# Dokumentation

ENEMY OF JAMES BOND

CG2LU, SS 2007
Reiterer, Stefan, 033532, 0525940, e0525940@student.tuwien.ac.at
Dinhobl, Erhard, 033532, 0525938, e0525938@student.tuwien.ac.at

**Inhaltsverzeichnis**

# 1 Gameplay

Dieses Spiel beruht auf dem 21. James Bond Streifen, mit dem Titel Casino Royal. James Bond ist nach Madagaskar gereist, um dort einen international operierenden Bombenbauer zu verfolgen. Bei dieser Mission wird 007 von einem anderen MI6-Agenten unterstützt. Als sie den Bombenbauer auf einem belebten Platz aufspüren, beginnen sie mit der Observation. Bonds Assistent mischt sich dabei unter die Menge, um näher an die Zielperson heranzukommen. Er ist dabei die ganze Zeit mit einem Funkgerät, mit James verbunden.
Auf einmal passiert etwas Unerwartetes. Der Assistent greift sich ans Ohr, um sein Funkgerät zu richten. Dies bleibt allerdings nicht unentdeckt. Der Bombenbauer begreift in diesem Moment, dass er observiert wird, und ergreift die Flucht. James reagiert sofort und beginnt die Verfolgung.

Und genau hier beginnt unser Spiel. Der Spieler schlüpft in die Rolle des Bombenbauers und muss die Botschaft erreichen, ohne dass er von James Bond erwischt wird.

## *1.1 Menü*

Wenn man das Spiel startet, gelangt man zuerst in das Menü. Die Navigation zwischen den Menüpunkten erfolgt mittels Pfeiltasten. Mittels Leertaste kann ein Menüpunkt betätigt werden.
Das Menü ist folgendermaßen aufgebaut:

### 1.1.1 Start

Wenn der Spieler diesen Eintrag betätigt wird das Spiel gestartet und man gelangt in das erste Level.

### 1.1.2 Help

Wenn der Spieler diesen Eintrag betätigt, wird eine kurze Hilfe angezeigt.

### 1.1.3 Exit

Mit diesem Menüpunkt kann das Spiel beendet werden.

## *1.2 Erstes Level*

Das fertige Spiel wurde mit zwei Levels ausgestattet. Das erste Level beinhaltet eine Baustelle. Hier sind folgende Objekte zu sehen:

### 1.2.1 Baustellentoilette (Toi Toi)

Hierbei handelt es sich um ein Objekt mit einer komplexen Form. Die Toilette besitzt einige runde und gebogene Oberflächen. Dadurch ist das Per-Pixel Lighting, welches auf dieses Objekt angewendet wird, besonders gut zu sehen. Genaueres darüber siehe Kapitel Spezialeffekte.

### 1.2.2 Baustellenkräne

Der Spieler kann auf Kräne klettern und darauf herumlaufen. Dies funktioniert momentan allerdings nicht nicht so richtig, weil wir noch Probleme mit PhysX haben.

### 1.2.3 Rohbauten

Der erstel Rohbau besteht aus Säulen und Deckenplatten. Hier sind keine Treppenhäuser vorhanden. Wenn man auf das Dach kommen will, muss man zuerst auf einen Kran springen und dann vom Kran auf das Dach des Gebäudes springen. Auf diesem Gebäude befindet sich das Ziel

Der zweite Rohbau ist etwas komplexer. Er besteht aus mehreren Stockwerken. Um auf das Dach zu kommen, kann man die Treppen verwenden. Das Gebäude hat zwei Treppenhäuser. Beide führen vom Erdgeschoß bis zum Dach.

### 1.2.4 Ziel

Hierbei handelt es sich um eine Tafel mit der Aufschrift Botschaft. Wenn der Spieler diese Tafel erreicht, hat er das erste Level erfolgreich beendet und kommt in das nächste Level.

### 1.2.5 Infoschild

Hierbei handelt es sich um ein Schild mit der Aufschrift „Achtung Baustelle“.

### 1.2.6 Enemy

Dieses Objekt repräsentiert unsere Spielfigur. Um den Enemy zu sehen, muss man zuerst mittels V-Taste in die Third Person Ansicht umschalten. Auch auf dieses Objekt wird der Spezialeffekt Per-Pixel Lighting angewendet.

