

# Theory Exploration

Johannes Diemke

Carl von Ossietzky Universität Oldenburg  
johannes.diemke@informatik.uni-oldenburg.de

**Zusammenfassung.** Theory Exploration ist ein Abschlussprojekt im Rahmen des Moduls OpenGL mit Java im Sommersemester 2007. Basierend auf der für dieses Modul entwickelten Photek 3D Engine ist eine aus sechs, in sich geschlossenen Szenen bestehende, selbstlaufende und nicht interaktive Demonstration entstanden, welche einfache und höhere Techniken der 3D Computergrafik aggregiert. Die folgende Ausarbeitung beschreibt grundlegende Konzepte und einige der verwendeten Techniken dieser Demonstration.

## 1 Einleitung

Theory Exploration ist eine in Java entwickelte, aus sechs in sich geschlossenen Szenen bestehende, selbstlaufende und nicht interaktive Computergrafik-Demonstration. Als Bibliothek macht sie sich OpenGL in Form der JOGL-Bindings zu nutze, mit denen der volle OpenGL-Sprachumfang verfügbar gemacht wird. Die Szenen folgen einem festen sequenziellen Ablauf und werden durch Transitions-Effekte ineinander übergeblendet. Alle Szenen und Effekte der Demonstration laufen synchronisiert zu einem Musikstück ab, welches Klaus Spang[1] zu diesem Zweck verfügbar gemacht hat. Primäres Ziel war es, mehrere Techniken aus der 3D Computergrafik aggregiert zu mehreren Szenen in einer einzigen Applikation zu integrieren und durch den gezielten Einsatz von Transitions-Effekten zu erreichen, dass diese nicht mehr isoliert, sondern vielmehr als ein Ganzes wahrgenommen werden. Ein Schwerpunkt neben der rein technischen Umsetzung war es also auch, ein ästhetisches Zusammenspiel der einzelnen Elemente zu erreichen.

## 2 Starten der Applikation

Die Hauptklasse ist in der Datei `TheoryExploration.java` in dem Package `photek.demos.exploration` anzutreffen. Aufgrund des hohen Speicherbedarfs muss die Applikation mit der Option `-Xmx256M` der virtuellen Maschine gestartet werden.

## 3 Verwendete Techniken

Im Folgenden sollen einige der grundlegenden und höheren Techniken, die in Theory Exploration zur Anwendung kommen, diskutiert werden.

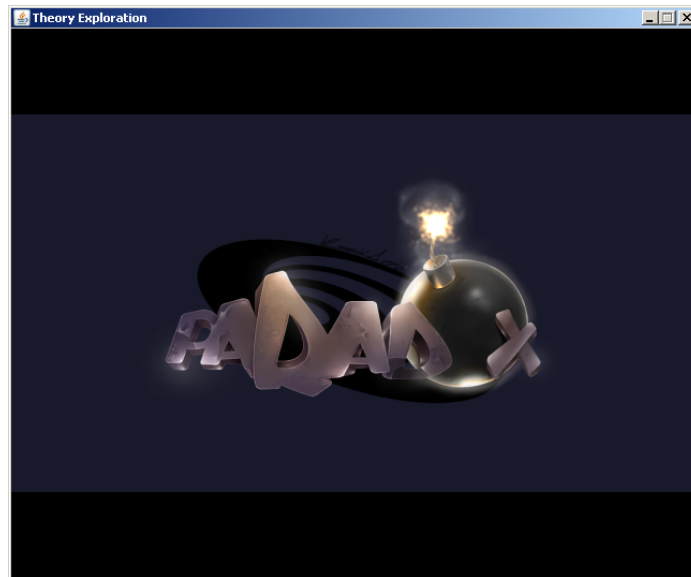
### 3.1 Animation

Die Animation ist ein wichtiges Thema in der Computergrafik. Eine häufige Bedingung, die an Animationen gestellt wird, ist, dass diese auf jedem Rechner in gleicher Geschwindigkeit ablaufen. Zu diesem Zweck führt Theory Exploration ein virtuelles Zeitsystem ein. Formal betrachtet ist eine Animation eine Abbildung (bzw. Funktion), die jedem Zeitpunkt ein Frame der Animation zuordnet. Wenn *Time* die Menge aller Zeitpunkte bezeichnet, so kann eine Animation wie folgt beschrieben werden:

$$f : TIME \longrightarrow FRAME \quad (1)$$

### 3.2 Billboarding und Partikelsysteme

Das Arbeiten mit geometrischen Primitiven ist aus der Sicht der GPU eine kostspielige Operation. Da schon relativ einfache 3D-Modelle mehrere tausend Primitiven besitzen können, ist es in einigen Fällen sinnvoll, andere Methoden zur Visualisierung zu verwenden. Eine beliebte Technik ist das Ersetzen des 3D-Modells durch ein texturiertes Rechteck, dessen Textur ein zweidimensionales Abbild des 3D-Modells darstellt. Auf diese Weise können Modelle aus mehreren tausend Primitiven mit nur zwei texturierten Dreiecken ersetzt werden. Auch Theory Exploration nutzt diese als Billboarding bekannte Technik zur Darstellung von Partikeln.[2] Abbildung 1 zeigt den Einsatz eines Partikelsystems in Theory Exploration.



