

Was macht den Raytracer so schnell?

Surface Area Heuristic

Traversieren der BVH mit **OpenCL**

Perfect Hemisphere Scattering

Surface Area Heuristic

Problem: Wie teilt man das 3D-Modell auf?

Idee: **Alle** möglichen Split-Punkte an **jeder** Achse mit Kostenfunktion c traversieren, $\min(c)$ verwenden

c richtet sich nach **Flächeninhalt**, der entstehen würde

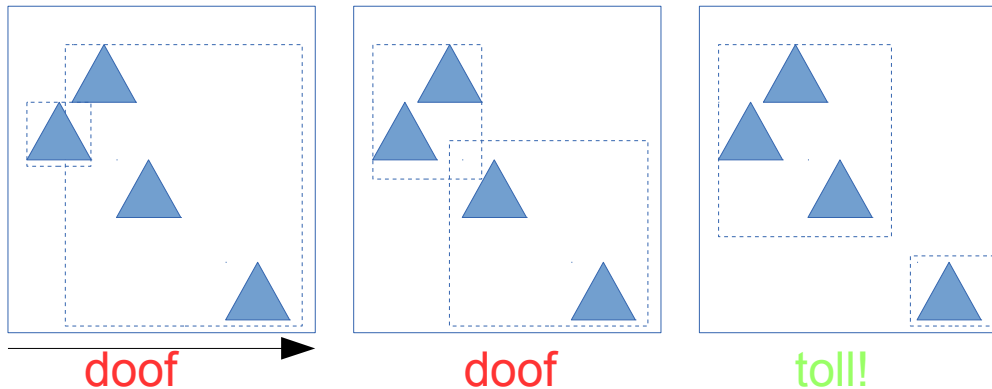
Jede Iteration liefert dann genau **einen** optimalen Punkt, an dem gesplittet werden sollte

Dauert zwar Ewigkeiten, erzeugt jedoch **effiziente** Bäume, da nicht mit x-, y- oder z-Achse gewichtet und wenige Schnitttests (**Flächeninhalt** hängt mit **Schnittwahrscheinlichkeit** zusammen)

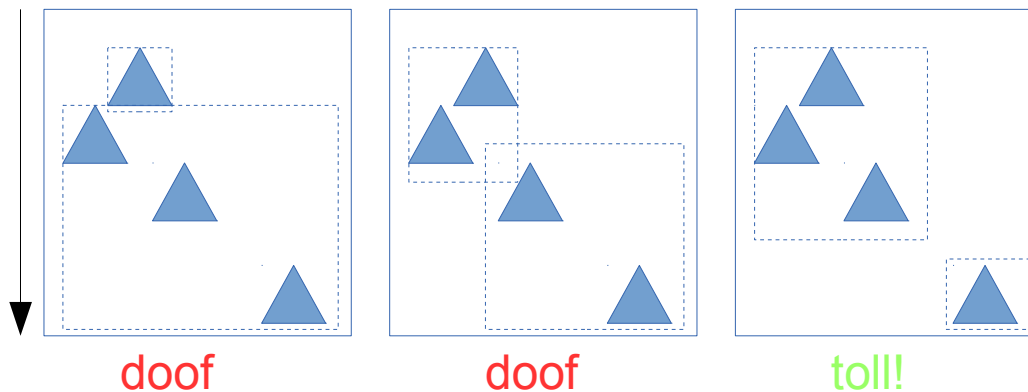
Surface Area Heuristic (SAH)

Beispiel: ...nur in 2D :'(

Sort an **X-Achse**:



Sort an **Y-Achse**:

