

4. Computergraphik Übung:

Komplexe Geometrien & Lichter

Heni Ben Amor

Arbeitsgruppe Virtuelle Realität und Multimedia

TU Bergakademie Freiberg

Aufgabe 1a: Geometrie

- Lösung siehe `loesung.cpp`

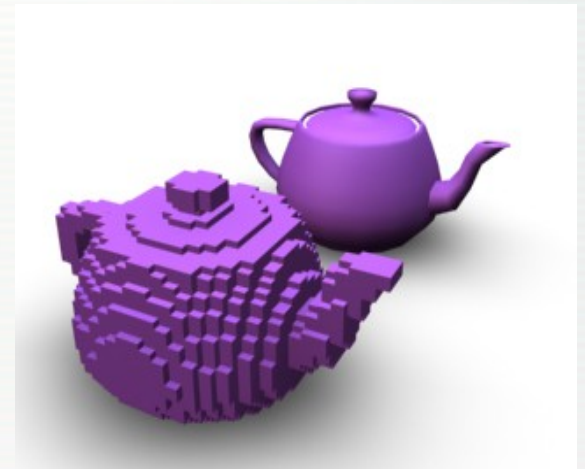
Aufgabe 2: Voxel

- **Anforderungen:**
 - Wenig Speicherverbrauch
 - Erweiterbarkeit
 - Schnelle Verarbeitung
- **Datenstrukturen:**
 - Array
 - Verkettete Liste
 - Hash-Table
 - Quad-Trees



Aufgabe 2: Voxel

- **Datenstrukturen in C++:**
 - **Array:**
 - Allokiert von Beginn an den Speicher für ein voll-ständiges Gitter
 - Nicht erweiterbar!
 - **Vector:**
 - Allokiert dynamisch Speicher
 - Erweiterbar
 - Abfrage eines Elementes schwierig!
 - **Hash-Table/Map:**
 - Allokiert dynamisch Speicher
 - Erweiterbar
 - Einfache Abfrage eines beliebigen Elementes



Aufgabe 2: Voxel

- Hash Table mit STL
- STL's Map Klasse erlaubt es Daten mithilfe eines Schlüssels abzuspeichern und daraufhin abzufragen
- Schlüssel kann ein Objekt beliebiger Klasse sein
- Beispiel String als Schlüssel:

```
std::map<string, int> notenListe;  
notenListe[ "frank" ] = 5;
```

Aufgabe 2: Voxel

- **Map-Funktionen:**

```
std::map<key_type, data_type, compare_func>
```

```
erase(key_type key_value);
```

```
int size();
```

```
bool empty();
```

```
void clear();
```

Aufgabe 2: Voxel

- **Voxel Map:**

```
std::map<int, osg::vec3> voxelMap;
```

- **Schlüssel für Voxel Map:**

- xyz als Integer
- z.B. : 111 = 1. Reihe, 1. Spalte, 1. Tiefenspalte
- Hash Funktion: $x + y * \text{dim} + z * \text{dim} * \text{dim}$
- dim = dimension

```
int compute_hash_key(int x, int y, int z)
{
    return x+y*dim+z*dim*dim;
}
```

Aufgabe 2: Voxel

- **Abfragen ob ein Voxel gesetzt wurde:**

```
if(voxelMap.find(compute_hash_key(1,1,1))==  
    voxelMap.end())  
{  
    std::cerr << "(main) Couldn't find voxel!" <<  
    std::endl;  
}
```

Aufgabe 3: CSG's

Tron



Aufgabe 3: CSG's

- **CSG's sind einfach zu implementieren, wenn elementare Datenstruktur vorhanden.**
- **Auf Voxeln sind CSG's sehr einfach zu implementieren. Ein CSG von zwei Voxeln ist wieder ein Voxel.**
- **Ein Voxel Repräsentiert ein Volumen.**
- **Bei Polygonen:**
 - **Einzelne Polygone repräsentieren kein Volumen**
 - **CSG von zwei Polygon ist oft kein Polygon**
 - **Algorithmen zur Volumen Bestimmung und zur Triangulation sind notwendig.**

Lichter in OSG

- **Bisher: Verwendung der Standardbeleuchtung von OSG**
- **Besser: Eigene Lichtquellen**
 - Punkt-Lichtquellen ●
 - Richtungs-Lichtquellen ↙
- **LightSource**
 - Extends the Group Node
 - addChild ist immer noch möglich
 - Enthält genau ein Light Objekt

Light

- **Kapselt OpenGL glLight() Funktionalitäten**
 - **setAmbient()** setzt die Ambiente Farbe
 - **setDiffuse()** setzt diffuse Farbkomponente
 - **setSpecular()** setzt spekulare Komponente