**Abb. 1.** Partikelsystem in Theory Exploration

### 3.3 Video-Texturen

Die Photek 3D Engine bietet mit Video-Texturen die Möglichkeit, kleinere Filmsequenzen und Video-Clips in OpenGL-Texturen zu rendern, um mit diesen anschließend Geometrie zu texturieren. In Theory Exploration wird dies genutzt, um eine kleine Film-Szene abzuspielen. Abbildung 2 zeigt die Film-Szene aus Theory Exploration.



Abb. 2. Video-Texturen im Einsatz

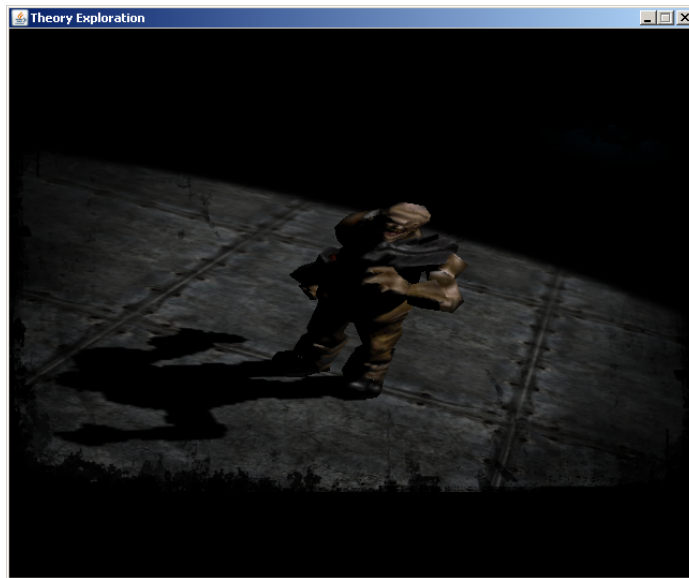
### 3.4 Shadow-Depth-Maps

Theory Exploration nutzt für die Darstellung von Schatten-Effekten Shadow-Depth-Maps. Bei den Shadow-Depth-Maps wird der Z-Buffer und das projektive Texture-Mapping verwendet um Schatten zu rendern. Bei dieser Technik wird die Szene so transformiert, dass die Kamera in die gleiche Position wie das entsprechende Spotlight verschoben wird, welches für den Schattenwurf verantwortlich ist. Aus dieser Einstellung wird daraufhin die Szene aus der Sicht der Lichtquelle gerendert. Die so erhaltenen Tiefeninformationen werden anschließend in eine Depth-Map kopiert. Diese Textur wird dann beim Rendering des fertigen Bildes wiederverwendet, indem sie mit der OpenGL-Textur-Koordinaten-Generierung so auf die Szene projiziert wird, dass die Texturkoordinaten eines Punktes den Vertexkoordinaten des Punktes bezüglich der Lichtquellen entsprechen. Anschaulich gesprochen, wird also die Depth-Map aus der Sicht der Lichtquelle auf die Szene gelegt. Mit der entsprechenden OpenGL-Extension kann nun

der Abstand eines jeden zu zeichnenden Fragments mit der Tiefeninformation in der Depth-Map verglichen werden. Ist der tatsächliche Abstand  $r$  des Fragments größer als der gespeicherte Wert des Texels in der Depth-Map, so liegt der betrachtete Pixel hinter einer Oberfläche, die von der Lichtquelle aus sichtbar ist. Dies bedeutet, dass er im Schatten liegt.[3]

Ein Vorteil dieses Verfahrens ist, dass der Aufwand nicht von der Komplexität der Szene abhängt wie es z.B. bei dem Shadow-Volume-Verfahren der Fall ist. Dieses benötigt nämlich zusätzlichen Rechen- und Speicheraufwand für Adjazenz-Informationen der Geometrie und ist somit unmittelbar von der Komplexität der Szene abhängig.[4]

Abbildung 3 zeigt den Einsatz von Shadow-Depth-Maps in Theory Exploration.

