NICHT-PHOTOREALISTISCHES RENDERING MIT PROGRAMMIERBARER GRAFIK-HARDWARE

Michael Horsch
Fachbereich Informatik
Hochschule Darmstadt

ABSTRACT
Diese Arbeit zeigt Techniken zum nicht-photorealistischen Rendering mit programmierbarer Grafik-Hardware.

KEY WORDS
NPR, GPU, Cel Shading, Outlines, Edge Detection

1 Einleitung

Nicht-Photorealistisches Rendering ist ein Teilgebiet der Computer Graphik bei dem es hauptsächlich darum geht, mit dem Computer generierte Bilder wie Zeichnungen aussehen zu lassen.
Obwohl das normale photorealistische Rendering viel mehr Anwendungsgebiete hat, gibt es auch Bereiche in den nicht-photorealistisches Rendering benötigt wird. So werden zum Beispiel große Teile von Zeichentrickfilmen mit der Unterstützung von Computern produziert. Damit die mit dem Computer generierten Bilder auch wie Zeichnungen aussehen ist der Einsatz verschiedener Techniken zum nicht-photorealistischen Rendering nötig. Die Berechnungen der einzelnen Techniken können aber sehr Zeitaufwändig sein.
Mit Hilfe moderner programmierbarer Grafik-Hardware können die Berechnungen deutlich beschleunigt werden. Dadurch ist es möglich, dass Anwendungen die nicht-photorealistisches Rendering verwenden bei interaktiven Frameraten ausgeführt werden. So kann die Entwicklungszeit von Produkten mit nicht-photorealistischem Rendering bei geringen Kosten (für die Grafik-Hardware) deutlich beschleunigt werden. Lange Rechenzeiten und kleine Vorschau-Bildchen gehören somit der Vergangenheit an, die Ergebnisse sind sofort sichtbar und werden bei interaktiven Frameraten angezeigt.

1.1 Analyse von Zeichentrickfilmen

Im Gegensatz zu Photographien suggerieren Zeichentrickfilmen nur eine grobe Repräsentation der Szenen. Da es sich um Zeichnungen handelt, gelten die Prinzipien des Malens von Umriss und Schattierung.
Die Schattierung der Szene ist meistens nur sehr einfach gehalten. Es gibt keine weichen Farbverläufe sondern nur ein paar Abstufungen pro Farbe, die die Schattierung andeutungsweise wiedergeben.
Die Umrisse der einzelnen Objekte können in verschiedenen Stilen gemalt sein. Meistens haben Flächen mit der gleichen Farbe einen Umriss. Die Farben des Umrisse sind meistens dunkel oder schwarz.

2 Grundlagen

Im Bereich des nicht-photorealistischen Rendering wurde schon viel Forschung betrieben. Die meisten Techniken benutzen Variationen und Kombinationen existierenden Lösungen um Zeichnungen zu imitieren. Um die Bilder wie Zeichnungen aussehen zu lassen, werden Techniken zur Berechnung der Schattierung, der Schatten und der Umrisse benötigt.

2.1 Cel Shading

Cel Shading ist eine Technik, bei der die Schattierungen neu quantisiert wird. Das heißt, die Anzahl der Farbabstufungen wird reduziert.
Anstatt weiche Schattierungen zu benutzen, wird die Schattierung in Intervalle eingeteilt. So werden Abstufungen in der Schattierung sichtbar. Das wiederum erzeugt eine Schattierung wie sie in Zeichentrickfilmen häufig verwendet wird.
Abbildung 1 zeigt den Unterschied zwischen einer weichen Schattierung und der Schattierung mit Cel Shading.

2.2 Umrisse

Das Zeichnen der Umrisse kann auf verschiedene Weisen realisiert werden. Die Techniken hierzu können in zwei Gruppen eingeteilt werden. Die Techniken der einen Gruppe benutzen die Informationen der verwendeten 3D-Modelle um die Umrisse zu zeichnen, die andere Gruppe umfasst die Techniken die im Screen Space ausgeführt werden.
Die Techniken die die Informationen über die verwendeten 3D-Modelle benutzen, berechnen die Kanten des 3D-Modells, die zur Silhouette gehören. Eine Kante gehört zur Silhouette wenn die Kante zu einem Polygon gehört das zum Betrachter zeigt und gleichzeitig zu einem Polygon gehört das vom Betrachter weg zeigt. Die erkannten Kanten werden dann entlang der Vertex-Normalen extrudiert. Mit

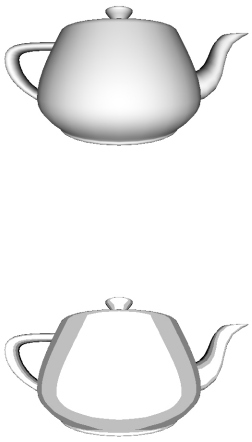

Abbildung 1. Oben: weiche Schattierung. Unten:Cel Shading

den erkannten Kanten und den extrudierten Kanten liegen genügend Informationen vor um einen Umriss zu zeichnen. Screen Space-Techniken zeichnen zuerst die Objekte in eine Textur. Auf diese Textur wird dann ein Kanten-Detektor angewendet. Damit der Kanten-Detektor die Umrisse richtig erkennt, können zum Beispiel alle Objekte die zu einem Umriss gehören in der gleichen Farbe in die Textur gezeichnet werden.

