|  |  |  |
| --- | --- | --- |
| vCity | Juni 10  2014 | |
| Entwicklungsdokumentation des SWP2-Projekts zur Berechnung des Volumens und Verschattung von Stadtmodellen mit Hilfe einer Grafikkarte. | | Softwareprojekt 2 |

Inhalt

[Projekt 2](#_Toc390176844)

[Klassendiagramm 2](#_Toc390176845)

[Model 2](#_Toc390176846)

[OpenCL 3](#_Toc390176847)

[Klassenbeschreibungen 4](#_Toc390176848)

[Model 4](#_Toc390176849)

[City 4](#_Toc390176850)

[Building 4](#_Toc390176851)

[BoundarySurface 5](#_Toc390176852)

[Polygon 5](#_Toc390176853)

[Triangle 6](#_Toc390176854)

[ShadowTriangle 6](#_Toc390176855)

[Vertex 6](#_Toc390176856)

[OpenCL 7](#_Toc390176857)

[CalculatorInterface 7](#_Toc390176858)

[CalculatorImpl 7](#_Toc390176859)

[ShadowPrecision 7](#_Toc390176860)

[VolumeCalculatorInterface 8](#_Toc390176861)

[VolumeCalculatorJavaBackend 8](#_Toc390176862)

[VolumeCalculatorOpenClBackend 8](#_Toc390176863)

[ShadowCalculatorInterface 8](#_Toc390176864)

[ShadowCalculatorJavaBackend 8](#_Toc390176865)

[ShadowCalculatorOpenClBackend 8](#_Toc390176866)

[OpenClContext 9](#_Toc390176867)

[SunPositionCalculator 9](#_Toc390176868)

[Werkzeuge 9](#_Toc390176869)

# Aufgabenstellung

Die Aufgabenstellung war die Erstellung eines Programms in Java zur Volumenberechnung eines Stadtmodells auf der Grafikkarte. Dazu soll aus einer CityGML-Datei das Stadtmodell eingelesen werden. Dann sollen die Daten transformiert und in den Ursprung verschoben werden. Zur Anzeige des 3D-Stadmodells soll OpenGL verwendet werden, zur Berechnung auf der Grafikkarte haben wir uns für OpenCL entschieden.

Später kam noch die Verschattungsberechnung dazu. Die Verschattung soll mit dem Ray-Tracing-Verfahren durchgeführt werden.

# Klassendiagramm

## P:\SWP\Klassendiagramm_model.pngModel

## P:\SWP\Klassendiagramm_opencl.pngOpenCL

# Klassenbeschreibungen

## Model

City ist gedacht als übergeordnete Speicherklasse, die allgemeine Daten zur Stadt speichert, sowie die Gebäude beinhaltet. Ein Gebäude besteht aus mehreren **BoundarySurfaces** die wiederum aus Polygonen bestehen. Ein **Polygon** besteht aus mehreren Dreiecken.   
Zur besseren Genauigkeit werden diese Dreiecke bei der Schattenberechnung nochmal in kleinere Schattendreiecke unterteilt. Diese Schattendreiecke werden auch im **Polygon** gespeichert.

### City

City-Klasse bietet Zugriff auf die gesamte Stadt und alle ihre Unterelemente.

* **getInstance**() : City  
  City ist ein Singleton, hierdurch erfolgt der Zugriff auf das City-Objekt.
* **addBuilding**(b : Building) : void  
  fügt ein Gebäude der Stadt hinzu.
* **getBuildings**() : ArrayList<Building>  
  hierdurch erfolgt der Zugriff auf alle Gebäude der Stadt.
* **getCenter**() : Vertex  
  gibt das Zentrum der Stadt zurück.
* **setTotalVolume**(totalVolume : double) : void  
  setzt das Gesamtvolumen der Stadt.
* **getTotalVolume**() : double  
  gibt das Gesamtvolumen der Stadt zurück.
* **setTotalShadowTrianglesCount**(triangleCount : int) : void  
  setzt die Gesamtanzahl von Schattendreiecken der ganzen Stadt.
* **getTotalShadowTrianglesCount**() : int  
  gibt die Gesamtanzahl von Schattendreiecken zurück.

### Building

Repräsentiert ein gesamtes Gebäude.

* **setVolume**(volume : double) : void  
  setzt das Volumen des Gebäudes.
* **getVolume**() : double  
  gibt das Volumen des Gebäudes zurück.
* **addBoundarySurface**(bs : BoundarySurface) : void  
  fügt eine Gebäudeoberfläche dem Gebäude hinzu.
* **addBoundarySurface**(bs : ArrayList<BoundarySurface>) : void  
  fügt dem Gebäude mehrere Gebäudeoberflächen hinzu.
* **getBoundarySurfaces**() : ArrayList<BoundarySurface>  
  gibt alle Gebäudeoberflächen eines Gebäudes zurück.
* **getId**() : String  
  gibt die ID des Gebäudes zurück.
* **translate**(x: float, y: float, z: float) : void  
  verschiebt das Gebäude um die gegebenen Koordinaten.
* **scale**(x: float, y: float, z: float) : void  
  Verändert die Größe des Gebäudes um die gegebenen Parameter.
* **getCenter**() : Vertex  
  gibt den Mittelpunkt des Gebäudes zurück.
* **setCenter**(Vertex) : void  
  setzt den Mittelpunkt des Gebäudes.
* **getStreetName**() : String  
  gibt den Namen des Straße, in dem sich das Gebäude befindet, zurück.
* **setStreetName**(name : String) : void  
  setzt den Namen der Straße zu der das Gebäude gehört.