### 1.2.7 Bond

Dieses Objekt repräsentiert James Bond. Er bewegt sich mit einer bestimmten Geschwindigkeit auf den Spieler zu. Damit James nicht an eine Wand anläuft, bzw. durch eine Wand hindurch läuft, bewegt er sich den gleichen Pfad entlang wie der Spieler. Wenn er sich bis zu einer gewissen Distanz dem Enemy nähert, hat man verloren. Auch hier wird der Spezialeffekt Per-Pixel Lighting angewendet.

Wenn man dieses Level erfolgreich absolviert hat, gelangt man in das zweite Level.

## *1.3 Zweites Level*

In diesem Level geht es darum das Botschaftsgebäude zu erreichen, ohne von Bond erwischt zu werden. Doch bis dort hin ist es ein weiter Weg und es stehen einem viele Objekte im Weg.
Das zweite Level beinhaltet folgende Objekte:

### 1.3.1 Rohbau

Hierbei handelte es sich um ein in Bau befindliches Gebäude, bei welchem bisher nur die Wände des Erdgeschoßes fertig gestellt wurden. Der Spieler muss sich einen Weg durch das Erdgeschoß suchen, um zum Botschaftsgebäude zu gelangen. Man muss dabei allerdings sehr aufmerksam sein, da nicht jeder weg zum Ausgang führt.

### 1.3.2 Botschaft

Dieses Objekt repräsentiert das Botschaftsgebäude. Vor diesem Bauwerk befindet sich eine Tafel mit der Aufschrift „Botschaft“. Wenn der Spieler diese Tafel erreicht, hat er das Spiel gewonnen.

### 1.3.3 Sonstige Objekte

Weiters befinden sich in diesem Level noch Baustellenkräne, Baustellentoiletten, der Enemy und James Bond. Für eine genauere Beschreibung zu diesen Objekten siehe Kapitel 1.2 (erstes Level).

# 2 Effekte

In diesem Kapitel sind die Spezialeffekte angeführt, die wir in unserem Spiel implementiert haben.

## 2.1 Per-Pixel Lighting (Phong-Beleuchtung im Fragment Shader) implementiert von Stefan Reiterer

Für diesen Effekt mussten wir einen Vertexshader und Fragmentshader implementieren und in unser Spiel einbinden. Aus diesem Grund haben wir die Struktur des Spiels so angelegt, dass man mehrere verschiedene Shader einbinden kann. Für jedes Objekt existiert eine INI-Datei. In diesen Dateien kann man festlegen, welche Shader auf welche Objekte angewendet werden.
Da allerdings nicht alle Grafikkarten Vertex Shader, bzw. Fragment Shader unterstützen, wird beim Start mittels Glew überprüft, ob die Grafikkarte das Ganze überhaupt unterstützt.
Wenn sie Shader unterstützt, legen wir die Shaderobjekte für Fragmentshader und Vertexshader an. Anschließend wird der Source-Code der Shader, welcher sich in einer Textdatei befindet, eingelesen. In einem weiteren Schritt wird der eingelesene Shader kompiliert und an ein Programm gebunden.
Beim Rendern der einzelnen Objekte wird in der INI-Datei nachgesehen, ob ein Shader auf das Objekt angewendet werden soll. Je nach dem wird dann der Shader mit dem Befehl `glUseProgramObjectARB()` auf das Objekt angewendet.

Da Shader großen Einfluss auf die Performance des Spiels haben, werden sie nur auf folgende Objekte angewendet.

- Baustellentoilette (Toi Toi)
  Der Phong-Shader wird auf die abgerundeten Ecken der Toilette angewendet.

- Enemy
  Hier wird der Shader auf das ganze Objekt angewendet.

- James Bond
  Wie auch beim Enemy, wird auch hier der Shader auf das ganze Objekt angewendet.

Folgende Grafik zeigt die Baustellentoilette mit Per-Pixel Lighting.


