

Games Engineering

Quake-Style Shooter from Scratch

Meilenstein 3



Bayer, Nico

DePaoli, Fabian

Eggers, Michael

Köhler, Benedikt

Rittenschober, Johann

Hochschule für angewandte
Wissenschaften München

Agenda

1. Demo in Trenchbroom

2. Vorteile von 3DCoat

3. Ergebnis Blender Texturierung

4. Lightmapper

5. Audio

6. Gegner KI

7. Demo

8. Ausblick

9. Quellenangaben

Demo in Trenchbroom

Vorteile von 3DCoat



- Nahtloses Painting: Intuitiv und ohne sichtbare Nähte.
- PBR-Workflow: Optimiert für realistische Texturen.
- Smart Materials: Automatisch reagierende Materialien.
- UV-Automatisierung: Schnelle und hochwertige UV-Layouts.

Ergebnis Blender Texturierung

Demo in Blender

Lightmapper

Ziele für Meilenstein 3

- Verbessertes UV-Mapping
- Patch Processing Pipeline
- Bugfix Renderer
- Dynamic Patch Resolution

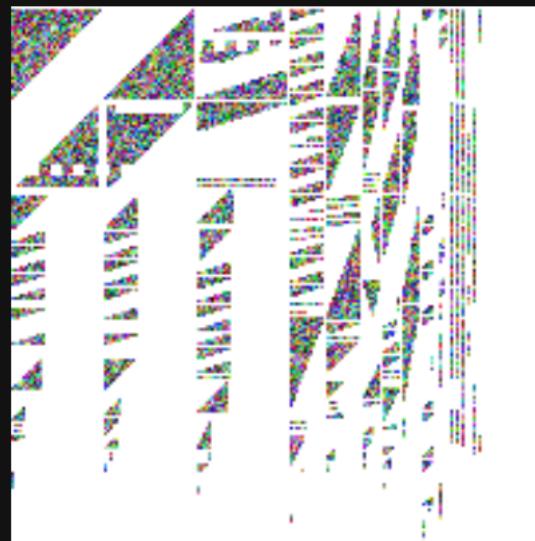
Ziele für Meilenstein 3

- Verbessertes UV-Mapping
- Patch Processing Pipeline
- Bugfix Renderer
- Dynamic Patch Resolution

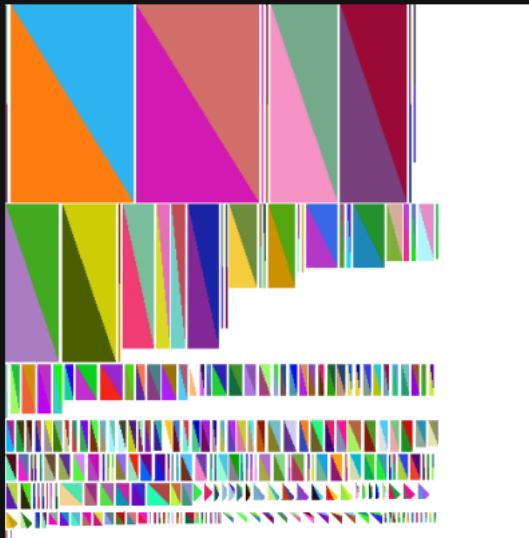
Ziele für Meilenstein 3

- Verbessertes UV-Mapping
- Patch Processing Pipeline
- Bugfix Renderer
- Dynamic Patch Resolution
- Verschiedene Bugfixes
- Point Lights
- Integration in die Engine

