \frac{x_c}{w_c} \\ \frac{y_c}{w_c} \\ \frac{z_c}{w_c} \end{pmatrix}$$

Perspektivische Division

$$\begin{pmatrix} x_w \\ y_w \\ z_w \end{pmatrix} = \begin{pmatrix} \frac{width}{2} x_{ndc} + x + \frac{width}{2} \\ \frac{height}{2} y_{ndc} + y + \frac{height}{2} \\ \frac{f` - n`}{2} z_{ndc} + \frac{f` + n`}{2} \end{pmatrix}$$

Viewport Multiplikation

* wenn r = -l & t = -b ist, dann gilt:

$$\frac{r+l}{r-l} = 0$$

$$\frac{t+b}{t-b} = 0$$

FOV – Field of View	n – vordere Clipping ebene	n` - vordere Clipping Plane [0,1]
aspect – Breite / Höhe	f – hintere Clipping Ebene	f` - hintere Clipping Plane [0,1]
r – rechte Seite des Clippings	$?_c$ – Koord. im Clippspace	
l – linke Seite des Clippings	$?_{ndc}$ – Normalisierte Device Koordinanten	
t – top Seite des Clippings	$?_w$ – window-space Koordinaten	
b – boden Seite des Clippings		

Positionsberechnung aus Tiefenbild

- Positionsbestimmung im WorldSpace
  - (discard, wenn Tiefe auf „farVal“ geclampt wurde)
  - Rückgängig machen der Viewport Transformation
    - **f`** = gl_DepthRange.far
    - **n`** = gl_DepthRange.near
  - Multiplikation mit inverser ViewProjektions-Matrix
  - W-Division

Depth Reconstruction

- Berechnung der Viewspace Position aus dem Tiefenbuffer
  - Es wird viel weniger Speicher benötigt (1 floating point Wert pro Pixel)
- Berechnung des Dept-Buffers
  - (vertex) Object-Space Position → World-Space → View-Space → Projektion auf near clipping plane (4D Vector mit dem view space z Koordinate in der 4. (w) Komponente)
  - Clipping der ersten 3 Komponenten zwischen –w und w
  - Perspektivische Division (durch w)
  - → Normalized Device Coordinates (xyz [-1,1] & w = 1)
  - !!! Z-Buffer [0,1]

Depth Reconstruction

Projektions Matrix

Perspektivische Projektion

$$\begin{bmatrix} \frac{1}{\text{aspect} * \tan\left(\frac{FOV}{2}\right)} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan\left(\frac{FOV}{2}\right)} & 0 & 0 \\ 0 & 0 & \frac{-n-f}{n-f} & \frac{2fn}{n-f} \\ 0 & 0 & -1 & 0 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{X}{\text{aspect} * \tan\left(\frac{FOV}{2}\right)} \\ \frac{Y}{\tan\left(\frac{FOV}{2}\right)} \\ Z\frac{-n-f}{n-f} + \frac{2fn}{n-f} \\ -Z \end{bmatrix}$$

aspect –Verhältnis (width/height)
FOV – field of view
n – near clipping plane
f – far clipping plane

Screen-Space Koordinaten

Tiefen Rekonstruction

$$Z_{NDC} = \frac{Z\frac{-n-f}{n-f} + \frac{2fn}{n-f}}{Z}$$

NDC – perspektivische Division (durch w-Komponente)

$$Tiefe = \left(\frac{\left(Z * \frac{-n-f}{n-f} + \frac{2fn}{n-f}\right)}{Z} + 1\right) * 0.5$$

Daher:

$$Z = \frac{\frac{2fn}{n-f}}{2*Tiefe-1-(\frac{-n-f}{n-f})}$$

Berechnen von **X** und **Y**

$$-1 \leq \frac{X}{Z*\text{aspect}*\tan\left(\frac{FOV}{2}\right)} \leq 1$$

$$-\text{aspect} * \tan\left(\frac{FOV}{2}\right) \leq \frac{X}{Z} \leq \text{aspect} * \tan\left(\frac{FOV}{2}\right)$$

$$-1 \leq \frac{Y}{Z*\tan\left(\frac{FOV}{2}\right)} \leq 1$$

$$-\tan\left(\frac{FOV}{2}\right) \leq \frac{Y}{Z} \leq \tan\left(\frac{FOV}{2}\right)$$

