HPI Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

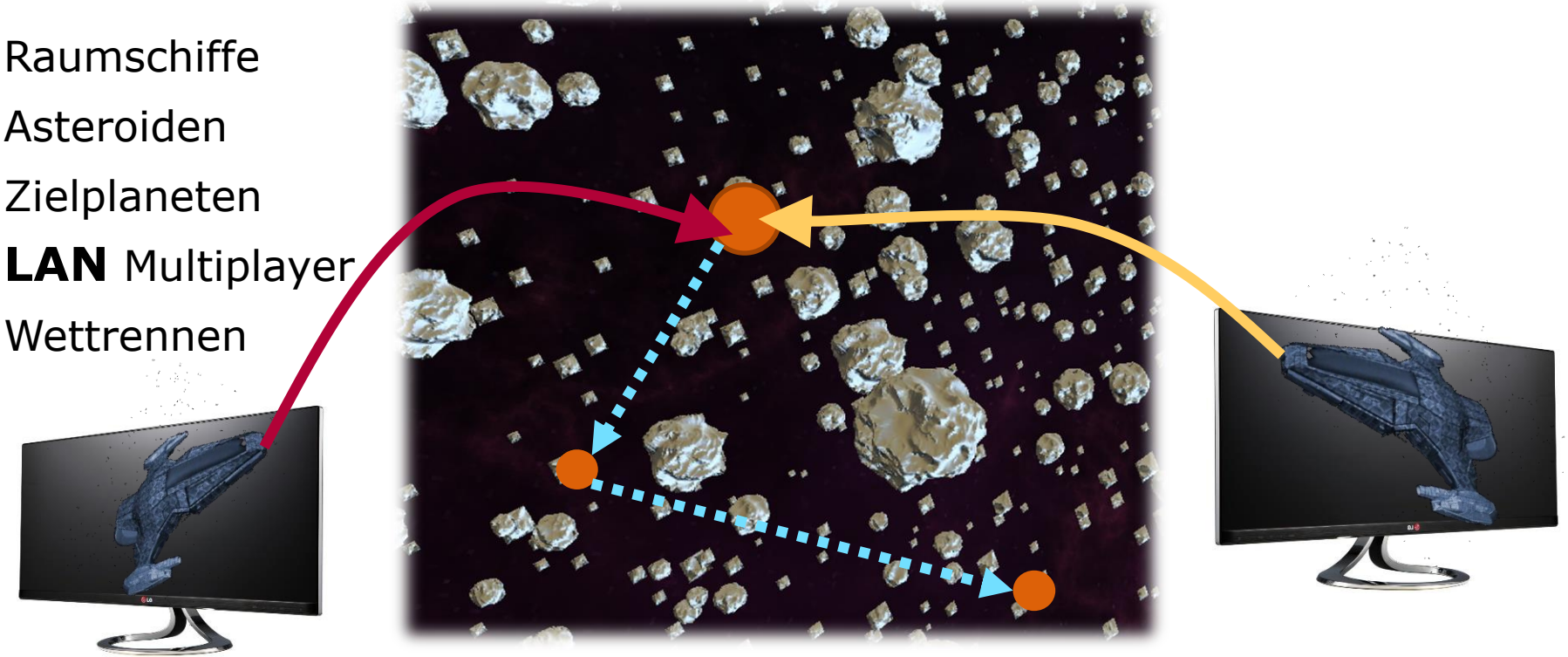
Gameprogramming WS2013/14

„Futurella" von Pavel Belskiy und Felix Niemeyer

Betreuer: Stefan Buschmann

“Futurella” Spielprinzip & Demo

- Raumschiffe
- Asteroiden
- Zielplaneten
- **LAN** Multiplayer Wettrennen



Features

- → Szenengraph mit **OSG**
- → UI mit **cegui**
- → Hintergrundsound mit **fmod**
- → Netzwerk mit **Boost.Asio**
- → Physik mit **Bullet**
- → Planeten Geometrie mit Tesselation-Shader
- → Entfernte Objekte + Skybox => Cubemap
- → Globales Licht als Light-Cubemap
- → Deferred Shading

UI mit cegui

- class CeguiDrawable : public osg::Drawable
- CeguiDrawable::passEvent(…)