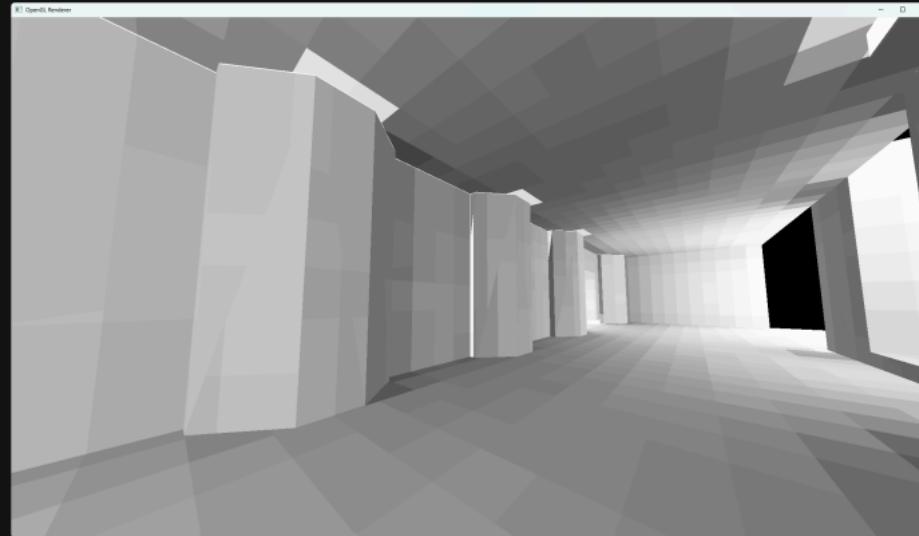
UV-Mapping



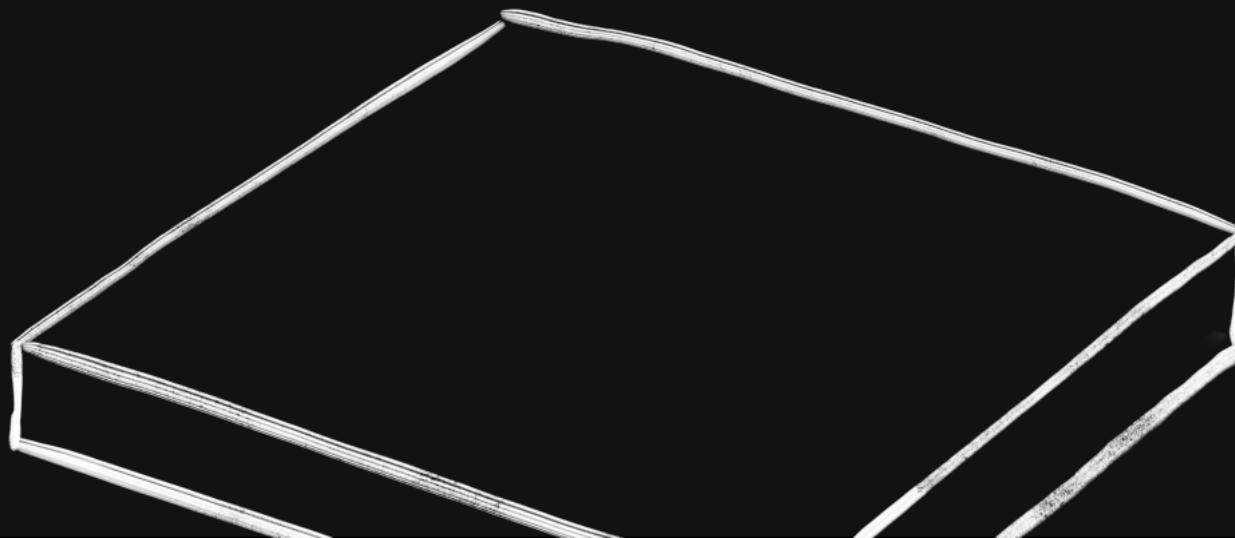
UV-Mapping



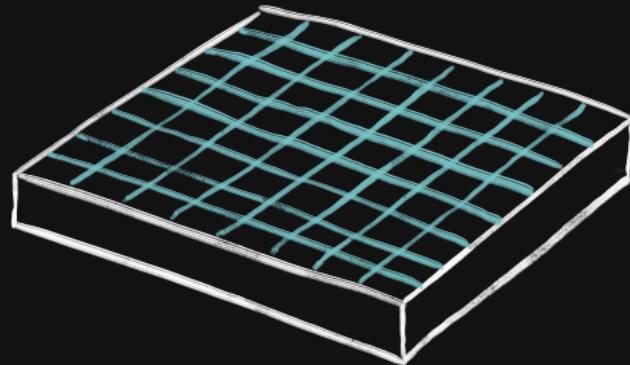
Patch Processing Pipeline: Unzulässige Patches



