

# **Computergrafik Framework - Architekturdokumentation, Komponenten und Konzepte**

Philipp Jenke

Hochschule für Angewandte Wissenschaften (HAW), Hamburg

11. September 2015

## Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>4</b>
<b>2</b>	<b>Getting Started</b>	<b>4</b>
<b>3</b>	<b>Komponenten</b>	<b>5</b>
3.1	Module . . . . .	5
3.2	Anwendungsobjekt . . . . .	6
3.3	Rendering . . . . .	6
3.4	Grafische Benutzerschnittstelle . . . . .	6
3.5	Menüs . . . . .	7
3.6	Ressourcen . . . . .	8
3.7	Logging . . . . .	8
<b>4</b>	<b>Konzepte</b>	<b>8</b>
4.1	Vektoren und Matrizen . . . . .	8
4.2	Szenengraph . . . . .	9
4.3	Rendering . . . . .	9
4.3.1	JOGL . . . . .	9
4.3.2	jMonkey . . . . .	10
4.4	Lichtquellen . . . . .	10
4.5	Texturen und Shader . . . . .	11
4.5.1	Texturen . . . . .	11
4.5.2	Shader . . . . .	11
4.6	Picking . . . . .	11
4.6.1	Erstellen von Picking-Items . . . . .	12
4.6.2	Interaktion mit Picking-Items . . . . .	12
<b>5</b>	<b>Verwendung</b>	<b>12</b>
5.1	Dreiecksnetze . . . . .	12
5.2	Punktwolken . . . . .	13
5.3	Kurven . . . . .	13
5.4	Animierte Daten . . . . .	13
5.5	Kamera . . . . .	14
5.6	Film-Export . . . . .	14
5.7	Octree . . . . .	15
5.8	Daten-Im- und Export . . . . .	15
5.8.1	Dreiecksnetze . . . . .	15
5.8.2	Motion-Capture-Daten . . . . .	15
5.8.3	Punktwolken . . . . .	15
<b>6</b>	<b>Projekte</b>	<b>16</b>
6.1	3D Scanner . . . . .	16
6.1.1	Abhängigkeiten . . . . .	16
6.1.2	Installation . . . . .	16
6.1.3	Einrichten des USB-Seriell-Adapters . . . . .	16
6.1.4	Testen des Distanzsensors . . . . .	17
6.2	Urban Reconstruction . . . . .	17
6.2.1	Abhängigkeiten . . . . .	17
6.2.2	Installation . . . . .	17
<b>7</b>	<b>Best Practises</b>	<b>17</b>
7.1	Git . . . . .	17
7.1.1	Anlegen eines neuen lokalen Repositories . . . . .	17

## Computergrafik Framework - Architekturdokumentation, Komponenten und Konzepte

7.1.2	Anlegen eines neuen Repositories auf dem Server . . . . .	18
7.1.3	Setzen eines Remote Repositories . . . . .	18
7.1.4	Lokale Änderungen an Server Repository senden . . . . .	18
7.1.5	Lokales Klon-Repository von Server holen . . . . .	18
7.2	Sonar . . . . .	18

## 1 Einleitung

Ziel dieses Dokumentes ist es, die grundlegenden Konzepte und die architektonischen Entscheidungen für die Softwarekomponenten in der Software-Bibliothek darzulegen. Außerdem werden Installationsanleitungen zum Auschecken und für die Inbetriebnahme der Projekte gegeben.

## 2 Getting Started

Wir beginnen mit dem Schnelleinstieg. Zunächst müssen auf dem Rechner die Voraussetzungen sichergestellt werden:

- Oracle Java 1.8 oder neuer  
(<http://www.oracle.com/technetwork/java/javase/overview/index.html>)
- Gradle 2.6 oder neuer (<https://gradle.org/>)
- Git 2.3 oder neuer (unterschiedliche Quellen je nach OS)
- Eclipse 4.5 (Mars) oder neuer (<https://eclipse.org/>)

Folgende Schritte müssen durchgeführt werden, damit das Framework verwendet werden kann:

- **Auschecken des Repositories:** Das Repository mit allen notwendigen Sourcen und sonstigen Dateien finden sich unter folgender URL:  
*git.informatik.haw-hamburg.de/srv/git/computergrafik/computergraphics.*  
Es muss auf den Entwicklungsrechner gecloned werden:  

```
git clone ssh://<login>@git.informatik.haw-hamburg.de/  
    srv/git/computergrafik/computergraphics
```
- **Kompilieren mit Gradle:** Gradle bietet die Möglichkeit, das Projekt über die Kommandozeile zu kompilieren. Dazu muss im Projektverzeichnis (Elternverzeichnis der Unterprojekte wie *graphics\_core*) folgender Befehl ausgeführt werden:  

```
gradle build
```
- **Erstellen der Eclipse-Projekte:** Zum Erstellen der Eclipse-Projektdateien wird wieder Gradle verwendet:  

```
gradle eclipse
```

  
Nach erfolgreicher Ausführung dieses Befehls befinden sich Eclipse-Projektdateien (*.project*, ...) in jedem der Unterprojekte.
- **Import in Eclipse:** Je nachdem, was genau gemacht werden soll, benötigt man verschiedene Unterprojekte. Beispielhaft soll hier die Demo-Anwendung **ObjTriangleMesh** ausgeführt werden. Diese befinden sich in dem Unterprojekt *apps*. Um das Projekt in Eclipse zu importieren sind die folgenden Schritte notwendig:
  - File
  - Import
  - Existing Projects into Workspace
  - Select root directory: *<Hier muss das Unterverzeichnis apps ausgewählt werden>*
  - Finish

Nun ist das Projekt in Eclipse importiert und erscheint im Projekt-Explorer.

- **Asset-Pfad einrichten:** Viele Programme verwenden Daten (z.B. Icons oder Dreiecksnetze). Standarddaten finden sich im Projektverzeichnis im Unterverzeichnis *assets*. Damit ausgeführte Programme diese Assets auch finden, muss der Pfad zu diesem Verzeichnis bekannt gegeben werden. Dazu wird der absolute Pfad in der Textdatei *resources.txt* eingetragen.