*Abbildung 1: Baustellentoilette mit Per-Pixel Lighting*

### 2.1.1 Fragmentshader

Hier findet die Berechnung von Ambient, Diffuse und Specular Light statt. Das Ambient Light bekommt man direkt aus der Lichtquelle. Das Diffuse Light ergibt sich aus dem Cosinuswinkel zwischen Normalvektor und Lightvector (Lambert). Und auch das Specular Light wird, wie bereits in CG1 gelernt, berechnet.

```
varying vec3 normal;
varying vec3 view;
varying vec3 lightdirection;

void main(void)
{
 vec3 V = normalize(-view);
 vec3 R = normalize(-reflect(lightdirection,normal));

 vec4 Ambient        = gl_LightSource[0].ambient;
 vec4 Diffuse        = gl_LightSource[0].diffuse * max(dot(normal,
lightdirection), 0.0);

 vec4 Specular       = gl_LightSource[0].specular * pow(max(dot(R, V),
0.0), gl_FrontMaterial.shininess);

 gl_FragColor = gl_FrontLightModelProduct.sceneColor + Ambient + Diffuse
+ Specular;
}
```

### 2.1.2 Referenzen

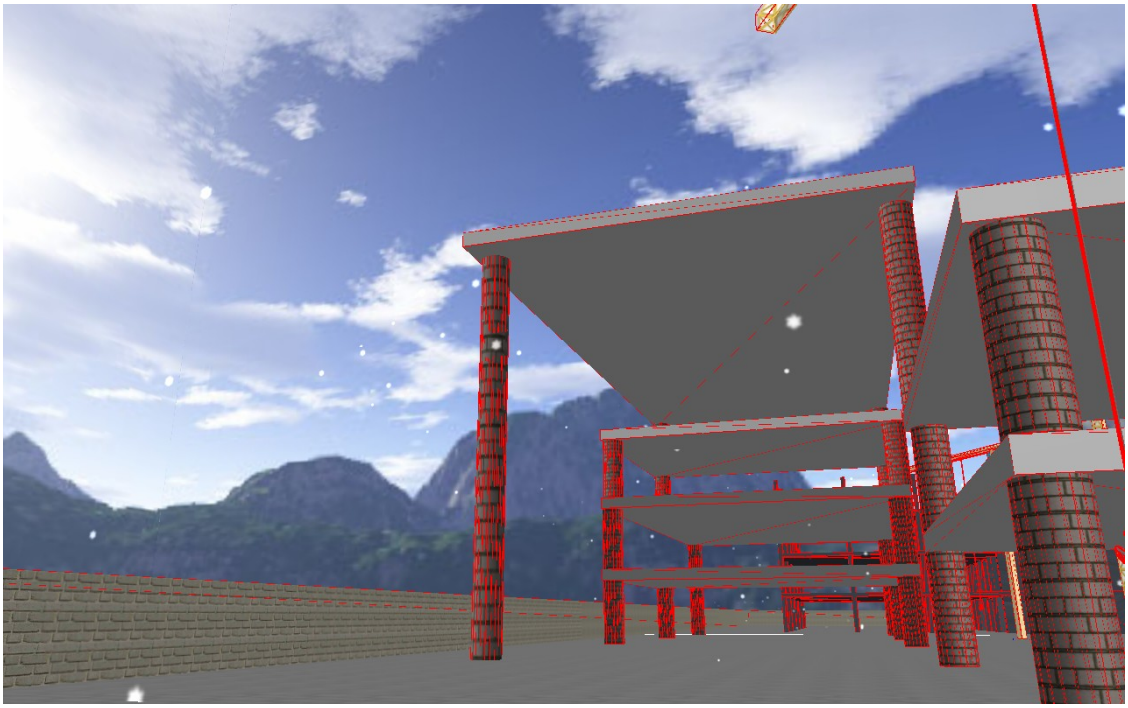
Bevor ich mit der implementierung des Effektes beginnen konnte, musste ich mich mal informieren, was Per-Pixel Lighting überhaupt ist und wie man in OpenGL Shader implementert. Dazu habe ich folgende Referenzen verwendet:

- http://wiki.delphigl.com/index.php/Tutorial_glsl2
  Hier wird unter anderem beschrieben was Per-Pixel Lighting ist und was Shader sind.
- http://www.lighthouse3d.com/opengl/glsl/index.php?ogloverview
  Hier wird beschrieben wie man shader verwendet.
- Red Book
  Hier wird beschrieben was die OpenGL Shading Language ist und wie man Shader verwendet

## *2.2 Partikelsystem implementiert von Erhard Dinhobl*

Es wurde ein Partikelsystem entwickelt welches ein einfaches Regnen/Schneien darstellen soll. Basis hierfür ist eine Textur, welche wiederholt für die einzelnen Partikel verwendet wird. Es wird ein Partikel angelegt, welcher genau einen Regentropfen/Schneeflocke darstellt. Für diesen kann Position/Fluggeschwindigkeit/Lebenszeit/Größe festgesetzt werden. Es anfangs eine bestimmte Anzahl an Partikel anfangs angelegt, welche dann von einem Emitter verwaltet werden. In jedem Frame wird überprüft, ob die Lebenszeit vom einem Partikel abgelaufen ist, wenn ja wird dieses als deaktiviert gesetzt und in einem Update-Zyklus

des Emitters dann aus dessen Liste gelöscht. So werden wieder Plätze frei für neue Partikel.



Beispiel für das Partikelsystem

### 2.2.1 Referenzen

- http://www.codeworx.org/opengl_par1.php
  Für ein Beispiel eines Partikelsystems
- Red Book
  Für einige Funktionserklährungen vom Beispiel von codeworx.org und einige zusätzliche Dinge.

# 3 Animierte Objekte

Zusätzlich zu den bewegten Objekten (Baustellenkräne, James Bond, Enemy), haben wir nun auch animierte Objekte implementiert. Bei diesen Objekten handelt es sich um den Enemy und James Bond. Damit die Bewegung realistischer aussieht, rotieren nun auch die Beine der Spielfiguren.
Hierfür haben wird diese Objekte so modelliert, dass sich der Drehpunkt der Beine im Ursprung befindet. In der folgenden Grafik kann man den Aufbau der Modelle gut erkennen.

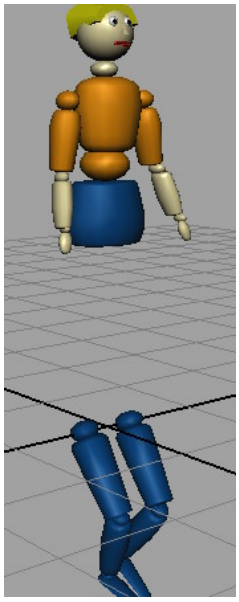


*Abbildung 2: Modell von James Bond in Maya*

Beim Rendern werden die Beine um den Ursprung rotiert und an die richtige Position verschoben.

# 4 Vertex-Repräsentation

Mittels F6-Taste kann man zwischen folgenden Modi hin- und herschalten.

## *4.1 Vertex-Arrays*

Die aus den FBX-Dateien ausgelesenen Vertices wurden in Arrays vom Typ Glfloat übernommen. Beim Rendern haben wir Pointer auf diese Arrays gesetzt und gezeichnet.

Mit folgendem Befehl wurde der Pointer auf das Array gesetzt:

```
glVertexPointer (3, GL_FLOAT, 3 * sizeof(GLfloat), mesh[i]);
```

Das Array „mesh“ ist in unserem Fall folgendermaßen definiert:

```
vector<GLfloat*> mesh;
```

Mesh besteht aus mehreren Arrays vom Typ Glfloat. Für jedes Objekt, aus denen ein komplexes Objekt besteht gibt es ein Array.

Anschließend wurden die Vertices mit folgendem Befehl gezeichnet:

```
glDrawArrays(GL_TRIANGLES, 0, iNumVertices[i]);
```

## 4.2 ARB_vertex_buffer_object (VBOs)

Um die Performance unseres Spiels zu steigern, verwenden wir Vertex Buffer Objects. Somit werden die Vertices in den Speicher der Grafikkarte übernommen und es muss nicht jedes Mal auf den Hauptspeicher zugegriffen werden.