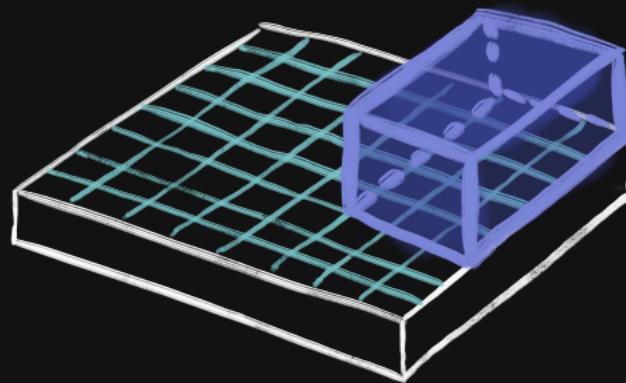
Patch Processing Pipeline: Unzulässige Patches



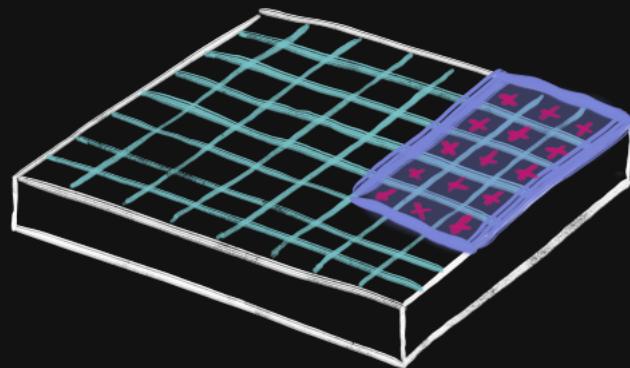
Patch Processing Pipeline: Unzulässige Patches



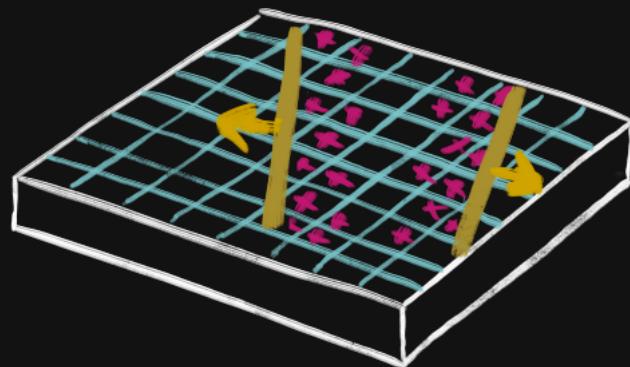
Patch Processing Pipeline: Unzulässige Patches



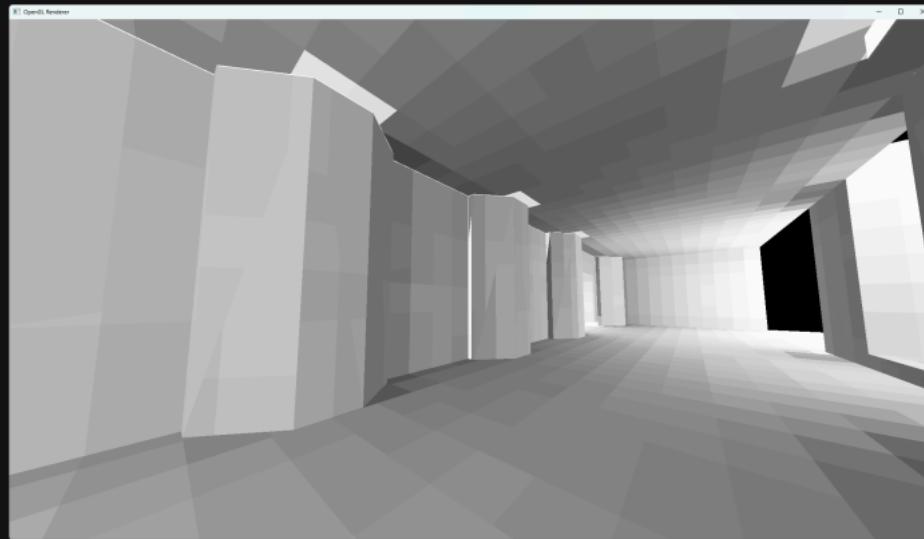
Patch Processing Pipeline: Unzulässige Patches