- **Demo-Programm starten:** Im Package `cgresearch.apps.trianglemeshes` kann nun die Klasse `ObjTriangleMesh` durch *Rechtsklick* → *Run As* → *Java Application* ausgeführt werden. Hat alles geklappt, dann öffnet sich ein Fenster und ein 3D-Modell erscheint.

## 3 Komponenten

### 3.1 Module

Das Framework besteht aus folgenden Modulen

- **graphics\_core:** Szenengraph (siehe Abschnitt 4.2), Interfaces für die Bausteine (Anwendung, UI und Rendering), Picking-Funktionalität, Kamera, Datenstrukturen und Algorithmen, I/O, Material
- **rendering\_jogl:** Rendering mit OpenGL (hier JOGL)
- **apps:** Programme, die die Grafik-Funktionalität verwenden
- **electronics:** Schnittstellen zu verschiedenen Elektronik-Komponenten wie Sensoren und Aktoren
- **smart\_home\_apps:** Programme aus Smart Home-Bereich
- **rc\_vehicles:** Zwei RaspberryPi-basierte Steuerungsprogramm (Boot, Bot)
- **smart\_home\_visualization:** Visualisierungslösungen für das Smart Home (Kombination aus Smart Home und Grafik)

Die Abhängigkeiten zwischen den Teilprojekten sind in Abbildung 1 aufgezeigt.

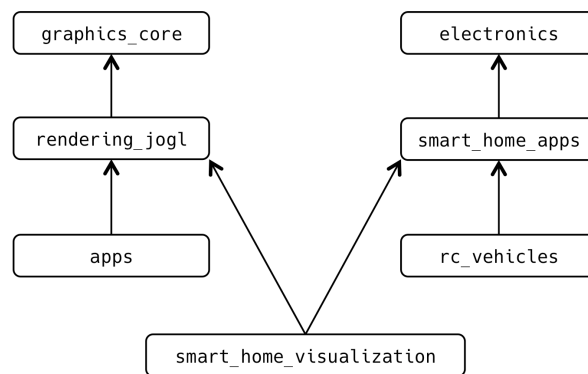


Abbildung 1: Abhängigkeiten zwischen den Teilprojekten

Im Projektverzeichnis befinden sich außerdem die folgenden Unterverzeichnisse, die keinen Code beinhalten:

- **assets:** Daten für die Anwendungen wie Icons und Dreiecksnetze
- **doc:** Dokumentation für das Framework, u.a. dieses Dokument
- **doc:** Skripte zum Verwenden des Frameworks auf einem Raspberry Pi

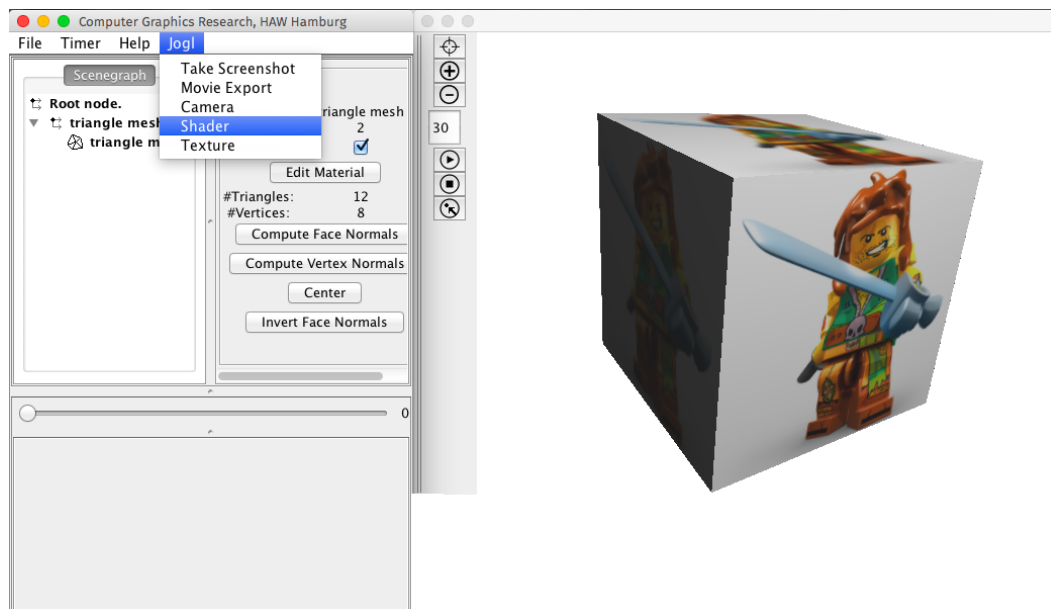


Abbildung 2: Screenshot einer Anwendung mit dem cgresearch Framework. Das Fenster besteht aus verschiedenen Blöcken: 3D-View (rechts), Debug-Konsole (unten), Szenengraph und Controller (oben links) Editor-Dialogs (Mitte) und Menü.

## 3.2 Anwendungsobjekt

In diesem Abschnitt wird beschrieben, wie mit dem cgresearch Framework eine Anwendung geschrieben werden kann. Als Rendering-System wird JOGL (siehe Abschnitte 4.3.1) verwendet. Die Anwendung liegt im Teilprojekt *apps*.

Im Zentrum einer Anwendung steht ein **CgApplication**-Objekt. Darin spielt sich die eigentliche Logik der Anwendung ab. Zur Darstellung wird (optional) eine Instanz eines Rendering-Frames erzeugt. Auch das Benutzerinterface ist eine optionale Komponente. Die Klasse **JoglSwingUserInterface** stellt beispielsweise ein Benutzerinterface für das JOGL-System mit Java Swing bereit.