Beim Start des Spiels werden die Bufferobjekte für die Vertices, Normalvektoren und UV-Koordinaten erstellt und initialisiert. Dies geschieht folgendermaßen:

```
void CGObject::genVertexBuffer()
{
	int i=0;

	for(int i = 0; i < iNumVertices.size(); i++)
	{
		GLuint vertexbuf;
		GLuint texturebuf;
		GLuint normalbuf;

		glGenBuffersARB(1, &vertexbuf);
		vertbuffer.push_back(vertexbuf);

		glBindBufferARB(GL_ARRAY_BUFFER_ARB, vertbuffer[i]);

		/* Daten laden
		 */
		glBufferDataARB(GL_ARRAY_BUFFER_ARB,
iNumVertices[i]*3*sizeof(GLfloat), mesh[i],GL_STATIC_DRAW_ARB);

		glGenBuffersARB(1, &normalbuf);
		normbuffer.push_back(normalbuf);

		glBindBufferARB(GL_ARRAY_BUFFER_ARB, normbuffer[i]);

		/* Daten laden
		 */
		glBufferDataARB(GL_ARRAY_BUFFER_ARB,
iNumVertices[i]*3*sizeof(GLfloat), meshnormals[i],GL_STATIC_DRAW_ARB);

	    /* Texturkoordinaten
	       */
	    glGenBuffersARB(1, &texturebuf);
	    texbuffer.push_back(texturebuf);

	    glBindBufferARB(GL_ARRAY_BUFFER_ARB, texbuffer[i]);

	    /* Daten laden
	       */
	    glBufferDataARB(GL_ARRAY_BUFFER_ARB,
iNumVertices[i]*2*sizeof(GLfloat), meshtextures[i],GL_STATIC_DRAW_ARB);

	    glBindBufferARB(GL_ARRAY_BUFFER_ARB, NULL);
	}
}
```

Die Namen (IDs) der Bufferobjekte werden in die Arrays `vertbuffer, normbuffer und texbuffer` gespeichert. Für jedes Objekt wurde nun ein ARB_vertex_buffer_object erstellt.
Beim Rendern werden nur die Pointer richtig gesetzt und die Objekte anschließend gezeichnet.

```
glBindBufferARB(GL_ARRAY_BUFFER_ARB, vertbuffer[i]);

/* Vertex-Pointer wird auf Vertex-Buffer gesetzt
*/
glVertexPointer(3, GL_FLOAT, 0, (char *) NULL);

glBindBufferARB(GL_ARRAY_BUFFER_ARB, texbuffer[i]);

/* Textur-Pointer wird auf Textur-Buffer gesetzt
*/
glTexCoordPointer(2, GL_FLOAT, 0, (char *) NULL);

glBindBufferARB(GL_ARRAY_BUFFER_ARB, normbuffer[i]);

/* Vertex-Pointer wird auf Vertex-Buffer gesetzt
*/
glNormalPointer(GL_FLOAT, 0, (char *) NULL);

glDrawArrays(GL_TRIANGLES, 0, iNumVertices[i]);
```

### 4.2.1 Immediate Mode

Dieser Modus ist bezüglich der Performance am schlechtesten. Der Grund dafür ist, dass jeder Vertex einzeln gezeichnet wird. Dies geschieht in unserem Spiel folgendermaßen:

```
void CGObject::drawImmediateMode(int i)
{
	int w = 0;

	glBegin(GL_TRIANGLES);
	cout << iNumVertices[i] << endl;
	for (int j = 0; j < iNumVertices[i]*3; j+=3)
	{
		glNormal3f (meshnormals[i][j], meshnormals[i][j+1],
meshnormals[i][j+2]);
		glTexCoord2f (meshtextures[i][w], meshtextures[i][w+1]);
		glVertex3f (mesh[i][j], mesh[i][j+1], mesh[i][j+2]);

		w += 2;
	}
	glEnd();
}
```

# Wireframe Modus / Framerate und andere Debug Ausgaben

Mittels der F-Tasten kann man verschiedene Modi und Debugausgaben ein, bzw. ausschalten.

## *4.3 F1-Taste*

Mit Hilfe der F1-Taste kann man die Hilfe einschalten. Wenn die Hilfe eingeschaltet ist, wird ein schwarzer Hintergrund, auf welchem der Hilfetext ersichtlich ist, angezeigt. Der Text wird aus einer Textdatei eingelesen.