### BoundarySurface

Beschreibt die verschiedenen Oberflächen eines Gebäudes.

* **BoundarySurface**(id : String)  
  erzeugt eine neue Oberfläche mit gegebener ID.
* **getId**() : String  
  gibt die ID einer Oberflöche zurück.
* **setType**(type : SurfaceType) : void  
  setzt den Type einer Oberfläche (WALL, ROOF, GROUND, OTHER)
* **getType**() : SurfaceType  
  gibt den Typ der Oberfläche zurück.
* **getPolygons**() : ArrayList<Polygons>  
  gibt alle Polygone zurück, die die Oberfläche darstellen.
* **addPolygon**(p : Polygon) : void  
  fügt der Oberfläche ein Polygon hinzu.
* **addPolygons**(polygons : ArrayList<Polygon>) : void  
  fügt der Oberfläche mehrere Polygone hinzu.

### Polygon

* **setPercentageShadow**(percentageShadow : double[]) : void  
  setzt den prozentualen Anteil des Dreiecks der sich im Schatten befindet.
* **getPercentageShadow**() : double[ ]  
  gibt den prozentualen Anteil zurück zu dem sich das Dreieck im Schatten befindet.
* **Polygon**(String)  
  Konstruktor mit ID als Übergabeparameter.
* **getId**() : String  
  gibt die ID des Polygons zurück.
* **getArea**() : double  
  gibt den Flächeninhalt des Polygons zurück.
* **setArea**(a : double) : void  
  setzt den Flächeninhalt des Polygons.
* **getTriangles**() : ArrayList<Triangle>  
  gibt alle zum Polygon zugehörigen Dreiecke zurück.
* **addTriangle**(t : Triangle) : void  
  fügt dem Polygon ein Dreieck hinzu.
* **addTriangles**(ts : ArrayList<Triangle>) : void  
  fügt dem Polygon mehrere Dreiecke hinzu.
* **addShadowTriangle**(st : ShadowTriangle) : void  
  fügt dem Polygon ein Schattendreieck hinzu.
* **getShadowTriangles**() : ArrayList<ShadowTriangle>  
  gibt alle zum Polygon gehörenden Schattendreiecke zurück.

### Triangle

* **getNormalVector**() : Vertex  
  gibt den Normalenvektor des Dreiecks zurück.
* **setNormalVector**(normalVektor : Vertex) : void  
  setzt den Normalenvektor des Dreiecks.
* **Triangle**(vertices : Vertex[])   
  Konstruktor mit drei Punkten als Array übergeben zum Konstruieren des Dreiecks.
* **Triangle**(v0 : Vertex, v1 : Vertex, v2 : Vertex)   
  Konstruktor mit drei Punkten die direkt übergeben werden.
* **getVertices**() : Vertex[ ]  
  gibt alle Punkte des Dreiecks zurück.

### ShadowTriangle

* **ShadowTriangle**(v : Vertex[], b : Building)  
  Konstruktor mit drei Punkten als Array die das Schattendreieck definieren und zusätzlich wird das Gebäude übergeben zu dem das Schattendreieck gehört.
* **ShadowTriangle**(v0 : Vertex, v1 : Vertex, v2 : Vertex, b : Building)  
  Konstruktor mit drei Punkten die direkt als Punkte übergeben werden.
* **getShadowSet**() : BitSet  
  gibt das zu diesem Schattendreieck gehörige SchattenBitSet zurück.
* **getBuilding**() : Building  
  gibt das Gebäude zurück zu dem das Schattendreieck gehört.
* **setShadowSet**(s : BitSet) : void  
  setzt das SchattenBitSet für dieses Schattendreieck.
* **getCenter**() : Vertex  
  gibt den Mittelpunkt des Schattendreiecks zurück.

### Vertex

* **Vertex**(coordinates : float[])  
  Konstruktor in dem die Koordinaten für den Ortsvektor als Array übergeben werden.
* **Vertex**(x : float, y : float, z : float)  
  Konstruktor in dem die Koordinaten für den Ortsvektor direkt als float übergeben werden.
* **getCoordinates**() : float[]  
  gibt die Koordinaten des Ortsvektors zurück.
* **getX**() : float  
  gibt die x – Koordinate des Ortsvektors zurück.
* **getY**() : float  
  gibt die y – Koordinate des Ortsvektors zurück.
* **getZ**() : float  
  gibt die z – Koordinate des Ortsvektors zurück.
* **translate**(x : float, y : float, z : float) : void  
  verschiebt den Ortsvektor um die gegebenen Parameter.
* **scale**(x: float, y : float, z : float) : void  
  skaliert den Ortsvektor um die gegebenen Parameter.

