DEFERRED SHADING & AMBIENT OCCLUSION

Inhaltsverzeichnis

G-BUFFER

G-Buffer

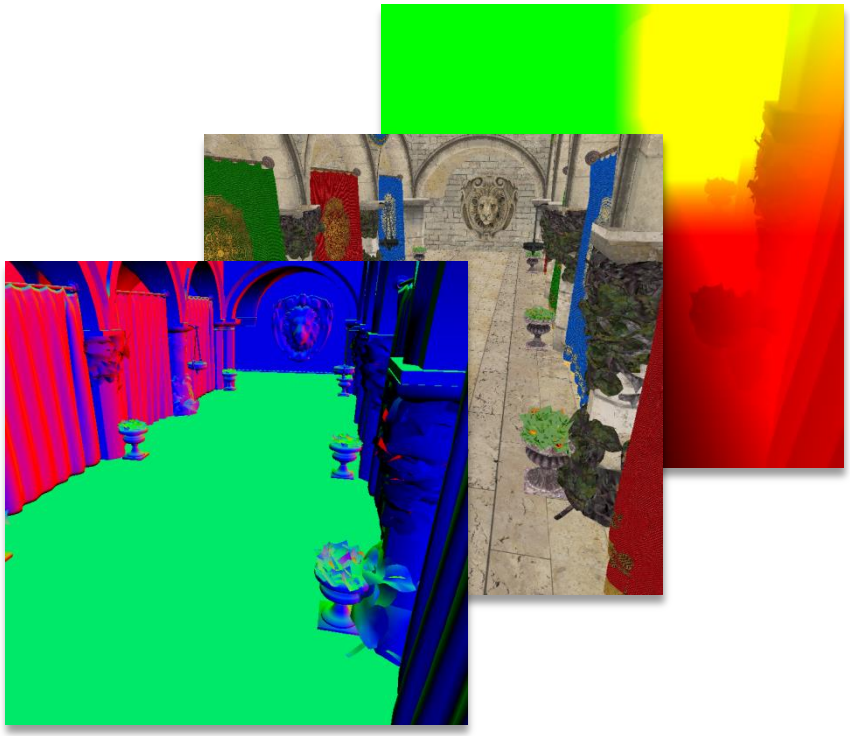
Trennung von Geometrieverarbeitung und Lichtberechnung

1. Renderdurchlauf – Füllen des G-Buffers

- Speichern aller beleuchtungsrelevanten Informationen (Farbe, Position, Normale) in verschiedenen Texturen (render to texture) in einem Schritt

2. Renderdurchlauf – Compositing

- Rendern eines Screen-Filling-Quad
- Beleuchtung pro Pixel mit Daten aus den Texturen



G-Buffer

Möglichkeiten:

Deferred Shading, Filter, SSAO,…

Nachteile:

- Speicherverbrauch des G-Buffers
- viele Texturzugriffe
- Antialiasing nicht ohne weiteres möglich
- Keine Transparenzen möglich

DEFERRED SHADING

Deferred Shading

Normalerweise:

Beleuchtung **jedes** Vertex mit **jeder** Lichtquelle der Szene.

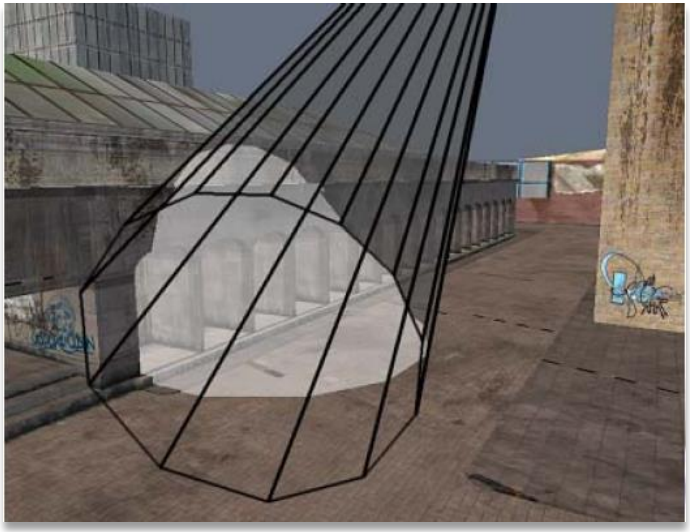
Mit Deferred Shading:

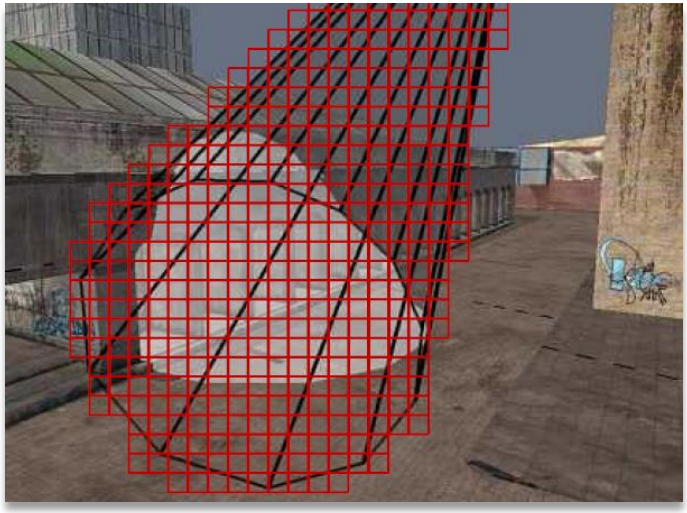
Nur die **sichtbaren** Pixel werden mit den Informationen der Texturen des G-Buffers beleuchtet.

[1]

Wie funktioniert die Beleuchtung?

- Lichtkegel wird durch ein konvexes Polygon approximiert
- Polygon mit aktiviertem Back-Face culling rendern
- Jedes Pixel, welches durch das Polygon überdeckt wird, wird mit Informationen aus den Texturen des G-Buffers beleuchtet
- Bei mehreren sich überlagernden Lichtquellen wird blending verwendet





[2]

Implementation – Vertex Shader:

```
layout (location = 0) in vec4 position;

uniform mat4 modelMatrix;
uniform mat4 viewMatrix;
uniform mat4 projectionMatrix;
out vec3 lightColor;
out vec4 passPosition;
out vec4 midPosition;
out vec4 color;
void main() {
color = vec4( lightColor, 1.0);
passPosition = viewMatrix * modelMatrix * position;
midPosition = viewMatrix * modelMatrix * vec4(0,0,0,1);
gl_Position = projectionMatrix * viewMatrix * modelMatrix
* position;
```

Implementation – Fragment Shader:

```
in vec4 passPosition;
in vec4 midPosition;
in vec4 color;

uniform sampler2D positionMap;
uniform sampler2D normalMap;

layout(location = 0) out vec4 output;

void main() {

vec2 uv = gl_FragCoord.xy/800.0;
vec4 pos = texture(positionMap, uv);
vec3 nor = texture(normalMap, uv).xyz;
float diffuseStr = 2 - length(pos - midPosition);
if ( diffuseStr < 0) discard;
float specularStr = clamp(diffuseStr * diffuseStr, 0, 1);
float diffuse = . . .; cos(phi)
float specular = . . .; cosn( psi)
output = color * (diffuseStr*diffuse + specularStr*specular);
```

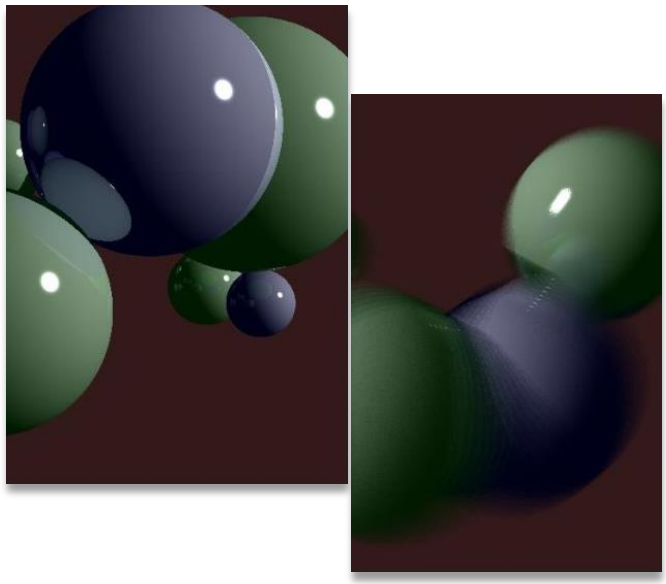
FILTER

Filter

Anwendung von Bildverarbeitungsfiltern auf dem fertigen Bild:

- Glow
- Glättung mit Maske
- Motion Blur…

[3]



[4]