## *4.4 F2-Taste*

Diese Taste dient zum Einschalten der Frameratenanzeige.

## *4.5 F3-Taste*

Hiermit kann der Wireframemodus aktiviert werden. Dieser Modus wird mit dem Befehl `glPolygonMode()` aktiviert.

## *4.6 F4-Taste*

Diese Taste dient zum Umschalten der Texturqualität. Man kann zwischen „Nearest Neighbour“ und „Bilinear“ wählen.

## *4.7 F5-Taste*

Diese Taste dient zum Umschalten der Texturqualität. Man kann zwischen „Aus“, „Nearest Neighbour“ und „Linear“ wählen.

## *4.8 F6-Taste*

Genaueres dazu siehe Kapitel 4.

## *4.9 F7-Taste*

Diese Taste dient zur Aktivierung der Displaylists. Dadurch ist es möglich mehrere Kommandos für eine spätere Ausführung zu speichern. Die meisten Grafikkarten speichern diese Listen in einem bestimmten Speicherbereich in einer optimierten Form ab.

Beim Start des Spiels werden die Displaylists angelegt und initialisiert. Dies geschieht in der Funktion `void CGObject::setDisplayList()`.
Beim Rendern wird mittels `glCallList(list[i])` die Displaylist gezeichnet.

Durch die Verwendung der Displaylists wird die Performance unseres Spiels merkbar erhöht.

## 4.10 F8-Taste

Bei dieser Taste sollte das Frustum-Culling aktiviert werden. Dieser Punkt wurde allerdings aus Zeitgründen nicht implementiert.

## 4.11 F9-Taste

Mit Hilfe dieser Taste kann man die Transparenz aktivieren. Wenn sie aktiviert ist, kann man zwischen den Streben der Kräne durchsehen. Folgende Grafik zeigt, die Baustellenkräne, wenn die Transparenz aus-, bzw. eingeschaltet ist.