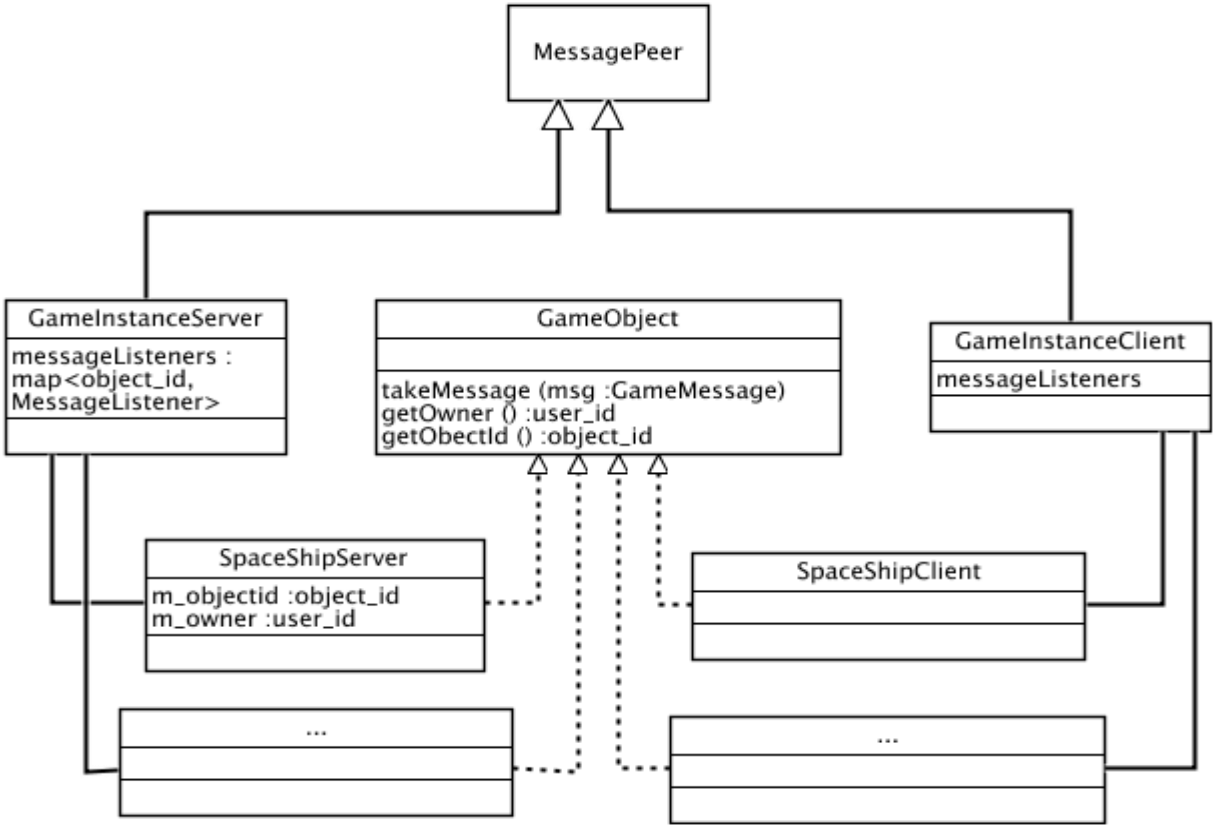
- Wir benutzen cegui für:
  - Konsole
  - (Menus)

Noch nicht implementiert, Grafiken existieren bereits

U.a. um Server zu erstellen und Servern beizutreten

Netzwerk mit Boost.Asio

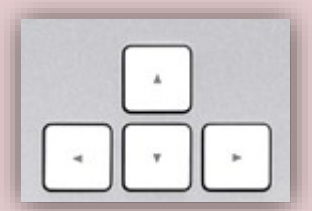
Server - Client Architektur

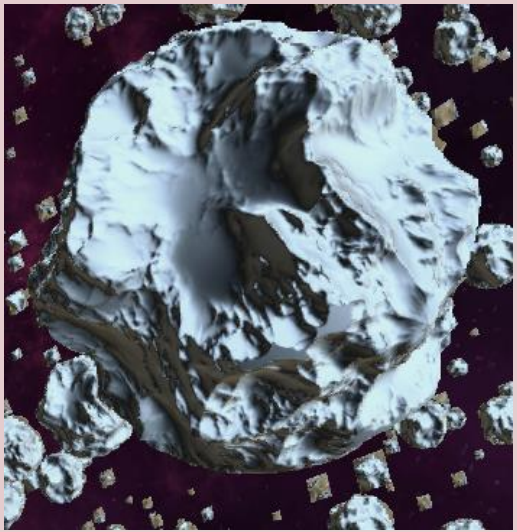


Server - Client Architektur

GameInstanceServer	GameInstanceClient

- Physik
- Weltengenerierung
- Game Logik
  - Punktestand
  - Zielposition
	- Rendering
	- User Input