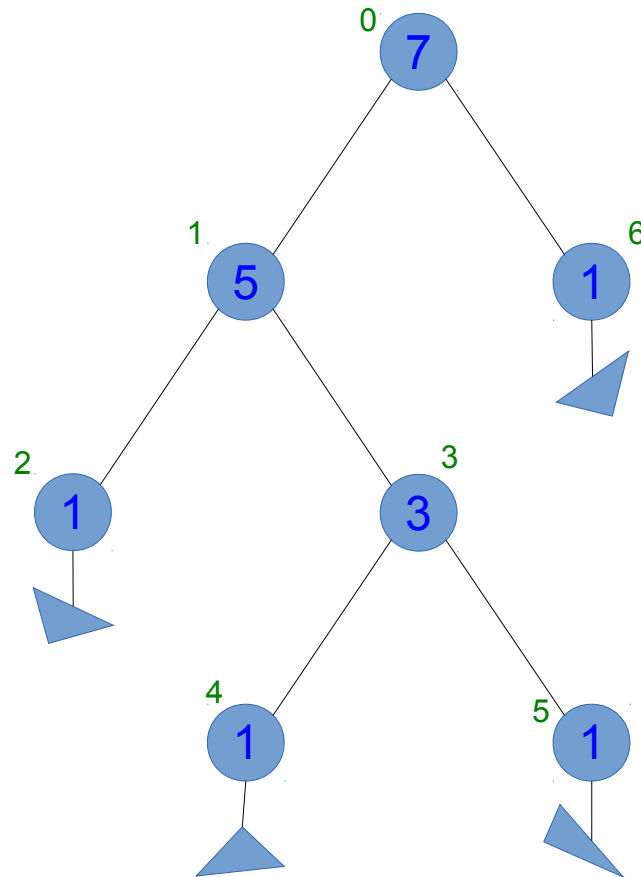


nur zufällig dieselben

→ Speedup von ~1.5 bei Sibenik

Traversierung der BVH in OpenCL

Binärbaum:



NodeCount (blau) repräsentiert die Größe des Unterbaums inklusive sich selbst

Index (grün) repräsentiert die Position des Nodes im Array

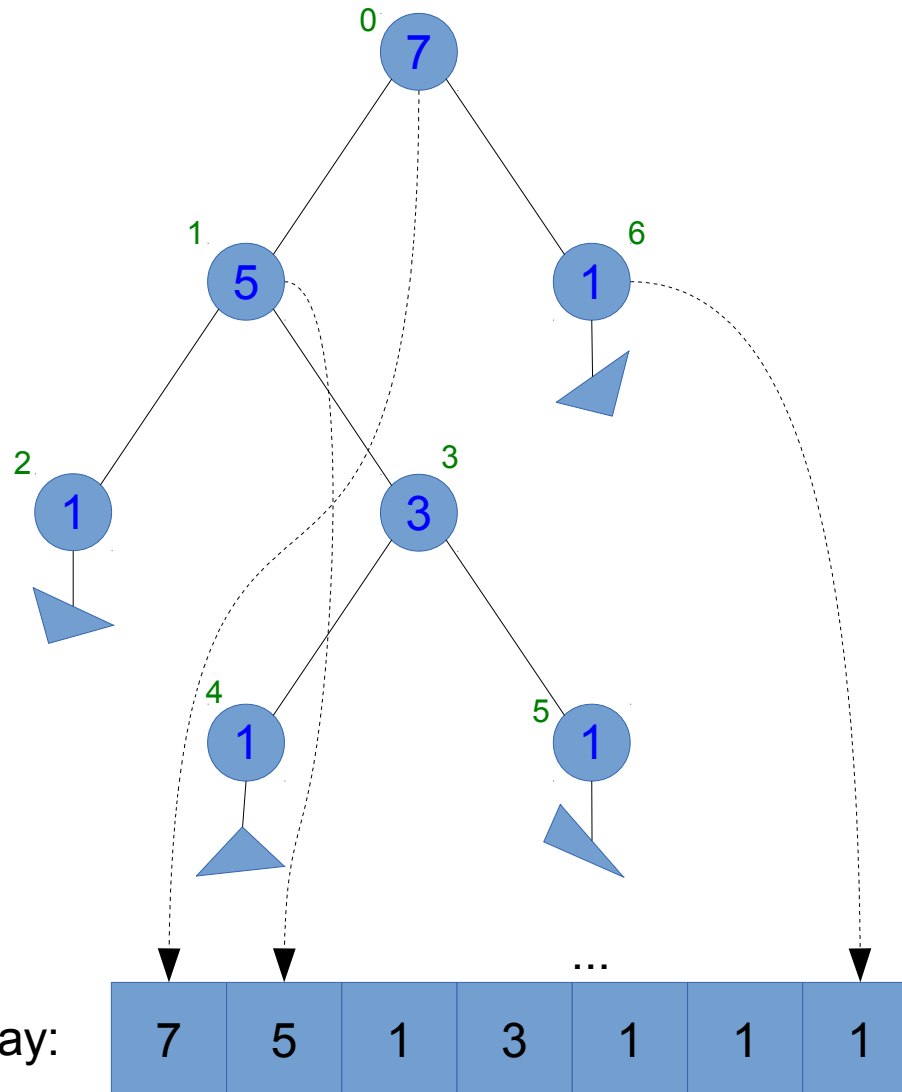
Wir legen fest:
Nur **ein** Dreieck pro Leaf!
→ Berechnungen einfacher