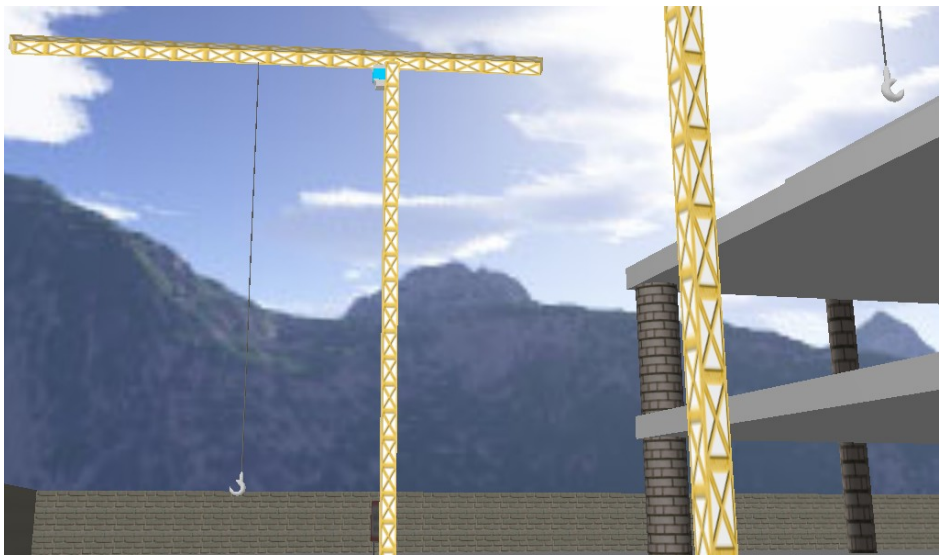


*Abbildung 3: Kräne, wenn die Transparenz ausgeschaltet ist*

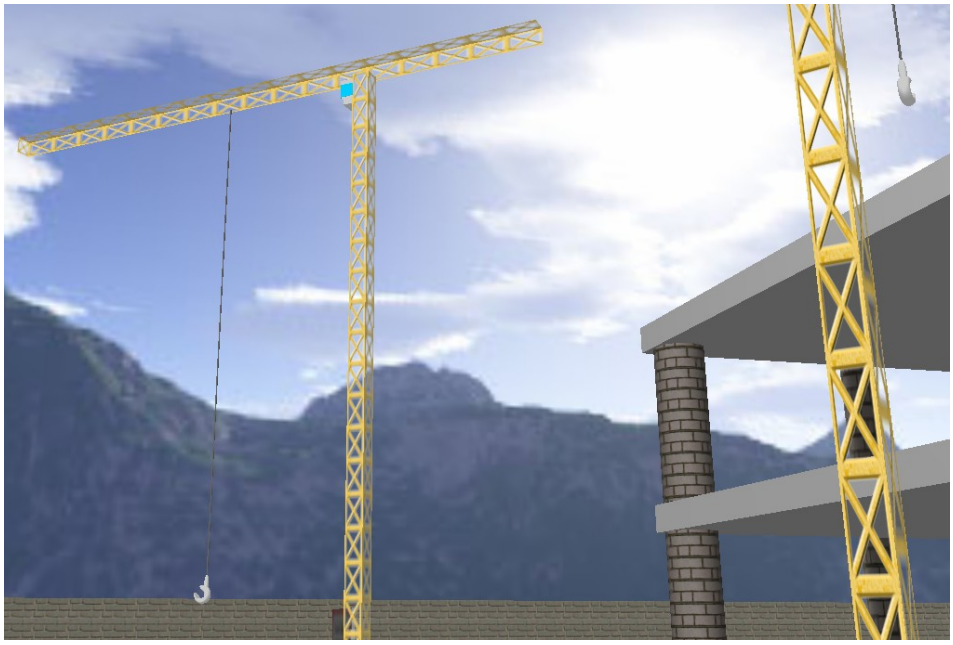


*Abbildung 4: Kräne, wenn Transparenz aktiviert ist*

# 5 Funktionierende Steuerung

Die Steuerung erfolgt mit Maus und Tastatur. Mit Hilfe der Tasten W, S, A und D kann der Enemy nach vor, nach hinten, nach links und nach rechts bewegt werden. Weiters kann man durch das betätigen der Leertaste springen.

Mittels V-Taste kann man zwischen den Ansichten “First Person”, “First Person (backview)” und “Third Person” navigieren.

Mittels M-Taste kann der Effekt Motion Blur aktiviert werden.

Das Spiel kann mit Hilfe der ESC-Taste beendet werden.

Mittels Maus kann die Blick- und Laufrichtung verändert werden. Wenn die Maus vertikal bewegt wird, kann der Spieler nach oben bzw. unten sehen.
Wenn die Maus horizontal bewegt wird, kann der Spieler nach rechts bzw. links sehen. Außerdem dreht sich die Spielfigur in diese Richtung mit und es wird dadurch die Laufrichtung geändert.

Wenn man die Verfolgung durch James Bond ausschalten will, braucht man nur die F-Taste drücken (cheat).

## *5.1 Besonderheiten im Gameplay*

Um im ersten Level das Ziel zu erreichen, muss man vom Dach des 2. Rohbaus auf einen Baustellenkran springen und von dem auf das Dach des 1. Rohbaus.
Am Dach des 2. Rohbaus sollte man noch auf einen Stahlträger springen, um weiter nach boen zu gelangen. Folgende Grafik zeigt, wie sich der Spieler auf einem solchen Träger befindet.