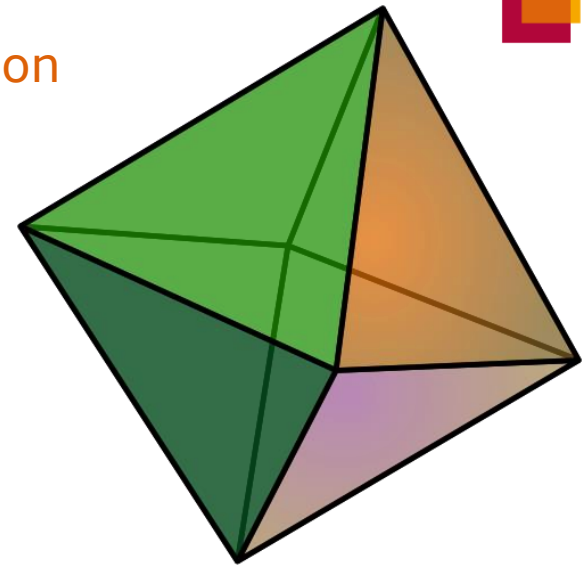
Physik mit Bullet

Zur Kollisionsbehandlung von
- Raumschiff – Planet
- Raumschiff – Raumschiff

Approximation der Planeten als Kugeln
- Sichtbare und “spürbare” Ungenauigkeiten bei Kollisionen

Physikberechnungen nur auf dem Server
- Einfacher
- Lineare Bewegungsinterpolation auf Clientseite könnte ergänzt werden

Planeten Rendering Tesselation



→ Ausgangsgeometrie: **Oktaeder**

- Tesselationslevel bestimmen
- Dreiecke unterteilen
- Zur Kugeloberfläche extrapolieren
- Abstand zum Mittelpunkt modifizieren (Heightmap)

```
out GeometryOut {
    vec2 vTexCoord;
    vec3 vPos;
    float texNumber;
};
```

Planeten Rendering

Hight & Normal Map:

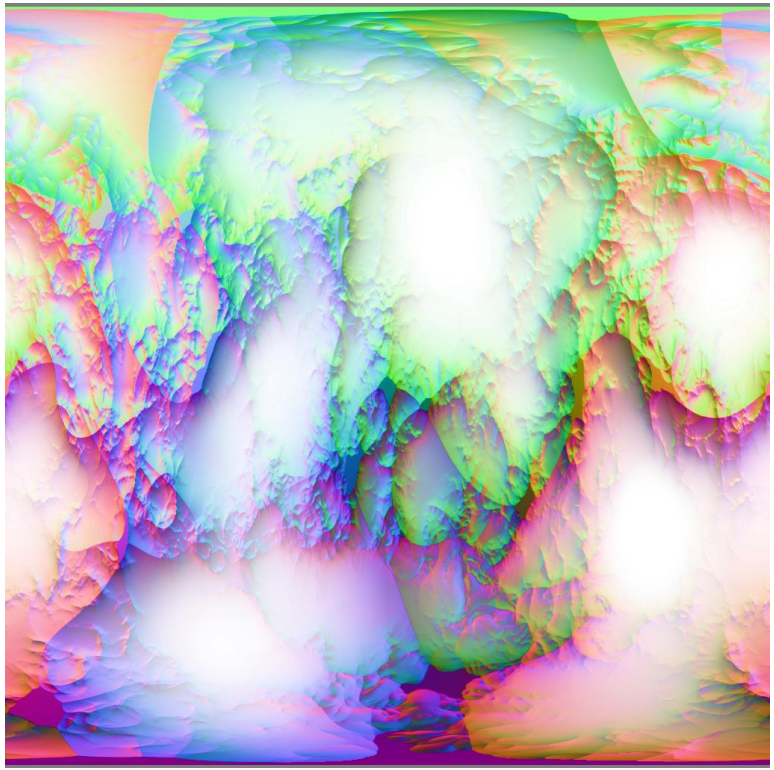
- Generierung mit Qt App
- Höhe im Alpha Kanal
- Normalen als RGB