Man erkennt:
Ein Leaf hat immer die Größe 1!

Traversierung der BVH in OpenCL



Binärbaum:



Traversierung der BVH in OpenCL



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Problem:

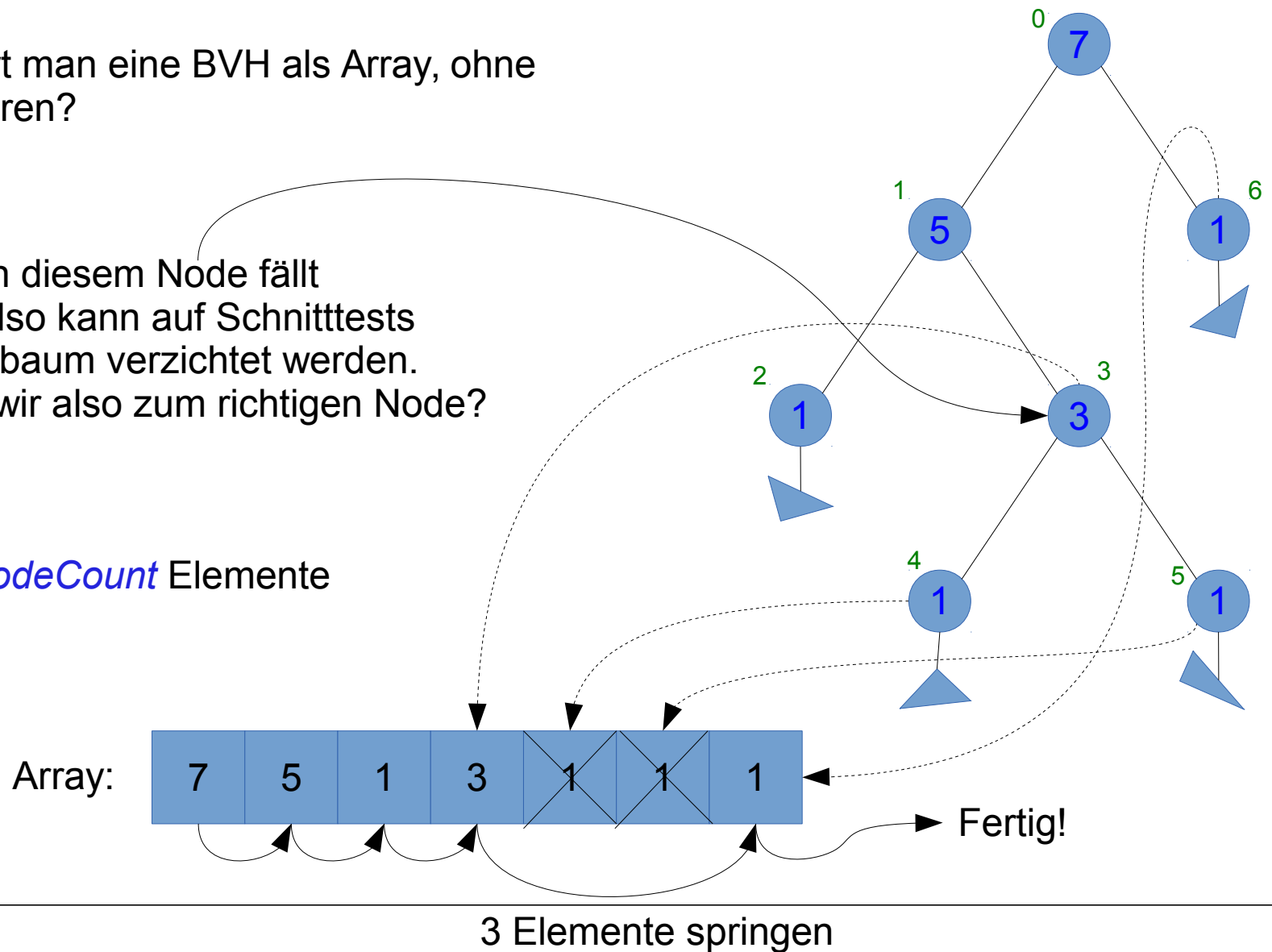
Wie traversiert man eine BVH als Array, ohne explizite Children?