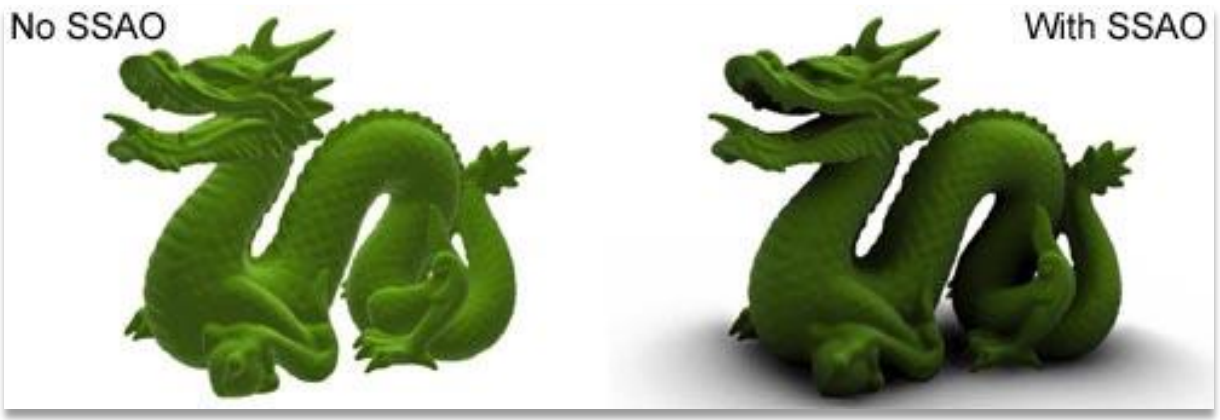
AMBIENT OCCLUSION

Ambient Occlusion

Ambient Occlusion - Umgebungsverdeckung:

In wie weit wird der zu beleuchtende Punkt durch seine Umgebung verdeckt?

- Darzustellende Farbe wird mit einem Faktor zwischen 0 und 1 gewichtet
- Bessere Wahrnehmung von räumlicher Nähe, Volumen und Tiefe
- Einfache globale Beleuchtung



[5]

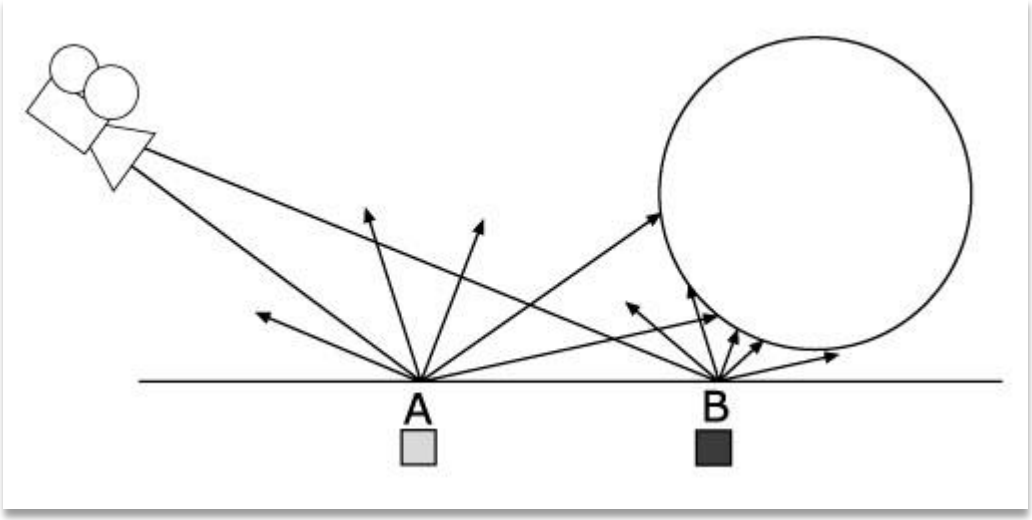
[6]

Ambient Occlusion

Unterscheidung zwischen **Objektraum-** und **Screen Space Ambient Occlusion**

Objektraum – Raytracing Ansatz :