15 Verschiedene Maps + Rotation

- → kaum Wiedererkennung

512x512

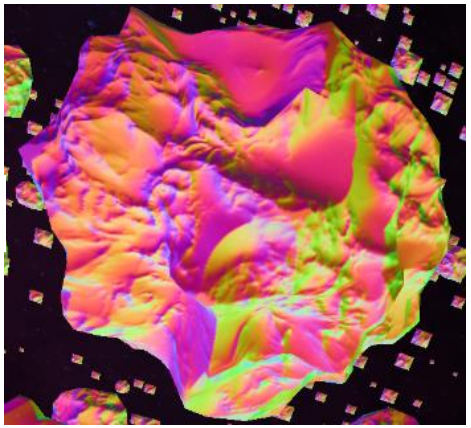
(u,v) = (latitude, longitude)



Planeten Rendering

→ Verwende normal maps (RGB)

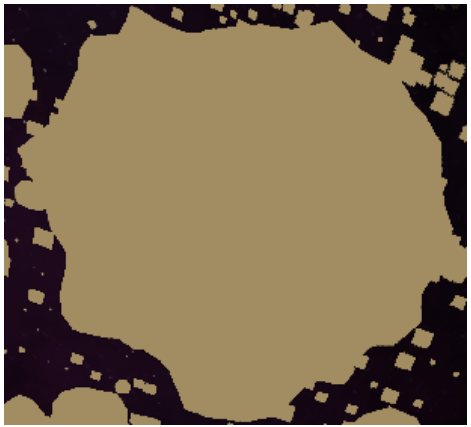
```
layout(location = 1) out vec3 fragNormal;
```



→ Irgendeine Farbe

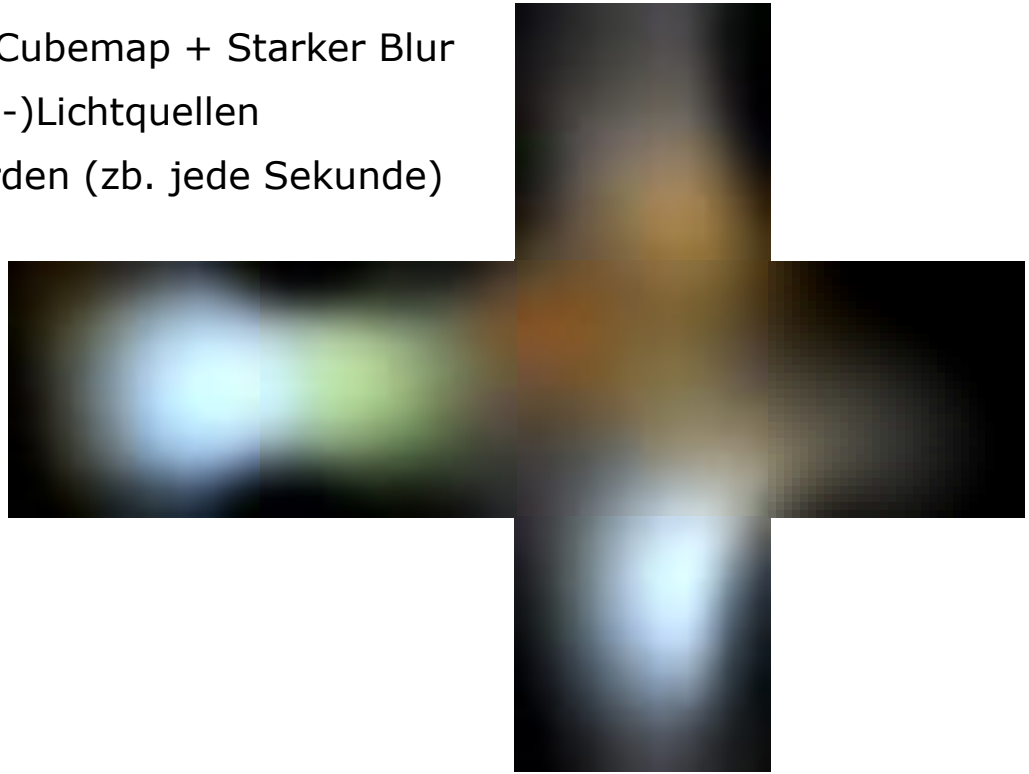
- Texturierung möglich? (Dreiecke kacheln)

```
layout(location = 0) out vec4 fragColor;
```