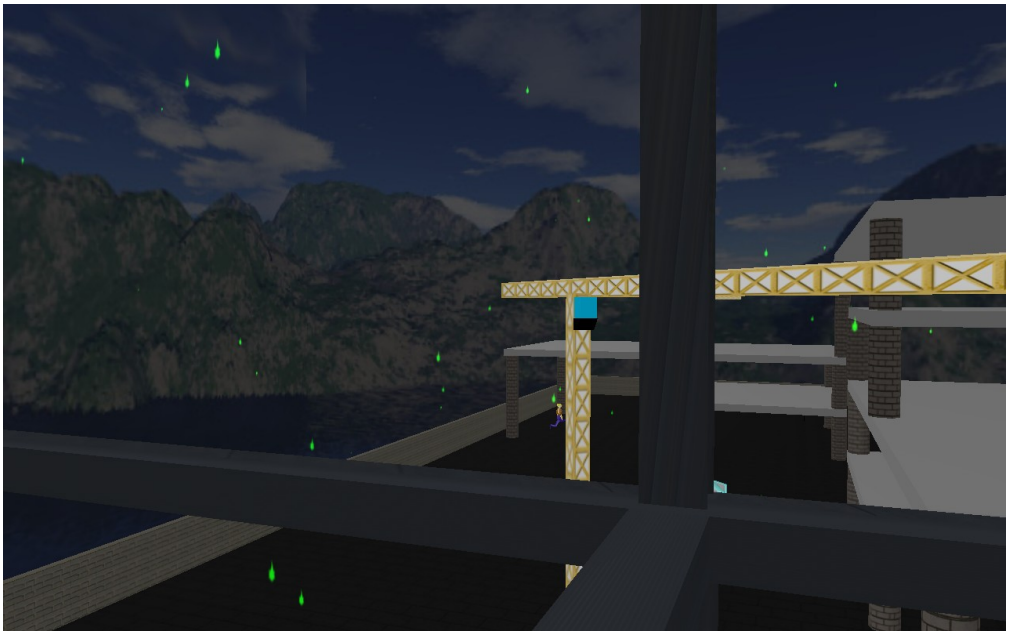


*Abbildung 5: Enemy befindet sich auf Stahlträger des 2. Rohbaus*

# 6 Besonderheiten

Um die Geometriedaten unserer Objekte einzulesen verwenden wir einen FBX-Loader. Unsere Objekte wurden mit Hilfe von Autodesk Maya 8.0 erstellt und in FBX-Dateien exportiert. Beim Start des Spiels werden alle Daten aus diesen Dateien eingelesen, die wir zur Darstellung der Objekte benötigen.

Dies sind:

- Eckpunkte
- Normalvektoren
- Texturkoordinaten
- Oberflächenfarben

# 7 Zusatztools

Zur Realisierung dieses Spiels wurden folgende Zusatztools verwendet.

## *7.1 GLUT*

Diese Library dient uns zur Erstellung des Fensters, und zur Definition der Callbackfunctions für das Rendering, die Tastaturabfrage und Mausabfrage.
Hierfür war folgendes Tutorial für uns sehr nützlich:
http://www.lighthouse3d.com/opengl/glut/

## *7.2 FMOD*

Diese Library dient uns zur Ausgabe des Sounds. Beim Start des Spiels wird der Sound initialisiert und eine mp3-Datei wiederholt abgespielt.
Die Musterbeispiele, die bei dieser Library dabei sind, waren für die Implementierung sehr nützlich. Fmod ist erhältlich unter: http://www.fmod.de/

## *7.3 PhysX*

Diese Library dient uns zur Realisierung der collision-detection. Dazu verwenden wir parallel zur Geometriewelt eine Physikwelt. In dieser Physikwelt werden alle statischen Objekte aus der Geometriewelt mit Hilfe von „dynamic triangle meshes“ nachgebildet.

Der Enemy wurde in der Physikwelt als Kugel modelliert und wird mittels Forcevector gesteuert. Die Tastatureingabe verändert den Forcevector. Wenn der Enemy gerendert wird holen wir uns dessen Position einfach aus der Physikwelt heraus.

Momentan gibt es allerdings noch ein paar Probleme bei der Steuerung. So kann man zum Beispiel noch nicht mit den Baustellenkränen interagieren. Wir werden versuchen dieses Problem bis zum 2. Spielevent in den Griff zu bekommen.

Nützliche Hinweise für die Implementierung haben wir aus der PhysX-Hilfe und aus den Musterbeispielen, welche bei PhysX dabei sind, erhalten.

# 8 Erstellung der Modelle

Unsere Modelle wurden mit Hilfe von Autodesk Maya 8.0 modelliert. Hierfür war der vom Institut für Computergraphik angebotene Maya-Kurs sehr hilfreich. Nach dem wir die Objekte erstellt, texturiert und eingefärbt haben, exportierten wir sie in FBX-Dateien.