Beispiel:

Schnitttest an diesem Node fällt negativ aus, also kann auf Schnitttests mit dem Unterbaum verzichtet werden. Wie springen wir also zum richtigen Node?

Lösung:

Sprung um *nodeCount* Elemente



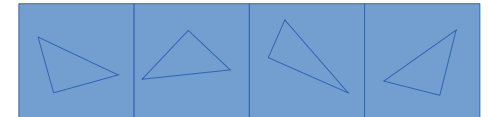
Traversierung der BVH in OpenCL



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Triangles werden in separatem Array gespeichert, weil sonst jeder Node einen Platz für ein Triangle bereithalten müsste

Array der Triangles
= sortierte FaceIDs:



Wenn wir ein Leaf entdecken, inkrementieren wir den Triangle-Index

Problem:

Was passiert, wenn wir einen Unterbaum überspringen?

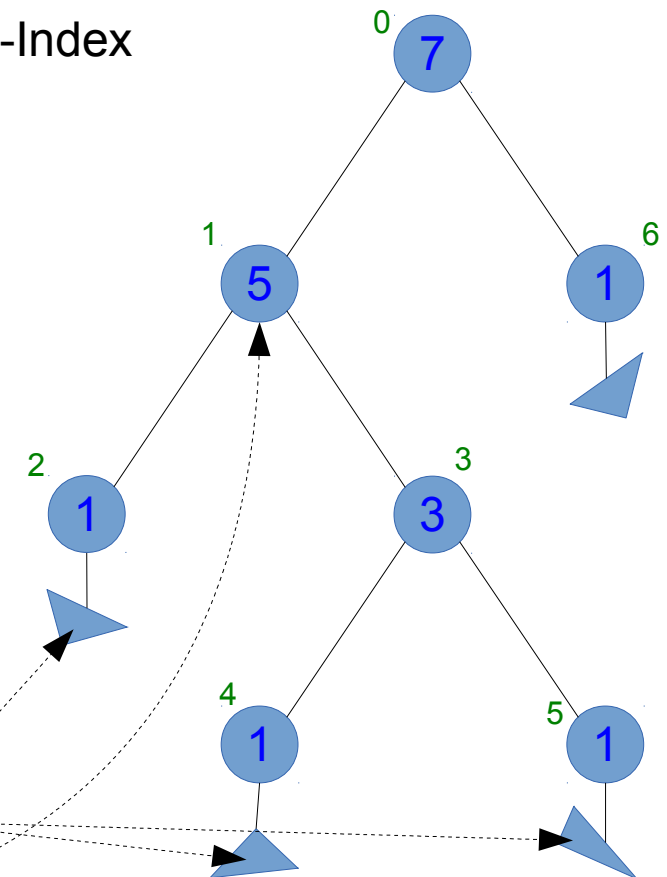
Lösung:

Wir errechnen, wie viele Leaves in dem zu überspringenden Node enthalten sind und addieren sie auf den Triangle-Index:

Anzahl der Triangles pro Node:
 $(\text{nodeCount} + 1) / 2$

Beispiel:

Node #1 hat $(5 + 1) / 2 = 3$ Triangles

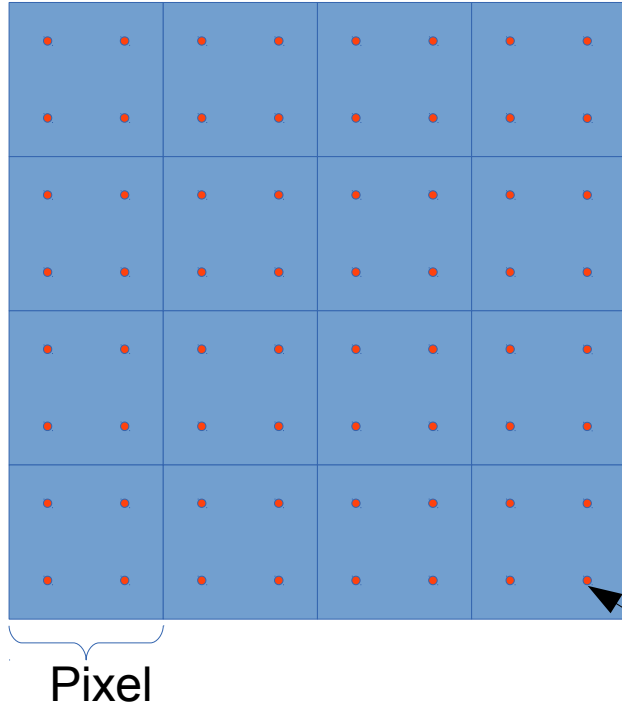


Supersampling



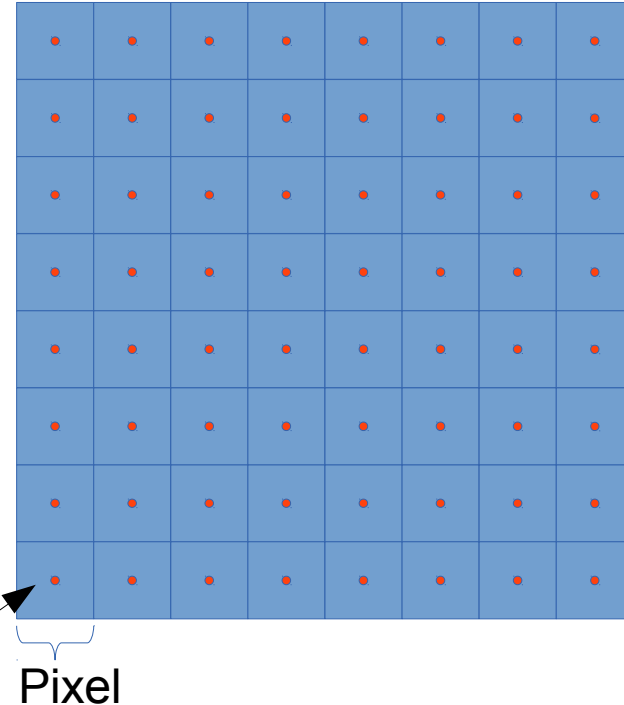
TECHNISCHE
UNIVERSITÄT
DARMSTADT

Normales Supersampling (vierfaches Supersampling)



Vier Subpixel werden berechnet
und dann der Durchschnittswert
berechnet

Zu groß rendern (einfaches Supersampling)



Jeder Subpixel wird als eigener
Pixel betrachtet

- **Bild wird zu groß:** hier 8×8 statt 4×4 px
- Einfach am Ende verkleinern, gleiches Resultat!