Die main-Methode der Anwendung könnte also lauten:

```
new ConsoleLogger(Logger.VerboseMode.DEBUG);
ResourcesLocator.getInstance().parseIniFile("resources.ini");
CgApplication app = new TriangleMeshFrame();
new JoglFrame(app);
new JoglSwingUserInterface(app);
```

Zu dem Logger finden Sie Informationen im Abschnitt 3.7, der **ResourcesLocator** ist in Abschnitt 3.6 dargestellt.

## 3.3 Rendering

Aktuell steht primär ein System für das Rendering zur Verfügung:

- JoglFrame (JOGL)

Die Rendering-Systeme werden ausführlicher in Abschnitt 4.3 besprochen. Neben dem JOGL-System wird rudimentär auch jMonkey als Render-Engine unterstützt.

## 3.4 Grafische Benutzerschnittstelle

Aktuell stehen zwei Klassen (und damit Systeme) für das Benutzerinterface (UI) zur Verfügung:

- `SwingUserInterface` (Java Swing)
- `JoglSwingUserInterface` (Java Swing mit Zusatzfunktionalität für JOGL)

Die meisten Anwendungen werden durch ein grafisches Benutzerinterface (GUI) gesteuert. In einer Anwendung können Sie eigene Benutzerinterfaces umsetzen und registrieren. Diese tauchen dann als zusätzlicher Tab im Fensterbereich *Szenengraph und Controller* auf. Zum Schreiben eines solchen Controllers implementieren Sie eine eigene Klasse, die von der abstrakten Klasse `IApplicationControllerGui` erbt. Diese Klasse wiederum erbt von `JPanel`. Sie müssen also das `this`-Objekt mit Ihrem GUI befüllen. Anschliessend registrieren Sie das Objekt bei Ihrer Instanz für die Grafische Benutzeroberfläche über die Methode `registerApplicationGUI()`. Hier eine einfache Beispiel-Implementierung für ein GUI, das aus einem Button besteht, der einen Tetraeder erzeugt:

```
public class PJController extends IApplicationControllerGui
    implements ActionListener {
    public PJController() {
        JButton button = new JButton("PJ!");
        button.addActionListener(this);
        add(button);
    }

    @Override
    public String getName() {
        return "PJ";
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        CgNode node = new CgNode(TriangleMeshFactory.createTetrahedron(), "Tetra");
        getRootNode().addChild(node);
    }
}
```

### 3.5 Menüs

Ihre GUI-Anwendung kann auch um benutzerdefinierte Menüs erweitert werden. Das Vorgehen ist sehr ähnlich zum Vorgehen beim Erstellen einer eigenen GUI-Komponente aus dem vorherigen Abschnitt. Zunächst muss eine Menüklass implementiert werden, die von der abstrakten Klasse `CgApplicationMenu` abgeleitet ist. Diese Instanz registrieren Sie in Ihrer GUI-Instance:

```
registerApplicationMenu(new <Menüklass>());
```

Die Elternklasse der Menüklass selber erbt von `JMenu`. Daher müssen Sie einfach das `this`-Objekt mit den gewünschten Menüeinträgen befüllen, um ein Menü zu erstellen. Eine Beispielumsetzung einer Menü-Klasse ist:

```
public class PjMenu extends CgApplicationMenu {
    public MyMenu() {
        super("MyMenu");
        JMenuItem itemPrintMessage = new JMenuItem("Print message");
        itemPrintMessage.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.out.println("Message");
            }
        });
    }
};
```

```
        add(itemPrintMessage);
    }
}
```

### 3.6 Ressourcen

An mehreren Stellen im cg-Framework wird auf Ressourcen aus dem Dateisystem zugegriffen (z.B. Icons, Bilder, Netze, Texturen, ...). Diese können sich in unterschiedlichen Pfaden befinden. Die Ressourcenpfade werden zentral über eine Singleton-Klasse **ResourcesLocator** verwaltet. In jedem Programm, das das Framework verwendet, muss daher der **ResourcesLocator** mit den notwendigen Ressourcenpfaden gefüllt werden. Dazu stehen zwei Methoden zur Verfügung:

- **addPath(String)** Hinzufügen eines Pfades
- **parseIniFile(String)** Lesen eine Liste von Pfaden (zeilenweise) aus einer INI-Datei

Alle Ressourcen, die intern geladen werden (z.B. Icons, Netze, Texturen), verwenden den **ResourcesLocator**, um alle Ressourcenpfade zu durchsuchen. Greift man in einem Anwendungsprogramm auch direkt auf Ressourcen zu, dann kann man die Methode **getPathToResource(String)** verwenden. Diese liefert entweder einen gültigen Pfad für eine Resource oder null, falls die Resource in keinem der Ressourcenpfade gefunden wurde.

Die Ressourcen, die direkt mit dem Projekt cgresearch bereitgestellt werden, befinden sich im Projektverzeichnis im Unterverzeichnis *assets*.

### 3.7 Logging

Innerhalb des Frameworks wird ein konsistentes Logging-System verwendet. Der Logger ist als Singleton implementiert und kann folgendermassen erreicht werden:

```
Logger logger = Logger.getInstance();
```

Dem Logger können folgende Nachrichten geschickt werden:

- **Nachricht:** Generelle Nachricht, wird normalerweise sofort ausgegeben
- **Debug-Nachricht:** Debugging-Nachricht, wird normalerweise nur ausgegeben, wenn Debugging aktiviert ist
- **Exception:** Auftritt einer Exception, wird normalerweise sofort ausgegeben.
- **Error:** Auftritt eines Fehlers, wird normalerweise sofort ausgegeben.