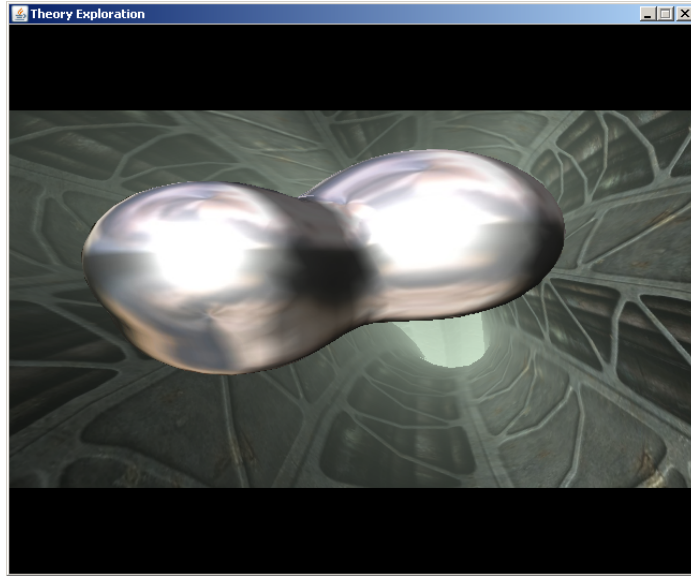


**Abb. 3.** Shadow-Depth-Maps in Theory Exploration

### 3.5 Surface Reconstruction

Mit dem „Marching Cubes“ Algorithmus lassen sich polygonale Repräsentationen eines Iso-Surfaces erzeugen. Dabei wird genau genommen nur eine der Äquipotentialflächen eines diskreten dreidimensionalen Skalarfeldes polygonisiert.[5] In dem Artikel „Polygonising a scalar field“ beschreibt Paul Bourke detailliert die Funktionsweise des „Marching Cubes“ Algorithmus.[6]

In Theory Exploration werden mit dem „Marching Cubes“ Algorithmus sogenannte Metaballs[7] gerendert. In Abbildung 4 sind die Metaballs aus einer Szenen der Demonstration zu sehen.



**Abb. 4.** Polygonisiertes Iso-Surface mehrerer Metaballs

Das Iso-Surface eines einzelnen Metaballs, welches durch den Marching Cubes Algorithmus polygonisiert werden soll, ist durch die folgende Funktion gegeben:

$$F(x, y, z) = \frac{r^2}{(x-a)^2 + (y-b)^2 + (z-c)^2} \quad (2)$$

Wobei  $r$  den Radius darstellt, wenn von einer mit  $F(x, y, z) = 1$  definierten Äquipotentialfläche ausgegangen wird. Mit dem Marching Cubes Algorithmus lässt sich nun eine polygonale Repräsentation dieser Funktion erzeugen. Benötigt werden nun noch die orthogonal zu dieser Äquipotentialfläche stehende Normalen-Vektoren. Diese sind aber genau durch den Gradienten der verwendeten Funktion des Iso-Surface bestimmt. Da der Gradient die Richtung des größten Anstiegs darstellt zeigen die durch ihn bestimmten Normalen-Vektoren jedoch noch in das Innere der Äquipotentialfläche, so dass dies durch Negation des Gradienten korrigiert werden muß:

$$N(x, y, z) = -\nabla F(x, y, z) \quad (3)$$

$$= -\left(\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}, \frac{\partial F}{\partial z}\right) \quad (4)$$

$$= -\left(\frac{-2*r^2*(x-a)}{((x-a)^2+(y-b)^3+(z-c)^2)^2}, \frac{-2*r^2*(y-b)}{((x-a)^2+(y-b)^3+(z-c)^2)^2}, \frac{-2*r^2*(z-c)}{((x-a)^2+(y-b)^3+(z-c)^2)^2}\right) \quad (5)$$

## Erklärung

Ich versichere, die vorliegende Ausarbeitung selbstständig und nur unter Benutzung der angegebenen Hilfsmittel angefertigt zu haben.

## Literatur

1. Klaus Spang: keith303 audio resources. <http://www.keith303.de/> (2007)
2. António Ramires Fernandes: Billboarding Tutorial. <http://www.lighthouse3d.com/opengl/billboarding/> (2001)
3. Cass Everitt: Shadow Mapping. [http://developer.nvidia.com/object/shadow\\_mapping.html](http://developer.nvidia.com/object/shadow_mapping.html) (2001)
4. Carsten Dachsbacher: Shadow Depth Maps mit OpenGL. <http://www-sop.inria.fr/reves/Carsten.Dachsbacher> (2002)
5. Wikipedia: Marching Cubes. [http://en.wikipedia.org/wiki/Marching\\_cubes](http://en.wikipedia.org/wiki/Marching_cubes) (2007)
6. Paul Bourke: Polygonising a scalar field. <http://local.wasp.uwa.edu.au/~pbourke/geometry/polygonise/> (1994)
7. Andreas Jönsson: Fast metaballs. <http://www.angelcode.com/dev/metaballs/metaballs.asp> (2001)