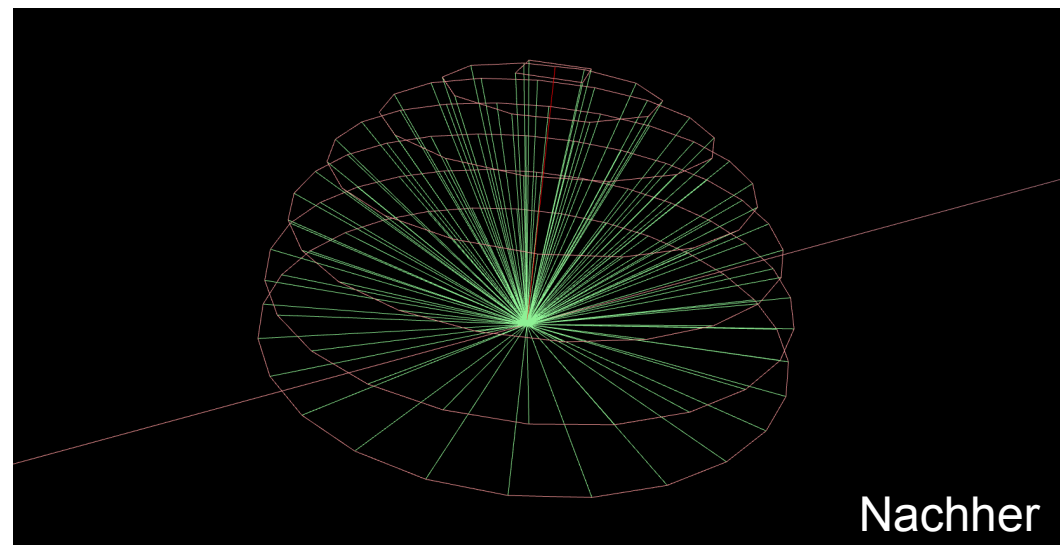
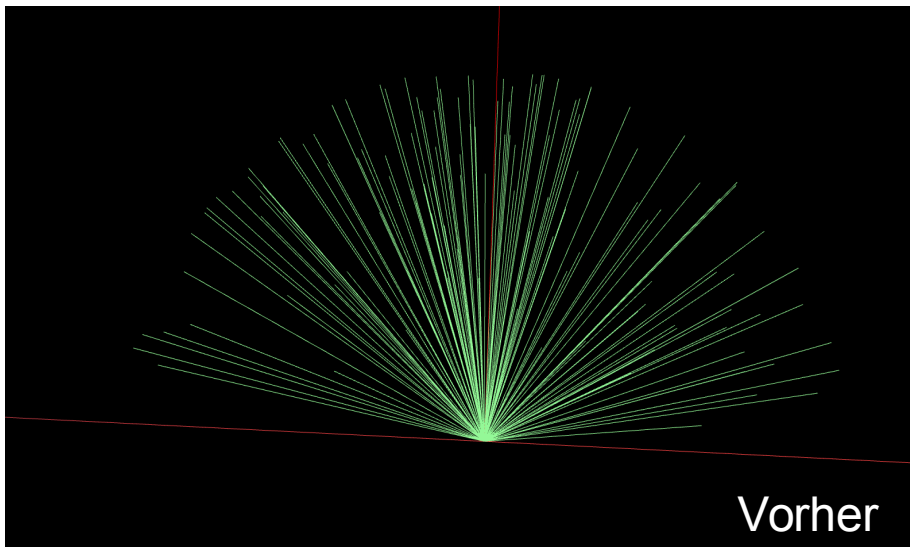
Perfect Hemisphere Scattering

Problem: Hemisphere-Sampler ist lahm und Strahlen werden gewichtet! :(

Grund: Mangelhafte *Pseudorandom*-Implementierung: Muss zunächst einen halbwegs akzeptablen *Randomwert* erzeugen.