Aktuell sind folgende Logger umgesetzt:

- **ConsoleLogger** Ausgabe auf der Konsole
- **LoggerPane** Wird bei Verwendung der Swing-GUI angezeigt

## 4 Konzepte

### 4.1 Vektoren und Matrizen

```
cgresearch.core.math
```

Um die Architektur möglichst flexibel zu halten, werden auf Vektoren und Matrizen nur über Interfaces zugegriffen (**IVector3**, **IVector4**, **IMatrix3**, **IMatrix4**). Im Package gibt es außerdem eine Fabrik-Klasse (**VectorMatrixFactory**) mit der Instanzen von Vektoren und Matrizen erzeugt werden können. Dazu gibt es Referenzimplementierungen für die Interfaces. Gegebenenfalls werden diese später noch durch performantere Implementierungen ersetzt.



## 4.2 Szenengraph

`cgresearch.graphics.scenegraph`

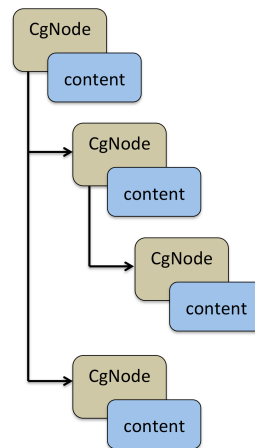


Abbildung 3: Der Szenengraph besteht aus Knoten vom Type `CgNode`. Jeder Knoten referenziert einen Inhalt (`content`). Dabei kann es sich z.B. um ein Dreiecksnetz (`ITriangleMesh`) handeln.

Alle Daten werden in einem Szenengraph organisiert. Der Szenengraph besteht aus Knoten vom Type `CgNode`. Ein Knoten kann unterschiedliche Daten repräsentieren. Auf die Daten wird mit der Methode

```
public ICgNodeContent getContent();
```

zugegriffen.

## 4.3 Rendering

`cgresearch.rendering`

Es werden unterschiedliche Rendering-Systeme unterstützt. Aktuell gibt es Implementierungen von JOGL (Abschnitt 4.3.1) und jMonkey (Abschnitt 4.3.2). Nicht alle Implementierungen unterstützen die vollständige Funktionalität. JOGL wird aktuell primär weiterentwickelt.

Jedes Rendering-System stellt einen zentralen Frame zu Verfügung. Um ein Rendering-System zu verwenden, wird eine Instanz des Frames erzeugt. Für die Visualisierung von einzelnen Knoten im Szenengraphen gibt es ein Plugin-Konzept. Für die unterschiedlichen Daten in den Szenengraph-Knoten können Fabriken registriert werden, die jeweils die zugehörigen Renderknoten erzeugen. Es entsteht also ein dualer Szenengraph für das Rendering-System. Fabriken implementieren das Interface `IRenderObjectsFactory<T>`. Sie können dem Rendering-System dann über die Methode

```
public void registerRenderObjectsFactory(IRenderObjectsFactory<T> factory);
```

des zentralen Frames mitgeteilt werden. Standard-Fabriken werden direkt über die Rendering-Systeme registriert. Eigene Fabriken können auch in den Anwendungsprojekten registriert werden.

### 4.3.1 JOGL

`cgresearch.rendering.jogl`

Das Rendering-System JOGL bietet die maximale Freiheit - es wird direkt auf die OpenGL-Ebene zugegriffen.

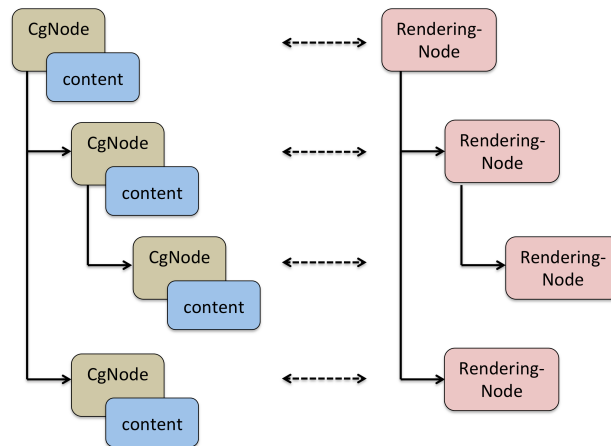


Abbildung 4: Zum Szenengraph wird dynamisch ein dualer Renderinggraph aufgebaut. Zu jedem Knoten des Szenengraphen wird ein Renderingknoten im Rendering-System erstellt. Die Renderingknoten übernehmen die Darstellung der Inhalte.

#### 4.3.2 jMonkey

`cgresearch.rendering.jmonkey`

### 4.4 Lichtquellen

Lichtquellen in der Szene werden zentral über den Wurzelknoten verwaltet. Der Wurzelknoten hat dazu einen speziellen Typ, nämlich `CgRootNode` und kann aus der zentralen Anwendung, abgeleitet von `CgApplication`, heraus erreicht werden:

```
CgRootNode getCgRootNode()
```

Der Wurzelknoten bietet Zugriff auf die Anzahl der Lichtquellen (`int getNumberOfLights()`) und auf eine einzelne Lichtquelle (`getLight(int index)`). Ändert sich die Beleuchtungssituation, so werden auch die verwendeten Renderer aktualisiert. Dazu muss die Methode `lightingChanged()` aufgerufen werden.

Lichtquellen sind in der Klasse `LightSource` repräsentiert. Neben Position, Farbe und Richtung (falls benötigt) wird dort auch der Typ der Lichtquelle abgelegt. Aktuell sind folgende Typen umgesetzt:

- Punktlichtquelle (Position, Farbe)
- Richtungslichtquelle (Richtung, Farbe)

Im der Swing Benutzerschnittstelle lassen sich die Lichtquellen verwalten und dynamisch verändern. Den zugehörigen Editor findet man im Menü unter *Rendering* → *Light*. Durch Doppelklick auf einen Wert kann dieser verändert werden.

Die Standart-Lichtquellen sind direkt im Konstruktion der Klasse `CgApplication` gesetzt.

