# Import Funktionen

## General CityGML Import

In der Parser.java Klasse werden die Daten nach den gegebenen Anforderungen geparst und modifiziert. Dazu zählen das Einlesen einer GML-Datei, die Koordinatentransformation und das Aufbauen und füllen des Datenmodells.

### Vorgehen:

Durch das Singleton-Pattern wird sichergestellt, dass nur eine Instanz des Parsers erzeugt wird. Nachdem durch das Parserinterface eine Instanz des Parsers erzeugt wurde, ist es nur notwendig die Methode parse(String InputFileName) aufzurufen. Die restlichen Schritte werden intern durch den Parser übernommen.

Funktionen des Parsers:

* CityGML-Datei einlesen
* Koordinaten transformieren
* Triangulieren
* Normalen Vektor eines Dreiecks ausrechnen

### CityGML einlesen

Analog zu der GML Datei muss der Parser durch die verschiedenen Tags durchgehen. Hierbei geht der Parser jeweils durch die cityObjectMember denn diese enthalten die Gebäude. Danach wird aus der geparsten Gebäude ID ein Gebäude aus dem internen Datenmodell erzeugt. Als Nächstes werden dem Gebäude der Stadtname, in dem es sich befindet und der Straßenname hinzugefügt. Um die Koordinaten der Polygone herauszufinden, wird hierbei die Methode surfaceRunner aufgerufen.

Das Durchlaufen der GML Datei geschieht hierbei durch die beiden Methoden:

referenceRunner**(**BoundarySurfaceProperty property**) und**

surfaceRunner**(**BoundarySurfaceProperty property**,** Building build**,** ArrayList**<**Triangle**>** polyTriangles**)**

Der referenceRunner durchsucht die GML nach dem Referenzwert, dieser Wert wird benötigt um die Koordinaten zum Ursprung zu transformieren. Der Referenzwert besteht aus dem Vertex mit der kleinsten x-Koordinate und dem dazugehörigen y-und z-Koordinaten.

Der SurfaceRunner durchsucht die Oberflächen nach Koordinaten. Dazu geht der Parser über alle Oberflächen und speichert ihren Typ. Danach gelangt der Parser über den LinearRing zu den gewünschten Gebäudekoordinaten. Diese Koordinaten werden in einer Liste gespeichert. Um die Koordinaten anschaulich zu machen, werden diese noch zum Koordinatenursprung transformiert und trianguliert. Nach der Triangulation wird hierbei gleich der Normalvektor der Dreiecksfläche ausgerechnet und dem Modell hinzugefügt.

Als letzten Schritt werden diese Dreiecke dem Datenmodell Polygon, die Polygone der BoundarySurface und der BoundarySurface dem Gebäude hinzugefügt. Zu guter Letzt werden die Gebäude der City hinzugefügt und zurückgegeben.

## Polygon Triangulation

Der für die Renderer benötigte Bibliothek kann nur konvexe Polygone zeichnen. Ein Polygon ist konvex, wenn der Innenwinkel an jedem Eckpunkt höchstens 180° beträgt. Falls auch nur ein Innenwinkel größer als 180° ist, ist das Polygon konkav. Um konkave Polygone zeichnen zu können, müssen diese Polygone erst in Dreiecke unterteilt werden denn Dreiecke sind immer konvex. Diese Unterteilung der Polygone nennt man Triangulation (engl.Tesselation).

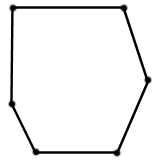
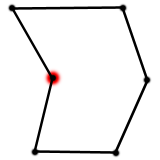
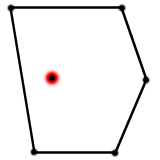
  

Abbildung 1 konvexes Polygon  Abbildung 2 konkaves Polygon Abbildung 3 Fehler beim konkaven Polygon

Um diese Triangulation durchzuführen bietet das Framework JOGL verschieden Routinen an. Diese Routinen erhalten als Input das Polygon und als Ergebnis werden die Dreiecke abhängig von Typ zurückgegeben.

### Vorgehen

In der Klasse PolygonTriangulator wird zuerst ein Tesselator Objekt erzeugt dies geschieht indem GLU.*gluNewTess*() aufgerufen wird. Danach werden durch verschiedene Callback Routinen Funktionen für das Objekt festgelegt. Diese Funktionen werden durch *gluTessCallback* definiert*.*

*gluTessCallback* ist folgendermaßen aufgebaut*:*

GLU.*gluTessCallback*(tessobj, type, funktion);

Hier werden dem Objekt tessobj die Funktion funktion hinugefügt. Der Typ des Callbacks wird durch type angeben.

In unserem Programm werden die Typen *GLU\_TESS\_VERTEX*, *GLU\_TESS\_BEGIN*, *GLU\_TESS\_END*,

*GLU\_TESS\_ERROR* und *GLU\_TESS\_COMBINE* verwendet.

*GLU\_TESS\_VERTEX* speichert die Eckpunkte eines Polygons

*GLU\_TESS\_BEGIN* startet ein neues Polygon

*GLU\_TESS\_END* beendet das aktuelle Polygo*n*

*GLU\_TESS\_ERROR* gibt eine Fehlermeldung mit dem Fehler aus

*GLU\_TESS\_COMBINE* erzeugt ein neues Vertex falls sich 2 Seiten schneiden.

Diese Callback Routinen werden im Laufe des Triangulationsvorgangs von der GLU-Bibliothek aufgerufen. Die Routinen werden in der Klasse PolygonTriangulatorCallback definiert und bekommt seine Daten wiederum durch die Klasse PolygonTriangulatorCollector.