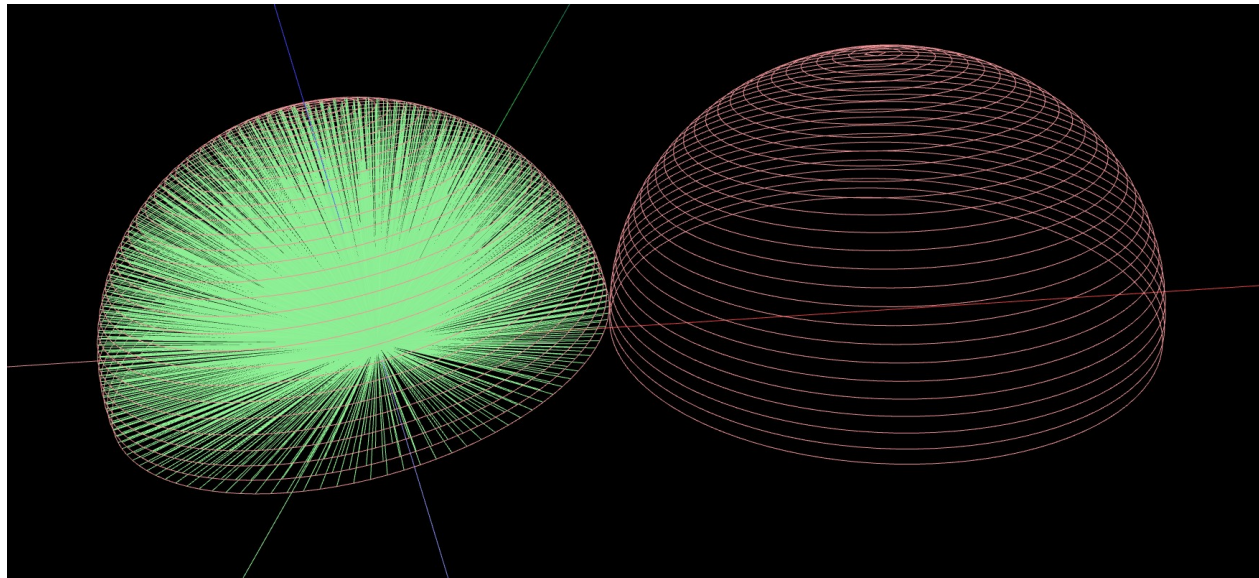
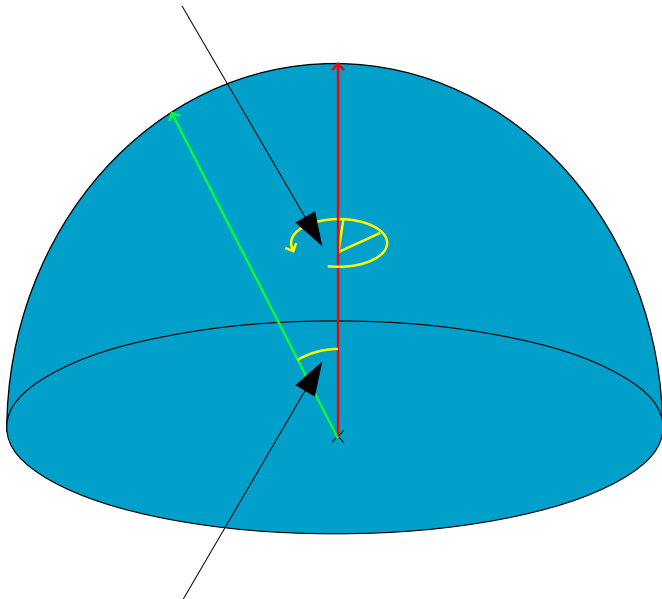
Lösung: Vektoren entlang der Hemisphere gleichverteilen

Durch Ringe parametrisiert, Gewichtung der resultierenden Vektoren ist nun **umgekehrt proportional zur Höhe des Rings** in der Hemisphäre, da untere Ringe einen **höheren Radius haben** (und somit einen größeren Projektionskegel)



Perfect Hemisphere Scattering

- Analyse des mitgelieferten Hemisphere-Samplers:
 - Generiert zwei *Zufallszahlen*, die in zwei Winkel umgerechnet werden:
 - φ : Rotationswinkel um die Normale