 - **setPosition()** setzt Position des Lichtes
 - **setDirection()** setzt Richtung des Spots

 - **setConstantAttenuation()**
 - **setLinearAttenuation()**
 - **setQuaraticAttenuation()**

Light

- **„Wo Licht ist ist auch Schatten“**
- **Stimmt nicht: in OSG/OpenGL gibt es per default keine Schatten**
- **Kann aber durch eigenen Code erreicht werden**
- **Nicht trivial!**

Light

- **Punktlicht:**

```
osg::Light* myLight = new osg::Light;
myLight->setLightNum(0);
myLight->setPosition(osg::Vec4(0,0,0,1));
myLight->setAmbient(osg::Vec4(0,0,1,1));
myLight->setDiffuse(osg::Vec4(0,0,1,1));

osg::LightSource* lightSource = new
    osg::LightSource;
lightSource->setLight(myLight);
lightSource-
    >setLocalStateSetModes(osg::StateAttribute::ON)
    ;
```

Light

- **Richtungslicht:**

```
osg::Light* myLight = new osg::Light;
myLight->setLightNum(0);
myLight->setPosition(osg::Vec4(0,0,0,0));
myLight->setAmbient(osg::Vec4(0,0,1,1));
myLight->setDiffuse(osg::Vec4(0,0,1,1));

osg::LightSource* lightSource = new
    osg::LightSource;
lightSource->setLight(myLight);
lightSource-
    >setLocalStateSetModes(osg::StateAttribute::ON)
    ;
```

Eselsbrücke:

„Ein Punkt ist keine Richtung!“

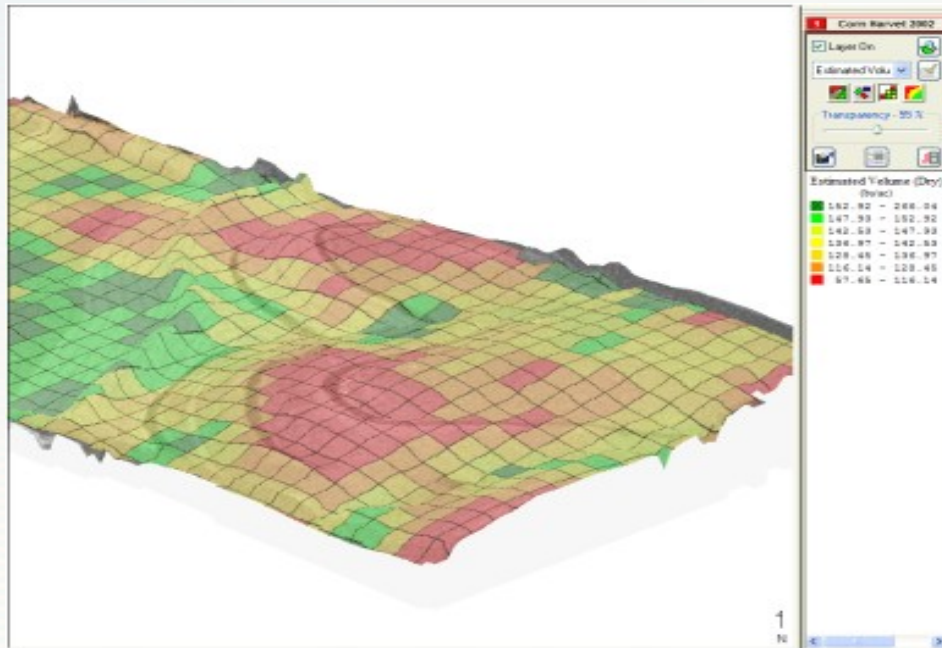
Light

- **Spotlight:**

```
osg::Light* myLight = new osg::Light;  
myLight->setLightNum(0);  
myLight->setPosition(osg::Vec4(0,0,0,1));  
myLight->setAmbient(osg::Vec4(0,0,1,1));  
myLight->setDiffuse(osg::Vec4(0,0,1,1));  
  
myLight->setSpotExponent(50.0);  
myLight->setDirection(osg::vec3(1,1,-1));
```

Heightfields

- Komplexe Geometrien sind oftmals definiert über Gitterstrukturen
- Die einzelnen Punkte des Gitters werden in die Höhe verschoben und ergeben die Geometrie



OSG Heightfields

- **OpenSceneGraph unterstützt Heightfields**
- **osg::Heightfields can als drawable benutzt werden**
- **Muss einem Geode zugeordnet werden**
- **Benutzung:**

```
osg::HeightField* grid = new osg::HeightField;  
grid->allocateGrid(20,20);  
grid->setXInterval(1);  
grid->setYInterval(1);
```

```
for(int x = 0; ...)  
    for(int y = 0; ...)  
        grid->setHeight(x,y, Höhe an Punkt <x,y>)
```