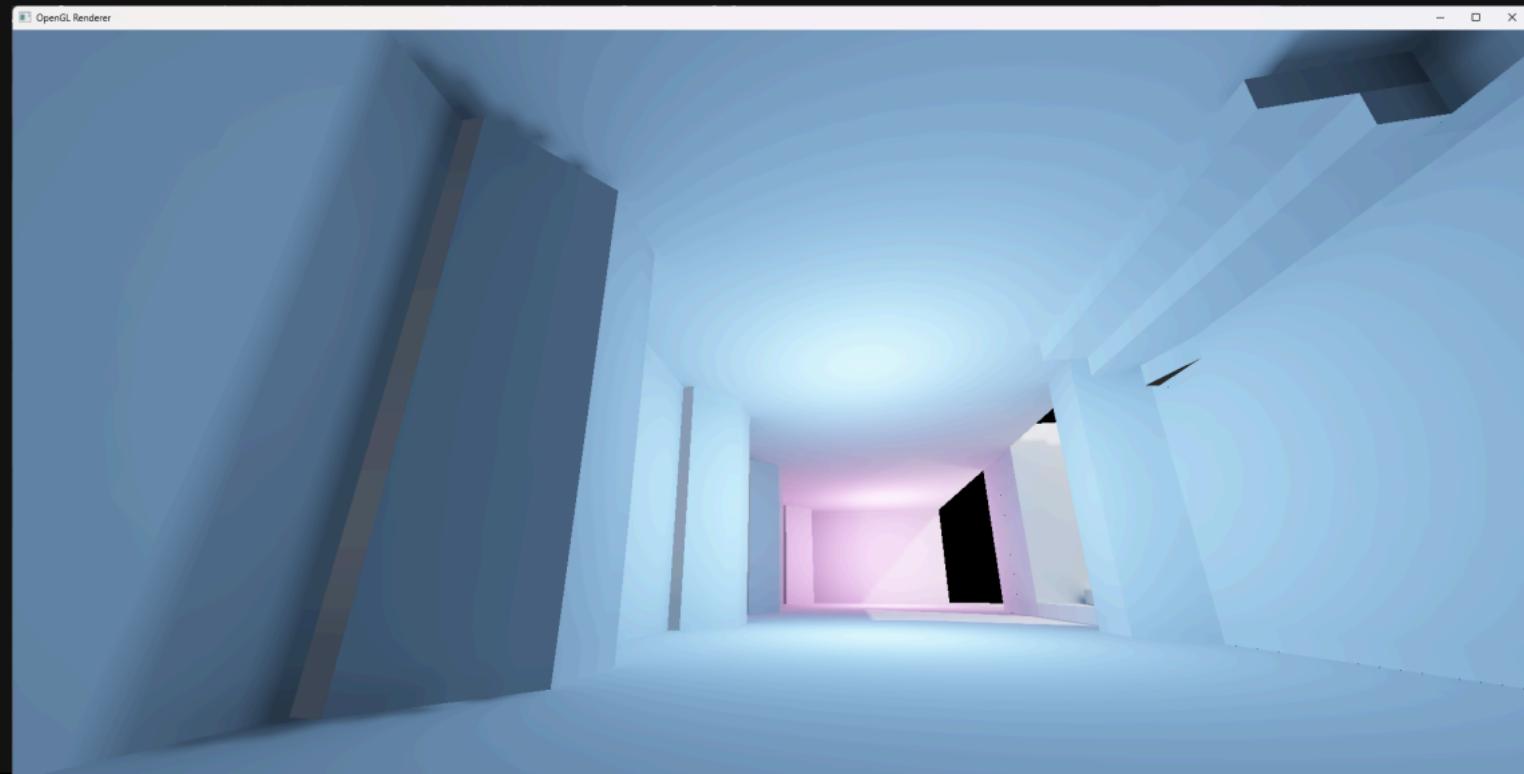
Patch Processing Pipeline: Unzulässige Patches