- θ : Krümmungswinkel der Normalen in Richtung des Großkreises

Perfect Hemisphere Scattering



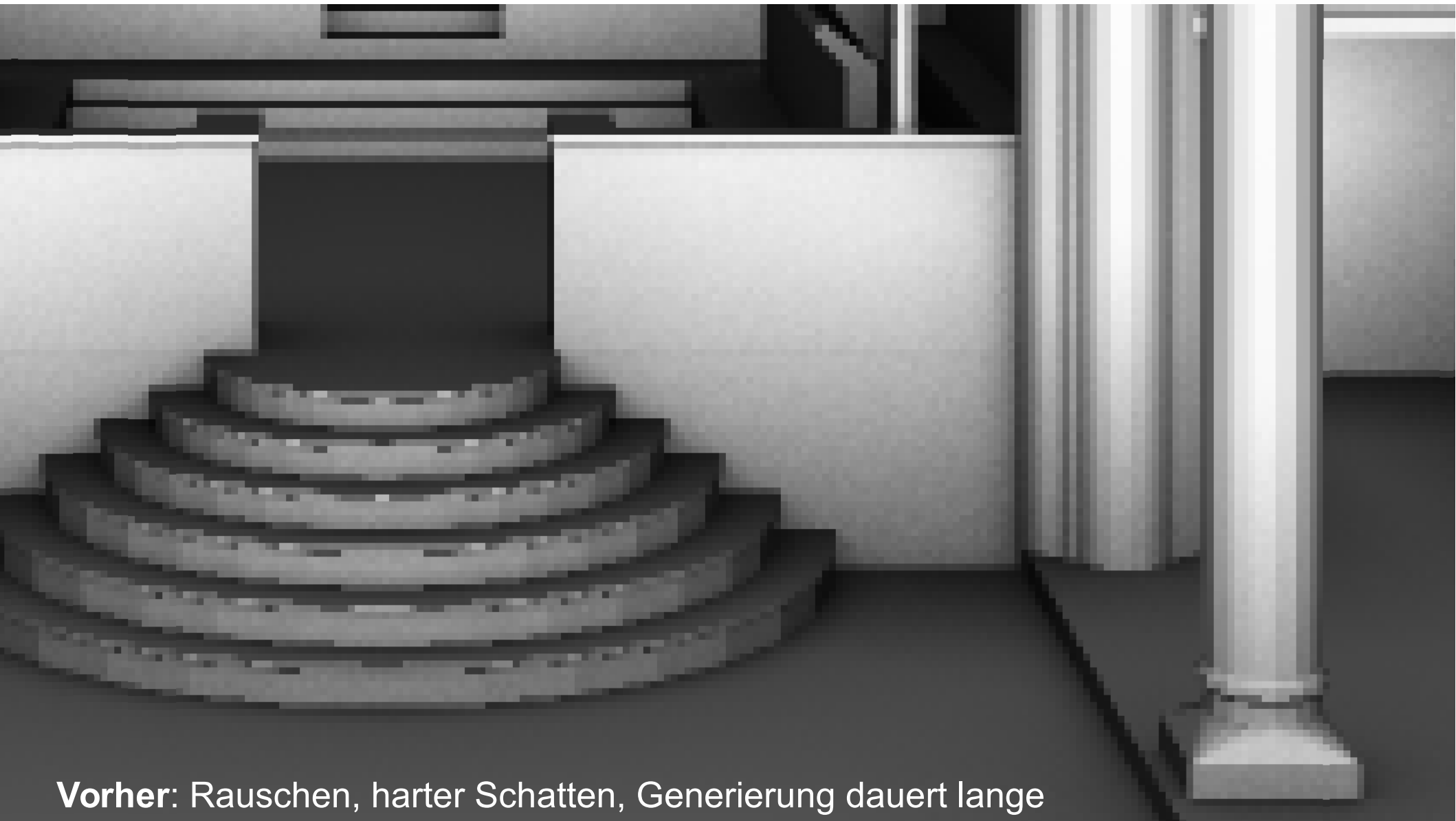
- **Idee:** Algorithmus finden, der φ und θ so bestimmt, dass eine gleichmäßige Hemisphäre entsteht
- → n Ringe R_i mit $1 \leq i \leq n$ in gleichen vertikalen Abständen auf der Hemisphäre stapeln
- → Für jeden Ring k_i Rays v_j mit $1 \leq j \leq k_i$ erstellen (Pseudocode: *rayCount*)

```
for (uint currentCircle = 0; currentCircle < circleCount; ++currentCircle) {  
    // Angle of each vertical step  
    float const stepAngleRad = alpha_max / circleCount;  
    // "Horizontal" angle  
    float const angleRad = (stepAngleRad * currentCircle) + alpha_min;  
    uint const rayCount = (uint) ((2.0f * M_PI * cos(angleRad)) / stepAngleRad);  
    float const theta = M_PI_2 - angleRad;  
    for (uint currentRay = 0; currentRay <= rayCount; ++currentRay) {  
        float const phi = (2.0 * M_PI * currentRay) / rayCount;  
        // Use phi and theta here  
    }  
}
```

Perfect Hemisphere Scattering



TECHNISCHE
UNIVERSITÄT
DARMSTADT



Vorher: Rauschen, harter Schatten, Generierung dauert lange

Perfect Hemisphere Scattering



TECHNISCHE
UNIVERSITÄT
DARMSTADT



Nachher: Kein Rauschen, weicherer Schatten, schnell \(\cdot\cup\cdot\)/

ZEIGT UNS JETZT DEN CODE!!!!111

(ノ◦□◦)ノ (└┐

GitHub:

https://github.com/magcks/opengl_raytracer

Dokumentation:

<http://kdex.de/renderer>

WebGL Hemispheres:

<https://maxvonbuelow.de/share/vecs/>