Beleuchtung mit Lightcubemap

- Statt Punktlichtquellen
- Berechnet sich aus der Environment-Cubemap + Starker Blur
  - Simuliert viele entfernte (Flächen-)Lichtquellen
  - Könnte dynamisch berechnet werden (zb. jede Sekunde)

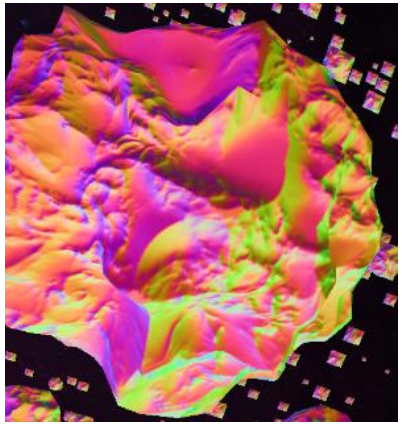


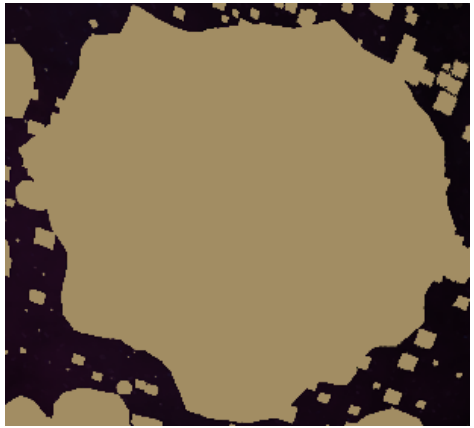
Deferred Shading

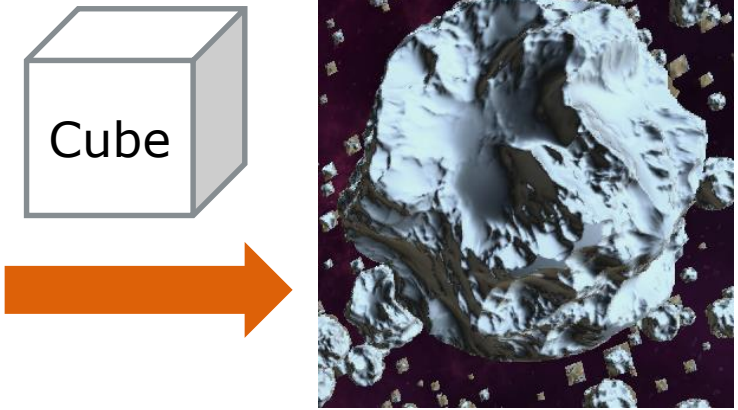
→ Layers: Farben, Normalen, (Materialien)

→ Screen Quad Fragmentshader

- Normalen + Lightcubemap für diffuse und spekulare Beleuchtung







color * texture(lightEnvMap, normal)

Geometrie Management mit chunks

→ Chunks

- Würfel mit gleicher Seitenlänge
- Werden serverseitig generiert, wenn ein Client Geometrie in neuem Bereich anfordert
- Nur **nahe** Chunks werden tesseliert

→ Transform Feedback

- Tesselationsergebnis werden im VRAM gespeichert
- Tesselation muss nicht jeden Frame durchgeführt werden → Performance++
- Hoher Speicheraufwand

Das wichtigste geschafft! ☺

COULD (Powerups, Item Shop, Spaceship upgrades, Player profiles) ☹

SHOULD

- Sound (fmod) 😐
- Shader & Postprocessing (osg::PPU) 😐

MUST

- Asteroidenfeld & Ziel ☺
- (Szenengraph OSG) ☺
- Raumschiffsteuerung & UI (cegui) ☺
- !!! LAN-Modus (asio) ☺

Der Prozess

Beobachtet:

- Ungleichgewicht im Knowhow = Herausforderung

Gelernt:

- Klare Zielvorstellung sehr wichtig
- Realistische und konkrete Feature Planung wichtig

- Man kann immer noch mehr gute Ideen haben