*Achtung: Aktuell sind noch nicht alle Shader umgestellt, um die Lichtquellen der Szene zu verwenden. Manche Shader haben die verwendeten Lichtquellen noch fest im Shadercode verdrahtet. Shader, die die Beleuchtung auf Basis der Lichtquellen berechnen:*

- **Material.SHADER\_PHONG\_SHADING:** Phong Beleuchtungsmodell, Phong Shading Verfahren
  - VS: `shader/vertex_shader_phong_shading.glsl`
  - FS: `shader/fragment_shader_phong_shading.glsl`

- **Material.SHADER\_TEXTURE\_PHONG:** Phone Beleuchtungsmodell auf Textur, Phone Shading Verfahren
  - VS: *shader/vertex\_shader\_texture\_phong\_shading.glsl*
  - FS: *shader/fragment\_shader\_texture\_phong\_shading.glsl*

## 4.5 Texturen und Shader

Shader und Texturen werden je in einem zentralen Ressourcen-Manager verwaltet. Der Zugriff erfolgt so:

```
ResourceManager.getShaderManagerInstance();
```

und

```
ResourceManager.getTextureManagerInstance();
```

### 4.5.1 Texturen

Will man beispielsweise eine Textur verwenden, dann vergibt man dafür eine Id und registriert diese Id und die Textur im zugehörigen Ressourcen-Manager:

```
CgTexture myTexture = new CgTexture("textures/my_texture.png");
String myTextureId = "myId";
ResourceManager.getTextureManagerInstance().addResource(myTextureId,
myTexture);
```

Alternativ kann eine Textur auch direkt aus einem **BufferedImage** erzeugt werden. Die Id verwendet man dann im Material, um die Textur zu verwenden, z.B.:

```
ITriangleMesh myMesh = new TriangleMesh();
myMesh.getMaterial().setTextureId(myTextureId);
```

### 4.5.2 Shader

Das Verwenden von Shadern funktioniert analog. Shader werden durch die Klasse **CgGlslShader** repräsentiert. Zur Konstruktion eines **CgGlslShader**-Objektes übergibt man die Dateinamen des Vertex- und des Fragmentshader-Codes.

Es ist möglich, mehrere Renderdurchgänge hintereinander durchzuführen in denen unterschiedliche Shader verwendet werden. Dazu setzt man mit der Methode

```
getMaterial().addShaderId(String shaderId)
```

einfach mehrere Shader in einem Material.

## 4.6 Picking

*Achtung: Picking wird aktuell nur mit dem JOGL-Rendering-System unterstützt.*

Das cg-Framework unterstützt ein System zum 'Picking' von Objekten (Punkten) im 3D-Fenster und zum Verschieben dieser Objekte. Es können allerdings nicht beliebige Szenenobjekte ausgewählt werden. Stattdessen werden Picking-Items explizit in die Szene eingefügt und mit diesen interagiert.

#### 4.6.1 Erstellen von Picking-Items

Zum Arbeiten mit Picking-Items muss zunächst eine Klasse implementiert werden, die von der abstrakten Klasse **CgApplicationPickable** erbt. Außerdem müssen ein oder mehrere Picking-Items in die Szene eingefügt werden. Zur Repräsentation von Picking-Items gibt es die Klasse **PickingItem**. Ein Picking-Item erzeugt mal also beispielsweise mit

```
PickingItem item = new PickingItem(VectorMatrixFactory.newIVector3(0,0,0));
```

Der dem Konstruktor übergebene Vektor stellt die Position des Picking-Items dar. Um ein Picking-Item in die Szene einzufügen, verwendet man die Methode **addPickingItem** des Objektes, das von **CgApplicationPickable** erbt. In der 3D-Darstellung wird ein Picking-Item als eine (graue) Kugel dargestellt. Die Größe der Kugel muss natürlich der Dimension der Szene angepasst werden. Da diese Anpassung nicht automatisch vorgenommen werden kann, muss der Anwender die Einstellung selber vornehmen. Dies geschieht über ein Singleton-Objekt *Picking*, dass das Picking verwaltet. Auf dieses Objekt greift man mit *Picking.getInstance()* zu. Die Skalierung wird mit dem Aufruf


```
Picking.getInstance().setScaling(0.1);
```

vorgenommen.

#### 4.6.2 Interaktion mit Picking-Items

Picking-Items können ausgewählt und in *x*-, *y*- und *z*-Richtung verschoben werden. Um mit den Picking-Items interagieren zu können, muss zunächst in den Picking-Modus gewechselt werden.

*Achtung: Im Picking-Modus kann die Kamera nicht mehr per Maus gesteuert werden!*

In den Picking-Modus wechselt man, indem man auf das entsprechende Icon in der Schnellstarte-Toolbar auf der linken Seite geklickt wird. . Das Icon verfärbt sich dann in Orange. Befindet man sich im Picking-Modus, dann kann man ein Picking-Item auswählen und verschieben.

*Selektion* Zum Auswählen eines Picking-Items klickt man in der 3D-Szene auf das Item. Ein Item wird dann ausgewählt, wenn man ausreichend genau darauf geklickt hat. Befinden sich mehrere Picking-Items in der gleichen Sichtbarkeitslinie, dann wird das ausgewählt, das der Kamera am nächsten liegt. Das ausgewählte Picking-Item erkennt man daran, dass es gelb eingefärbt ist und dass dafür ein Koordinatensystem gezeichnet wird. Im Koordinatensystem zeigt der rote Pfeil in die *x*-Richtung, der grüne Pfeil in die *y*-Richtung und der blaue Pfeil in die *z*-Richtung.

*Verschieben* Das aktuell selektierte Picking-Item kann in *x*-, *y*- und *z*-Richtung verschoben werden. Die Richtungen sind durch die entsprechenden Pfeile angegeben. Zum Verschieben in eine Richtung drückt man die zugehörige Taste auf der Tastatur, hält die linke Maustaste gedrückt und bewegt die Maus nach links bzw. nach rechts. Eine Mausbewegung nach links bewegt das Picking-Item gegen die Pfeilrichtung, eine Mausbewegung nach rechts bewegt das Picking-Item in Pfeilrichtung. Zur Bewegung in die *x*-Richtung drückt man die *x*-Taste, analog *y* und *z*.