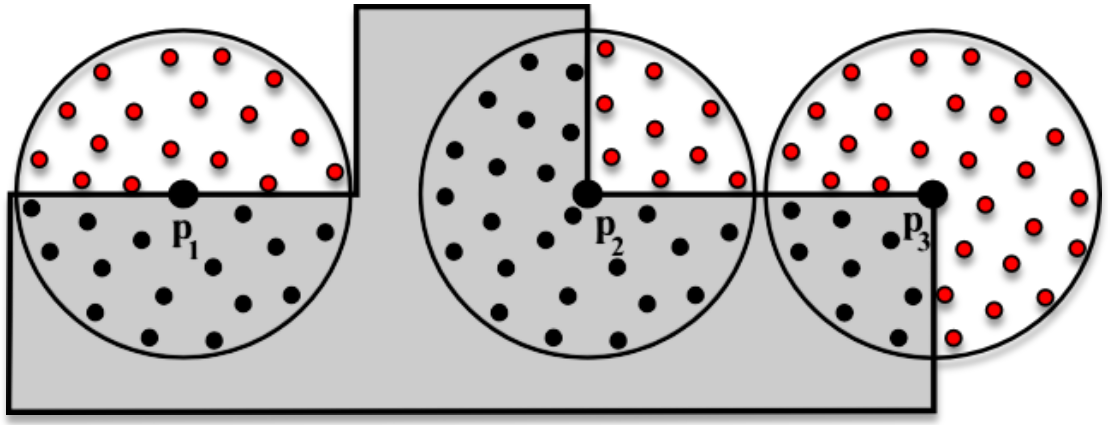
Aussenden von Strahlen vom aktuellen Punkt in die Welt und Überprüfung, wann die erste Geometrie getroffen wird



[7]

Screen Space - Tiefenwerte im Bildraum:

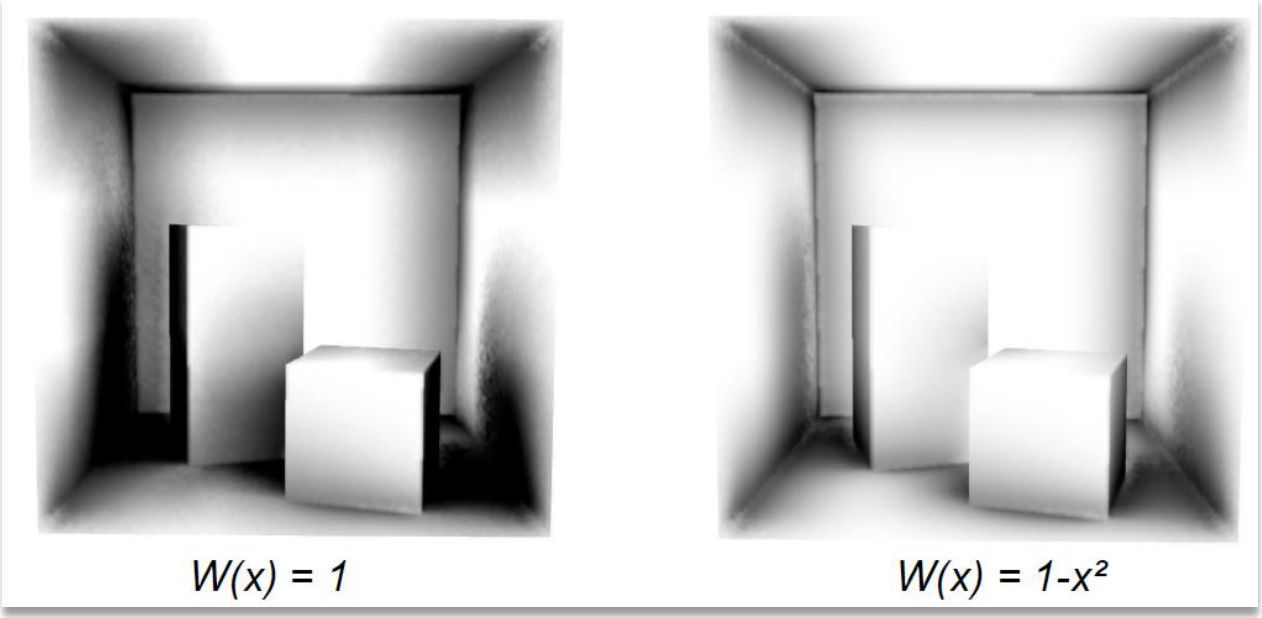
- Bestimme Weltkoordinaten jedes Pixels und addiere **n** Zufallsvektoren darauf
- Rechne Vektoren zurück in Bildschirmkoordinaten → Z-Wert ergibt Tiefenwert
- Vergleiche errechneten Z-Wert mit dem Tiefenwert an der Textur
- Falls errechneter Z-Wert größer als Tiefenwert wird der Punkt verdeckt
- Betrachtet wird nur der Halbraum über dem Punkt



[8]

Erweiterungen – Sampleinterpretation:

- Samples nicht gleich wichten, sondern Abhängig von Abstand zum eigentlichen Punkt
- Quadratische Gewichtungsfunktion: $W(x) = 1 - x^2$