## OpenCL

Das **CalculatorInterface** und die Implementierung **CalculatorImpl** dienen als Schnittstelle um die Volumen- sowie die Schattenberechnung aufzurufen.  
In der Aufzählung **Shadowprecision** wird eine Unterteilung der Präzision der Schattenberechnung durch Festlegung der maximal erlaubten Größe der Dreiecksflächen vorgenommen.  
Falls die Berechnung mit Hilfe von OpenCL fehlschlagen sollte, so wird die Berechnung mit Java durchgeführt.   
Die Klasse **SunPositionCalculator** dient dazu die Position der Sonne einem Himmelsabschnitt zuzuordnen.

### CalculatorInterface

* **calculateVolume**() : void  
  berechnet das Volumen aller Gebäude der Stadt.
* **calculateShadow**(precision : ShadowPrecision, splitAzimuth : int, splitHeight : int) : void  
  berechnet die Verschattung der Stadt mit der gegebenen Präzision und der gegebenen Aufteilung des Himmelsmodells.
* **calculateArea**() : void  
  berechnet die Fläche aller Polygone der Stadt.

### CalculatorImpl

Implementiert das **CalculatorInterface**

### ShadowPrecision

* **VERY\_LOW**  
  Schattendreiecke mit maximaler Fläche von 5m²
* **LOW**  
  Schattendreiecke mit maximaler Fläche von 2,5m²
* **MID**  
  Schattendreiecke mit maximaler Fläche von 1,25m²
* **HIGH**  
  Schattendreiecke mit maximaler Fläche von 0,75m²
* **ULTRA**  
  Schattendreiecke mit maximaler Fläche von 0,375m²
* **HYPER**  
  Schattendreiecke mit maximaler Fläche von 0,10m²
* **AWESOME**  
  Schattendreiecke mit maximaler Fläche von 0,01m²
* **getArea**() : float  
  gibt die maximal erlaubte Fläche der Präzisionsstufe zurück.
* **getName**() : String  
  gibt den Namen der Präzisionsstufe zurück.

### VolumeCalculatorInterface

* **calculateVolume**() : void  
  berechnet das Volumen der Gebäude die zur Stadt gehören.

### VolumeCalculatorJavaBackend

Implementiert das Interface **VolumeCalculatorInterface**.

### VolumeCalculatorOpenClBackend

Implementiert das Interface **VolumeCalculatorInterface**.

### ShadowCalculatorInterface

* **calculateShadow**(precision : ShadowPrecision, splitAzimuth : int, splitHeight : int) : void  
  berechnet die Verschattung der Stadt mit der gegebenen Präzision und der gegebenen Aufteilung des Himmelsmodells.
* **vertexDiff**(v1 : Vertex, v0 : Vertex) : Vertex

### ShadowCalculatorJavaBackend

Erweitert die abstrakte Klasse **ShadowCalculatorInterface**.

### ShadowCalculatorOpenClBackend

Erweitert die abstrakte Klasse **ShadowCalculatorInterface**.

### OpenClContext

* **getInstance**() : OpenClContext  
  Gibt das Objekt von OpenClContext zurück.
* **reinit**() : void  
  erstellt einen neuen OpenCL-Kontext.
* **createKernelFromFile**(filename : String) : cl\_kernel  
  erstellt einen OpenCL – Kernel aus der angebenen Datei.
* **finalizeKernel**(cl\_kernel : kernel) : void  
  gibt die Ressourcen des Kernels wieder frei.
* **getDevice**() : cl\_device\_id  
  Gibt die ID der zu verwendenden GPU zurück.
* **getClContext**() : cl\_context  
  gibt den OpenCL - Kontext zurück.
* **getClCommandQueue**() : cl\_command\_queue  
  gibt die OpenCL – Commandqueue zurück.
* **profile**(kernelEvent : cl\_event) : void  
  schreibt Timing-Daten auf die Console.

### SunPositionCalculator

* **SunPositionCalculator**(d : Date, parser : ParserInterface)  
  Konstruktor mit übergebenem Datum und ParserInterface. Berechnet die Azimuth- und Höhenwinkel der Sonne zum gegebenen Datum und Ort.
* **getAzimutAngle**() : double  
  gibt den Azimuthwinkel zurück.
* **getAltitude**() : double  
  gibt den Höhenwinkel zurück.
* **getX**() : double  
  gibt die x – Koordinate des Vektors in Richtung der Sonne zurück.
* **getY**() : double  
  gibt die y – Koordinate des Vektors in Richtung der Sonne zurück.
* **getZ**() : double  
  gibt die z – Koordinate des Vektors in Richtung der Sonne zurück.
* **getSunPosition**(splitAzimuth : int, splitHeight : int) : int  
  gibt den Index des Himmelsabschnitts in dem sich die Sonne befindet zurück.

# Werkzeuge