## 5 Verwendung

In diesem Abschnitt werden einige Anwendungsfälle für das Framework vorgestellt. Es ist aber immer möglich, dass im Framework Funktionalität enthalten ist, die hier noch nicht beschrieben wird. Eine zweite Anlaufstation für Anwendungsfälle sind die Beispielanwendungen im Modul **cgresearch.apps**.

### 5.1 Dreiecksnetze

Für Dreiecksnetze existiert das Interface **ITriangleMesh**. Ein Dreiecksnetz (hier Kugel mit einer Tessellierung von 10x10 Flächen) kann so erzeugt und in den Szenengraphen eingefügt werden:

```
ITriangleMesh mesh = TriangleMeshFactory.createSphere(10);
CgNode meshNode = new CgNode(mesh, "mesh");
getCgRootNode().addChild(meshNode);
```

Alternativ können Dreiecksnetze auch 'von Hand' erzeugt werden:

```
ITriangleMesh mesh = new TriangleMesh();
int a = mesh.addVertex(new Vertex(VectorMatrixFactory.newIVector3(1,0,0));
int b = mesh.addVertex(new Vertex(VectorMatrixFactory.newIVector3(0,1,0));
int c = mesh.addVertex(new Vertex(VectorMatrixFactory.newIVector3(0,0,1));
mesh.addTriangle(new Triangle(a,b,c));
```

Informationen zu den existierenden Importern für Dreiecksnetze finden sich im Abschnitt 5.8.

## 5.2 Punktwolken

Für Punktwolken existiert das Interface `IPointCloud`. Eine Punktwolke (hier Würfelvolumen) kann so erzeugt und in den Szenengraphen eingefügt werden:

```
IPointCloud pointCloud = PointCloudFactory.createDummyPointCloud();
CgNode pointCloudNode = new CgNode(pointCloud, "point cloud");
getCgRootNode().addChild(pointCloudNode);
```

Punkt wollen können außerdem aus Dreiecksnetzen erzeugt werden. Dazu werden zufallsverteilt Punkte (hier: 5000) auf der Oberfläche eines Dreiecksnetzes gesampled:

```
IPointCloud pointCloud = MeshSampler.sample(mesh, 5000);
```

Informationen zu den existierenden Importern für Punktwolken finden sich im Abschnitt 5.8.

## 5.3 Kurven

Es sind verschiedene Kurven implementiert, die als gemeinsames Interface `ICurve` verwenden. Für jede Kurve können dabei Kontrollpunkte definiert und ausgelesen werden und der Funktionswert und die Ableitung für beliebige Parameterwerte ausgewertet werden. Aktuell sind folgende Kurventypen implementiert:

- Monom
- Lagrange
- Hermite
- Bezier

## 5.4 Animierte Daten

Zur Verwendung zeitlich veränderlicher Daten gibt es einen `AnimationTimer`. Dieser ist als Singleton umgesetzt und kann folgendermaßen erreicht werden:

```
AnimationTimer animationTimer = AnimationTimer.getInstance();
```

Der Timer verwaltet eine diskrete Zeitleiste mit Startwert (`minValue`), Endwert (`maxValue`) und aktuellem Wert (`value`).

Der Timer kann über verschiedene Wege gesteuert werden:

- Zeitliste als Slider direkt im GUI
- Einstellungen im Menü `Timer`

- direkt über die Singleton-Instanz

Unterstützung für den Timer ist im Szenengraph mit dem Szenengraph-Inhalt **Animation** gegeben:

```
CgNode animationNode = new CgNode(new Animation(), "animation");
```

Dieser Knoten sorgt dafür, dass von seinen Kindern nur das angezeigt wird, dessen Index mit dem aktuellen Wert des **AnimationTimers** übereinstimmt. Für dynamische Daten erzeugt man also je einen Szenengraph-Knoten pro Zeitschritt und fügt diese als Kinder einem Szenengraph-Knoten, der einen **Animation**-Inhalt hat, zu.

Der **AnimationTimer** kann auch dazu 'missbraucht' werden, um verschiedenen Datensätze schnell sichtbar und unsichtbar zu schalten. Hat man mehrere Daten, von denen immer nur genau ein Datensatz angezeigt werden soll, dann kann man die Daten als Kinder eines Animationsknotens in den Szenengraph einfügen. Über den Zeitleisten-Slider kann immer genau ein Datensatz sichtbar geschaltet werden.

## 5.5 Kamera

Die virtuelle Kamera wird über einen Kamera-Controller gesteuert. Aktuell sind folgende Controller implementiert:

- **InspectionCameraController**
- **CameraPathController**
- **FirstPersonCameraPathController**
- **MovableInspectionCameraPathController**

Der Standard-Controller ist der **InspectionCameraController**.

Der **InspectionCameraController** eignet sich besonders zum Betrachten eines Objektes oder einer Szene von allen Seiten. Die Steuerung erfolgt über die Maus. Bei gedrückter linker Maustaste und gleichzeitiger Bewegung der Maus wird die Kamera um den Referenzpunkt der Kamera herum gedreht. Mit gedrückter linker Maustaste kann an das Objekt heran- oder wieder herausgezoomt werden.

Der **CameraPathController** dient dem Abspielen eines gespeicherten Kamerapfades. Der Kamerapfad wird mit Hilfe von Keypoints vorgegeben. Zwischen den Keypoints wird die Position interpoliert. Die Keypoints können über das Menu *Jogl* → *Camera* verwaltet werden (Hinzufügen und Entfernen von Keypoints). Ist der **CameraPathController** selektiert, dann wird der Kamerapfad über die Zeitleiste des **AnimationTimers** kontrolliert. Der vollständige Pfad läuft über die gesamte Länge der Zeitleiste - darüber kann also auch die Auflösung der Bewegung und die Geschwindigkeit der Kamerabewegung festgelegt werden.