Hammersley Sequenz

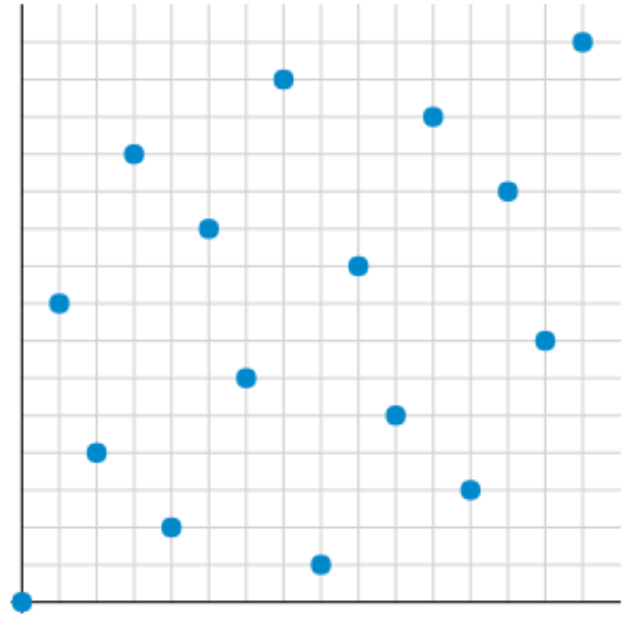
Definition

Sequenz von 2D-Punkten mit einer annähernden Gleichverteilung im Intervall [0, 1) auf beiden Achsen

$$H_n = \left\{ \begin{pmatrix} \frac{i}{N} \\ \Phi_2(i) \end{pmatrix}, \forall i \in 0, \dots, N-1 \right\}$$

Eigenschaften

- Pseudo-Zufällig (sieht aber zufällig aus)
- Anzahl Punkte (N) muss vorab feststehen



http://holger.dammertz.org/stuff/notes_HammersleyOnHemisphere.html

Hammersley Sequenz (Implementierung)

i dezimal	*i* binär	$\Phi_2(i)$ binär	$\Phi_2(i)$ dezimal
0	0000.0	0.0000	0.0
1	0001.0	0.1000	0.5
2	0010.0	0.0100	0.25
3	0011.0	0.1100	0.75

```
float radicalInverse_VdC (uint bits) { // Investierung der Binärdarstellung der X-Komponente am Komma
        bits = (bits << 16u) | (bits >> 16u) ;
        bits = ((bits & 0x55555555u) << 1u ) | ((bits & 0xAAAAAAAAu) >> 1u ) ;
        bits = ((bits & 0x33333333u) << 2u ) | ((bits & 0xCCCCCCCCu ) >> 2u ) ;
        bits = ((bits & 0x0F0F0F0Fu)  << 4u ) | ((bits & 0xF0F0F0F0u   ) >> 4u ) ;
        bits = ((bits & 0x00FF00FFu)  << 8u ) | ((bits & 0xFF00FF00u   ) >> 8u ) ;
        return float (bits) * 2.3283064365386963e-10; // 0x100000000
}

vec2 hammersley (uint i, uint N) {
        return vec2(float(i)/float(N), radicalinverse_VdC(i));
}
```

http://holger.dammertz.org/stuff/notes_HammersleyOnHemisphere.html

Sampling der Hemisphäre

Prinzip

- Hammersley-Punkt x_i mit der *hammersley-Funnktion* berechnen
- Abbildung des Punktes x_i auf einen Vektor v_i in der Hemisphäre mit der sampleHemisphereCos-Funktion

Implementierung

- Verteilung entspricht dem cos(Θ) (Nahe der Normalen mehr Samples)
- Vektoren befinden sich im **Tangent Space (z-Achse)**

```
vec3 sampleHemisphereCos (vec2 xi) { // Z ist oben
        float phi = xi.y * 2.0 * PI;
        float cosTheta = sqrt(1.0 – xi.x);
        float sinTheta = sqrt(1.0 – cosTheta * cosTheta);
        return vec3(cos(phi) * sinTheta,
                    sin(phi) * sinTheta,
                    cosTheta);
}
```

Explizite Uniform Locations

- *glGetUniformLocation* muss einen String-Vergleich ausführen, um die Location einer Uniform-Variable zu ermitteln (sollte in der Main-Loop vermieden werden)
- **Locations** können ab OpenGL 4.3 im Shader explizit festgelegt werden
- Ermöglicht Definition einer einheitlichen Schnittstelle für mehrere Shader
- Bei structs und Arrays werden fortlaufende Locations vergeben

Host

```
const int modelMatrixLocation = 0;
glUniformMatrix4f(modelMatrixLocation, 1, GL_FALSE, (GLfloat*)matrix);
```

Device

```
layout (location = 0) uniform mat4 ModelMatrix;
```

Binding Points

- Können genutzt werden, um Texture- oder Image-Units festzulegen
- Keine Zuweisung per glUniform1i notwendig
- Ermöglicht Definition einer einheitlichen Schnittstelle für mehrere Shader

Host

```
// Zuweisung einer Textur auf die Texture-Unit 0
glActiveTexture(GL_TEXTURE0 + 0);
glBindTexture(GL_TEXTURE_2D, mainTexture);

// Zuweisung des Bildes (Level 0 der Textur mainTexture) auf die Image_Unit 0
glBindImageTexture(0, maintexture, 0, GL_FALSE, 0, GL_READ_WRITE, GL_RGBA8);
```

Device

```
// Sampler arbeitet auf Texture_Unit 0
layout (binding = 0) uniform sampler2D mainTexture;

// Zugriff auf Bild auf Image_Unit 0
layout (binding = 0) uniform image2D mainImage;
```

Due Date

16.01.