Aktueller Stand



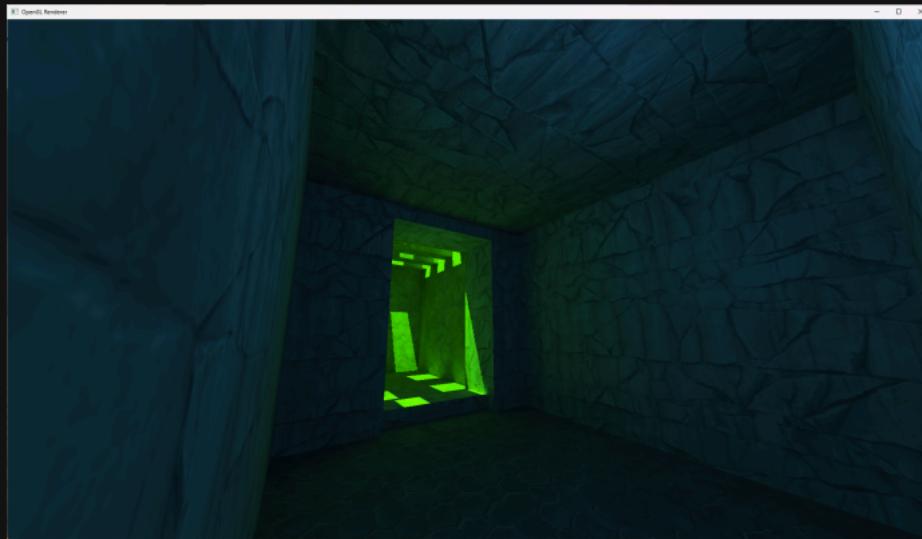
Aktueller Stand



Aktueller Stand



Aktueller Stand



Ziele für Meilenstein 4

- Python Skript zu C++ Programm
- Dynamic Patch Resolution und Multiprocessing
- Bugfixes
- Compute Shaders

Audio

Audio (Management)

Konfigurierbares Mixing über mehrere Audio-Busse (über Konsole änderbar):

```
Audio::m_SfxBus  
Audio::m_AmbienceBus  
Audio::m_MusicBus
```

Globales `AudioSource` Management:

```
Audio::LoadSource("sfx/footsteps.wav");
```

Wird einmalig in Memory geladen, dann von mehreren Entities/Entity-Instanzen referenzierbar

Audio (2D Sounds)

- einfach zu verwenden
- für Musik, Ambience und Player SFX

```
Audio::m_AmbienceBus.play(*m_Ambience, -1);
```

```
if ( m_PrevCollisionState == ES_ON_GROUND ) {
    if ( m_WantsToJump ) {
        Audio::m_SfxBus.play(*m_SfxJump, -1);
        printf("Jumping....\n");
    }
}
```

Audio (3D Sounds)

- für Environment und Entity SFX
- die Audio-Engine (bzw. diverse Effekte) muss mit korrekter Skalierung konfiguriert werden:

```
constexpr float DOD_COORD_UNIT_FACTOR = 37.65f; // derived from quake units
```

- Attenuation bestimmt, wie Sounds mit Entfernung leiser werden:

```
audioSource.set3dAttenuation(DOD_AUDIO_ATT_MODEL, DOD_AUDIO_ATT_ROLLOFF);  
audioSource.set3dMinMaxDistance(DOD_AUDIO_MIN_DIST, DOD_AUDIO_MAX_DIST);
```

Audio (3D Sounds)

- Sound wird als 3D abgespielt, die Position muss ggf. kontinuierlich aktualisiert werden:

```
if ( Audio::m_Soloud.isValidVoiceHandle(m_SfxHandle) ) {  
    Audio::m_Soloud.set3dSourcePosition(m_SfxHandle, pos.x, pos.y, pos.z);  
    Audio::m_Soloud.set3dSourceVelocity(m_SfxHandle, vel.x, vel.y, vel.z);  
    Audio::m_Soloud.update3dAudio();  
} else {  
    m_SfxHandle =  
        Audio::m_SfxBus.play3d(*m_Sfx, pos.x, pos.y, pos.z, vel.x, vel.y, vel.z);  
}
```