Alle Einstellungen zur Kamera sind im Camera-Singleton abgelegt. Darauf lässt sich mit

```
Camera.getInstance()
```

zugreifen. In der Kamera kann auch der aktuelle Controller gesetzt werden. Alternativ lassen sich die meisten Kameraeinstellungen auch per GUI im Menu unter *Jogl* → *Camera* vornehmen.

## 5.6 Film-Export

Es ist möglich, die 3D-Darstellung zu exportieren. Konkret kann jeder Zeitschritt als ein Einzelbild exportiert werden. Die Einzelbilder können dann über externe Tools zu Videos zusammengestellt werden. Achtung: Der Export der Einzelbilder kostet einige Zeit. Es ist daher ggf. notwendig, den **AnimationTimer** mit einer geringeren Geschwindigkeit laufen zu lassen, damit auch wirklich von jedem Frame ein Bild exportiert wird. Das Exportieren der Einzelbilder kann über das Menu *Jogl* → *Movie Export* gestartet und wieder beendet werden. Dort lässt sich auch der Ausgabepfad festlegen.

## 5.7 Octree

Zur effizienten Bearbeitung von Szenen eignen sich oft hierarchische Datenstrukturen. Im Framework sind dazu bislang Octrees umgesetzt. Die Octrees sind generisch und wissen zunächst nicht, welche Daten sie verwalten. Zum Aufbauen eines Octrees benötigt man daher ein Objekt, dass mit den konkreten Daten umgeht. Hier ist dies mit den Strategy-Entwurfsmuster umgesetzt. Ein Strategie-Objekt wird verwendet, um eine Factory (Fabrik) zu erzeugen, die den Octree aufbaut. Es existieren aktuell Strategien für Dreiecksnetze und Punktwolken. Einen Octree für ein Dreiecksnetz erzeugt man so (maximale Tiefe des Baumes 7 und Teilungsgröße einer Zelle 20):

```
OctreeFactoryStrategyTriangleMesh octreeFactoryStrategyMesh =  
    new OctreeFactoryStrategyTriangleMesh(mesh);  
OctreeFactory<Integer> octreeFactoryMesh =  
    new OctreeFactory<Integer>(octreeFactoryStrategyMesh);  
OctreeNode<Integer> octreeMeshRoot = octreeFactoryMesh.create(7, 20);
```

## 5.8 Daten-Im- und Export

Für verschiedenen Datentypen wird der Im- und Export unterstützt:

### 5.8.1 Dreiecksnetze

- OBJ-Import: Klasse `ObjFileReader`
- OBJ-Export: Klasse `ObjFileWriter`
- PLY-Import: Klasse `PlyFileReader`

Beispielhafter Import aus Obj-Datei:

```
ObjFileReader reader = new ObjFileReader();  
CgNode meshNode = reader.readFile("meshes/bunny.obj");
```

Das Ergebnis ist eine `CgNode`, die importierten Meshes aus der Obj-Datei sind als Kindknoten dieses Knotens abgelegt.

### 5.8.2 Motion-Capture-Daten

- Daten aus dem HAW Wellefeldsynthese-Labor: Klasse `MoCapImporter`

### 5.8.3 Punktwolken

- ASCII-Daten-Import: Klasse `AsciiPointsReader`
- ASCII-Daten-Export: Klasse `AsciiPointsWriter`

Beim Import von ASCII-Daten gibt es verschiedene Formate in denen die Punktinformationen vorliegen können. Um diese mit einem Importer abdecken zu können, muss beim Import das Format mit angegeben werden. Das Format wird über ein Objekt der Klasse `AsciiPointFormat` repräsentiert. Es wird davon ausgegangen, dass die Ascii-Daten zeilenweise mit einem konstanten Trennungszeichen angegeben sind. Alle Zeilen, die das Format erfüllen, werden importiert. Um beispielsweise Daten im Format

<Position-X> <Position-Y> <Position-Z> <Normale-X> <Normale-Y> <Normale-Z>

mit einem Leerzeichen als Trennungszeichen einzulesen, wird folgendes Formatierungsobjekt erzeugt:

```
AsciiPointFormat format = new AsciiPointFormat()  
    .setPosition(0, 1, 2).setNormal(3, 4, 5).setSeparator("\\s+");
```

Die Parameter geben die Position des entsprechenden Wertes in der Datenzeile an. Der Konstruktor erzeugt ein Objekt und mit den Methoden wird das Objekt angepasst und jeweils wieder zurückgegeben (method chaining). Damit ist es möglich, das Objekt mit einem Ausdruck zu erzeugen. Außerdem können Farbinformation (RGB) und eine Skalierung der Farbwerte angegeben werden.

Der Export arbeitet analog.

## 6 Projekte

### 6.1 3D Scanner

#### 6.1.1 Abhängigkeiten

- cg
- JOGL
- Tinkerforge-Bibliotheken (Unterverzeichnis lib)

#### 6.1.2 Installation

Zunächst müssen die Projekte cg (Abschnitt 2) und JOGL (Abschnitt 4.3.1) eingerichtet werden. und das Git-Repository ([git.informatik.haw-hamburg.de/srv/git/computergrafik/cg\\_3dscanner](https://git.informatik.haw-hamburg.de/srv/git/computergrafik/cg_3dscanner)) gecloned werden (siehe Abschnitt 7.1.5). Dann wird auch das Projekt *3dscanner* in Eclipse importiert. Als Compiler und für die Laufzeitbibliothek muss wegen der Abhängigkeit von JOGL Java 1.6 verwendet werden.

Setzen Sie die notwendigen Ressourcenpfade wie unter 3.6 beschrieben.

#### 6.1.3 Einrichten des USB-Seriell-Adapters

Der Treiber (Beispiel Windows XP) befindet sich im Projektverzeichnis unter *software/software/usb\_rs232/Usb\_to\_Rs232-2303\_includ\_2\_IC/win98-winme.vista.win2000-winxp*. Unter Windows wird die Hardware beim Einstecken automatisch erkannt. Der Treiber kann direkt über den Hardware-Wizard installiert werden. Es muss nur das Verzeichnis angegeben werden. Abbildung 5 beschreibt, wie überprüft wird, ob korrekt ein COM-Port angelegt wurde. Achtung, je nach Betriebssystem, kann es sein, dass beim Neustart eines Rechners ein anderer COM-Port vergeben wird.