In der Klasse PolygonTriangulatorCollector werden die Vektoren gesammelt und anhand ihres Typs ausgewertet. Diese Typüberprüfung findet am Ende eines Polygons statt.

Diese Typen sind:

* GL\_TRIANGLE
* GL\_Triangle\_STRIP
* GL\_TRIANGLE\_FAN

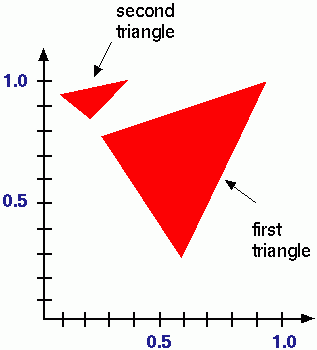


Abbildung 4 GL\_TRIANGE

Jeweils 3 Vektoren bilden hierbei ein unabhängiges Dreieck.

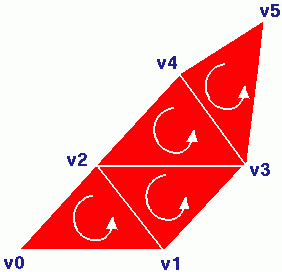


Abbildung 5 GL TRIANGLE\_STRIP

Die Dreiecke sind durch eine Seite (2 Eckpunkte) miteinander verbunden. Hier bei ist zu beachten, dass alle Dreiecke die gleiche Drehrichtung haben.

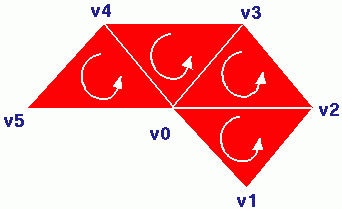


Abbildung 6 GL\_TRIANGLE\_FAN

Der Triangle Fan ist ähnlich zu dem Triangle Strip außer, dass der Anfangspunkt bei alles Dreiecken gleich ist.

## Vertex Translation

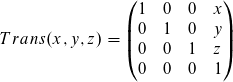
Damit die Daten beim Renderer passend ankommen müssen sie etwas verändert werden, dazu müssen die Koordinaten einmal zum Koordinatenursprung verschoben und gedreht werden. Diese beiden Veränderungen geschehen mit der Methode:

translateToOrigin**(** ArrayList**<**VertexDouble**>** poly**,** double**[]** reference**)**

Diese Methode bekommt als Parameter eine Liste vom Typ VertexDouble und ein Array mit den Referenzwerten. Die Liste enthält die geparste Posliste aus der GML Datei. Das Referenzarray wird vom Parser ermittelt.

### Verschiebung(Translation)

Die x,y und z Koordinaten einer Vertex werden durch eine Matrix zum Koordinatenursprung verschoben. Die Matrix lautet:



Dabei entsprechen die Werte x,y und z den aus dem Parser ermitteltem Referenzwert.

### Drehung (Rotation)

Die Koordinatenachsen müssen an der X-Achse gegen den Uhrzeigersinn um 90° gedreht werden, weil die Achsen aus der GML Datei nicht mit denen des Renderers übereinstimmen.

x’ ´

z ´

y ´

x

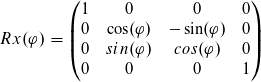
y

z

Koordinatenachsen aus der Koordinatenachsen des Renderers

GML Datei

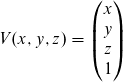
Die Matrix für die Rotation lautet:



φ entspricht hierbei -90°

### Rechnung

Die zu verändernden Koordinaten werden aus der geparsten Liste entnommen und in diese Form eingefügt:



Im nächsten Schritt werden die Verschiebe- und Rotationsmatrizen miteinander multipliziert. Schließlich werden Alle Vektoren V mit der zuvor berechneten Transformationsmatrix multipliziert. Als Ergebnis kommen die Transformierten Vektroren heraus.

# Quellen

Open GL Programming,

<http://profs.sci.univr.it/~colombar/html_openGL_tutorial/en/02rendering_022.html>, 15.06.2014

DGL Wiki, <http://wiki.delphigl.com/index.php/Konvex>, 15.06.2014

Kiet Le, Polygon Tessellation in JOGL,

<http://www.java-tips.org/other-api-tips/jogl/polygon-tessellation-in-jogl.html>, 15.06.2014

GLU Polygon Tessellation, <http://www.felixgers.de/teaching/jogl/gluTessellation.html>, 15.06.2014

Estevao, Martin, Polygon Tessellation In OpenGL,

<http://www.flipcode.com/archives/Polygon_Tessellation_In_OpenGL.shtml>, 15.06.2014

## Bildquellen:

Abbildung 1: <http://wiki.delphigl.com/images/2/28/Konvex_normal.png>,15.06.2014

Abbildung 2: <http://wiki.delphigl.com/images/4/4c/Konvex_konvex.png>,15.06.2014

Abbildung 3: <http://wiki.delphigl.com/images/2/29/Konvex_konvex2.png>,15.06.2014

Abbildung 4: <http://profs.sci.univr.it/~colombar/html_openGL_tutorial/images/triangles.gif>,15.06.2014

Abbildung 5:

<http://profs.sci.univr.it/~colombar/html_openGL_tutorial/images/triangle_strips.gif>, 15.06.2014

Abbildung 6: <http://profs.sci.univr.it/~colombar/html_openGL_tutorial/images/triangle_fans.gif>,15.06.2014