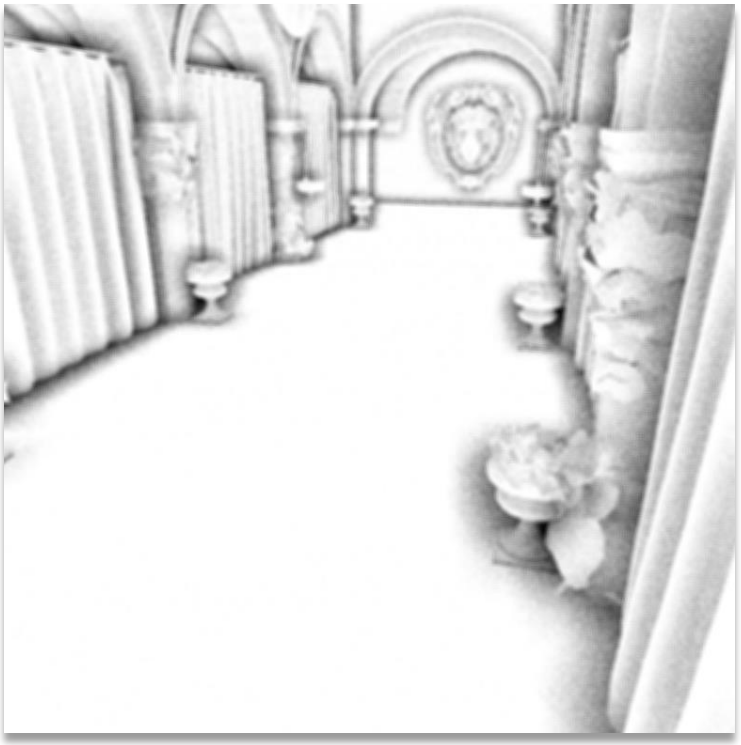


[9]

Ambient Occlusion

Erweiterungen – Filterung:

- Bilinearer Filter auf dem Ergebnis um Rauschen zu verhindern
- Weiterer Shader und weiteres FBO für gefiltertes SSAO Ergebnis benötigt



Ambient Occlusion

Implementation – Vertex Shader

- Durchgeben der benötigten Werte (UV-Koordinate und Position)

Implementation – Fragment Shader

```
in vec2 passUVCoord;

uniform sampler2D positionMap;
uniform sampler2D normalMap;
uniform mat4 sceneProjectionMatrix;
uniform float radius;
uniform float quality;

out vec4 fragmentColor;


float calcOcclusion(vec3 pos, vec3 normal, float d)
{
float occ = 1.0;
for(int i=0; i<quality; i++) {
vec3 v = randomVector();
v = sign( dot( v, normal)) * v * radius;
vec4 diff = sceneProjectionMatrix * vec4( pos+v, 1);
diff = (diff / diff.w)*0.5 + 0.5;
vec4 td = texture( positionMap, diff.xy);
if (td.z >= d) occ -= 1.0/quality;
}
return occ;
}
```

FRAGEN..?

Literatur- & Abbildungsverzeichnis

[CGII]
Müller
Computergraphik 2 2014
Universität Koblenz – Vorlesungsmaterialien

[1]
Abbildung - Deferred Shading City
http://msg4svc.net/static/smithbuck/images/talk_deferredshadingtechnique.jpg

[2]
Abbildung – Lichtkegelapproximation
Müller
Computergraphik 2 2014
Universität Koblenz – Vorlesungsmaterialien

[3]
Abbildung - Motion Blur Filter
http://www.ozone3d.net/public/jegx/201309/glslhacker_shadertoy_antonalog_motion_blur.jpg

[4]
Abbildung - Glow-Filter City
http://http.developer.nvidia.com/GPUGems/elementLinks/fig21-01a.jpg

[5]
Abbildung - AO Dragon
http://hothardware.com/articleimages/Item1497/small_SoDx11-SSAO.jpg

Literatur- & Abbildungsverzeichnis

[6]
Abbildung - AO Teapot
http://www.yaldex.com/open-gl/images/13fig01.jpg

[7]
Abbildung - AO im Objektraum
http://joomla.renderwiki.com/joomla/images/stories/render/tech_ambocc_01.jpg

[8]
Abbildung - SSAO – Sphere
http://blog.evoserv.at/wp-content/uploads/2012/12/ssao_sphere_samples.png

[9]
Abbildung – SSAO - Gewichtung
Müller
Computergraphik 2 2014
Universität Koblenz – Vorlesungsmaterialien