Audio (3D Sounds)

- Die Position des Listeners (Player) muss ebenfalls für die 3D-Berechnung aktualisiert werden:

```
Audio::m_Soloud.set3dListenerPosition(position.x, position.y, position.z);
Audio::m_Soloud.set3dListenerAt(forward.x, forward.y, forward.z);
Audio::m_Soloud.set3dListenerVelocity(velocity.x, velocity.y, velocity.z);
Audio::m_Soloud.set3dListenerUp(WORLD_UP.x, WORLD_UP.y, WORLD_UP.z);
Audio::m_Soloud.update3dAudio();
```

Audio (Dynamische Sounds)

- Sounds mit unbestimmter Länge (z.B. Türen)
- wird durch Events aktiviert/deaktiviert
- zusammengesetzt aus Basis-Loop und einem Abschluss-Effekt am Ende



```
class DynamicSound {  
public:  
    DynamicSound(SoLoud:: AudioSource* loop, SoLoud:: AudioSource* end);  
    void Begin3d(glm::vec3 pos, glm::vec3 vel, double loopFadeIn = 0.5);  
    void End3d(glm::vec3 pos, glm::vec3 vel, double loopFadeOut = 0.5);
```

Gegner KI

Wie nehmen wir unsere Umgebung wahr?

- Organismen interagieren mit der Welt durch ihre **Sinne**.
- Die Sinne helfen uns
 - zu überleben
 - uns anzupassen
 - und Entscheidungen zu treffen

Wie nehmen wir unsere Umgebung wahr?

- Interne und Externe Sinne (z.B. Hunger und Sicht)
- Fun fact: Es gibt Schlangenarten die Infrarot sehen können. (1)
- **Frage:** Wie würde unser Leben aussehen wenn wir Infrarot sehen könnten?



Bildquelle: <https://screenrant.com/dune-2-austin-butler-feyd-rautha-black-white-explained/>

Wie funktioniert Sehen überhaupt?

- Sicht beinhaltet das wahrnehmen und umwandeln von Licht in Information.
- Stäbchen und Zapfen auf der Netzhaut
- Wie modelliert man Sicht aber nun performant?

Unser Sicht Modell

Was nimmt die Entity wahr?

- **Distanz:** Wie weit ist ein Objekt entfernt?
- Die **Z-Position** des Objekts im Raum.
- **Winkel:** Relative Ausrichtung des Objekts zur Sichtlinie.

später noch: **Verdeckung** durch Level Geometrie

Demo

Aussicht - Weitere Sinne

- Hören -> Impuls der alles in einem Radius benachrichtigt
- Riechen -> Gerüche in HalfLife sind quasi Sounds mit einem Flag
- Fühlen -> physics system mit collision detection

Ausblick

Herausforderungen

- Collision Detection: Schlechte Performance. **Alle** Entities prüfen jedes Frame auf Kollision mit **gesamter Welt**. Unterteilung in Octree würde Abhilfe schaffen.
- Collision Detection: Nicht framerate unabhängig.
- Animationssystem: Schlechte Performance. Das Aufbauen der aktuellen Pose ist im Moment sehr teuer. Wahrscheinlich schlechte Implementierung. Optimierung über Compute Shader wäre möglich.
- Entity System: Wir überlegen auf ein **Actor-Component** Modell wie in Unreal Engine umzustellen, um den Code innerhalb einer Entity

Zu implementierende Features

- First Person Camera
- HUD Rendering für Lebensanzeige und Fadenkreuz
- Schießen auf Gegner
- Gegner AI weiter ausbauen (Navigation durch Welt, Interaktion mit Spieler)
- Eigene Gegner-Modelle
- Audio

Quellenangaben

- 1. Sichert AB, Friedel P, van Hemmen JL. Snake's Perspective on Heat: Reconstruction of Input Using an Imperfect Detection System. *Phys Rev Lett* [Internet]. 2006 Aug;97(6):068105. Available from: <https://link.aps.org/doi/10.1103/PhysRevLett.97.068105> (22)