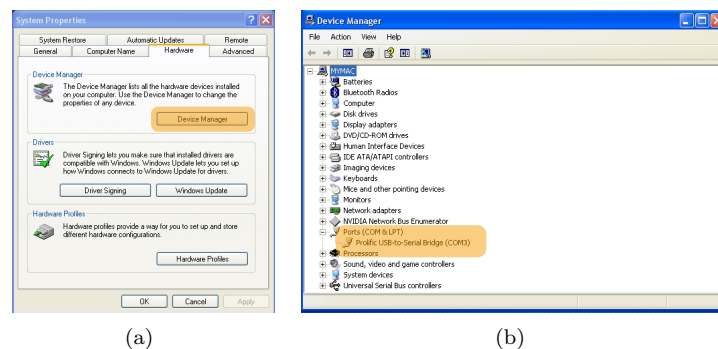


Abbildung 5: Prüfung, ob nach der Treiberinstallation ein neuer COM-Port angelegt wurde. a) Systemeinstellungsmenu unter Windows, Öffnen des *Device Managers*. b) Es sollte sich ein COM-Port unter dem Menüpunkt *Ports (COM&LPT)* befinden.



### 6.1.4 Testen des Distanzsensors

Die Software zum Testen des Distanzsensors befindet sich im Projektverzeichnis um Unterverzeichnis *software/laser\_distance\_sensor*. Die Installation wird über *setup.exe* gestartet. Abbildung 6 zeigt, wie die mitgelieferte Software verwendet werden kann, um den Distanzsensor zu testen.

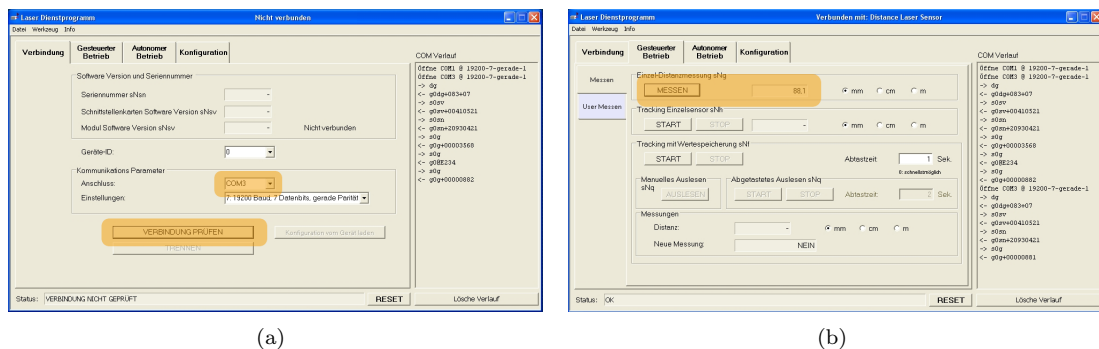


Abbildung 6: Ausführen der installierten Software. a) Auswahl des korrekten COM-Ports und Prüfen der Verbindung (dabei wird die Verbindung aufgebaut). b) Ausführen einer Entfernungsmessung, der gemessene Distanzwert wird direkt angezeigt.

## 6.2 Urban Reconstruction

### 6.2.1 Abhängigkeiten

- cg
- JOGL

### 6.2.2 Installation

Zunächst müssen die Projekte cg (Abschnitt 2) und JOGL (Abschnitt 4.3.1) eingerichtet werden. und das Git-Repository ([git.informatik.haw-hamburg.de/srv/git/computergrafik/cg-urbanreconstruction](https://git.informatik.haw-hamburg.de/srv/git/computergrafik/cg-urbanreconstruction)) gecloned werden (siehe Abschnitt 7.1.5). Dann wird auch das Projekt *3dscanner* in Eclipse importiert. Als Compiler und für die Laufzeitbibliothek muss wegen der Abhängigkeit von JOGL Java 1.6 verwendet werden. Setzen Sie die notwendigen Ressourcenpfade wie unter 3.6 beschrieben.

## 7 Best Practises

### 7.1 Git

#### 7.1.1 Anlegen eines neuen lokalen Repositories

- `mkdir <Verzeichnisname>`
- `cd <Verzeichnisname>`
- `git init`

### 7.1.2 Anlegen eines neuen Repositories auf dem Server

- `mkdir <Verzeichnisname>`
- `cd <Verzeichnisname>`
- `git init --bare`

Beispiel:

- `mkdir cg`
- `cd cg`
- `git init --bare`

### 7.1.3 Setzen eines Remote Repositories

- `git remote add <Name des Remote Repositories> <Pfad zum Repository>`
- Beispiel (gleicher Rechner): `git remote add local /Users/abo781/repository/cg`
- Beispiel (Server): `git remote add server ssh://abo781@git.informatik.haw-hamburg.de/srv/git/computergrafik/cg`

### 7.1.4 Lokale Änderungen an Server Repository senden

- `git push <Name des Remote Repositories> <Name des Branches>`
- Beispiel: `git push server master`

### 7.1.5 Lokales Klon-Repository von Server holen

- `git clone <Name des Remote Repositories>`
- Beispiel: `git clone ssh://abo781@git.informatik.haw-hamburg.de/srv/git/computergrafik/cg`

## 7.2 Sonar

Starten von Sonar mit

```
cd <Sonar-Verzeichnis>/bin/<Betriebssystem>/
./sonar.sh start
```

Prüfen, ob der Sonar-Server läuft (Im Browser):

`http://localhost:9000/`

Sonar-Analyse durchführen (benötigt Maven-Installation).

```
mvn sonar:sonar
```