2.3 Shadow Mapping

Shadow Mapping [3] ist ein Textur-basierter Ansatz zur Schattenberechnung, der eigentlich aus dem Bereich des realistischen Rendering kommt. Die Idee bei Shadow Mapping ist, dass das was das Licht nicht sieht im Schatten liegt. Dazu wird die Szene von der Position der Lichtquelle aus gezeichnet und die Werte des Tiefenpuffers werden in einer Textur gespeichert, der sogenannten Shadow Map. Diese Textur wird dann wieder vom Licht aus auf die Szene projeziert. Die in der Textur gespeicherten Tiefenwerte des Lichts werden mit den Tiefenwerten des Lichts verglichen die der Betrachter sieht. Ist der Tiefenwert in der Textur kleiner, befindet sich der Punkt im Schatten.

3 Nicht-Photorealistisches Rendering mit programmierbarer Grafik-Hardware

Um die Berechnungen für die Verarbeitung mit programmierbarer Grafik-Hardware zu Beschleunigen müssen die Algorithmen entsprechend angepasst werden. Die folgenden Beispiele zeigen wie die einzelnen Techniken mit Hilfe programmierbarer Grafik-Hardware realisiert werden können.

3.1 Cel Shading GPU Implementierung

Die Realisierung von Cel Shading auf programmierbarer Grafik-Hardware unterscheidet sich kaum von der Implementierung auf der CPU. Zuerst wird das Skalarprodukt zwischen dem Normalen Vektor und der Richtung zu der Lichtquelle gebildet. Dadurch erhält die Szene eine weiche Schattierung, wie bei der diffusen Beleuchtung mit Phong-Shading[2]. Für das Cel Shading werden dann die Grauabstufungen der Farben durch eine neue Einteilung stark verringert.

Listing 1. Cel-Shading Beispiel-Code.

```
//weiche Schattierung berechnen
float shading = dot(normal, lightDir);

//weiche Schattierung neu quantisieren
float celShading=1.0;

celShading =
shading < 0.5 ? 0.75 : celShading;

celShading =
shading < 0.25 ? 0.5 : celShading;

celShading =
shading < 0.1 ? 0.25 : celShading;

vec4 finalColor = Color*celShading;
```

Listing 1 zeigt wie Cel Shading mit programmierbarer Grafik-Hardware realisiert werden kann. Je nach gewünschtem Detailgrad können diese Berechnungen entweder auf Vertex-Ebene oder auf Pixel-Ebene durchgeführt werden. Die Anzahl der Farbabstufungen kann frei variiert werden.

3.2 Umrisse GPU Implementierung

Für das Zeichnen der Umrisse mit Hilfe programmierbarer Grafik-Hardware bieten sich vor allem die Screen-Space-Techniken an, die wie Post-Processing Effekte funktionieren.
Im Gegensatz zu anderen Techniken werden keine Informationen über die verwendeten 3D-Modelle benötigt. Es muss nur festgelegt werden, welche 3D-Modelle zum gleichen Umriss gehören. Damit man mit einem Kanten-Detektor die Umrisse gut herausfiltern kann, werden hierfür die Farben der Objekte genutzt. Das heißt, die Objekte werden zuerst mit ihren Farben in eine Textur gezeichnet. Auf dieser Textur wird dann mit Hilfe programmierbarer Grafik-Hardware ein Kanten-Detektor

ausgeführt. Die erkannten Kanten werden zum Schluß wieder mit der restlichen Szene kombiniert.

Listing 2. Beispiel-Code zur Kantenerkennung.

```
float fSize=1.0/textureSize;
vec4 sample[9];
vec2 offset[9];

offset[0]=TexCoord;

//Textur-Koordinaten für die
//benachbarten Pixel
offset[1]=TexCoord+vec2(fSize);
offset[2]=TexCoord+vec2(-fSize);
offset[3]=TexCoord+vec2(fSize,0.0);
offset[4]=TexCoord+vec2(-fSize,0.0);
offset[5]=TexCoord+vec2(0.0,fSize);
offset[6]=TexCoord+vec2(0.0,-fSize);
offset[7]=TexCoord+vec2(fSize,-fSize);
offset[8]=TexCoord+vec2(-fSize,fSize);

vec4 edge=vec4(0.0);

//filter anwenden
int i;
for(i=1; i<9;i++)
{
edge+=texture2D(tex, offset[i]);
}
edge+=texture2D(tex, offset[0])*(-8.0);

//kanten invertieren und ausgeben
finalColor = vec4(1.0)-edge;
```

Listing 2 zeigt ein Beispiel zur Kantenerkennung mit programmierbarer Grafik-Hardware. Als Kantendetektor wurde ein Laplace-Filter[1] benutzt.

3.3 Verzerrung des Bildes

Um dem Bild ein natürlicheres Aussehen zu geben, wird das Bild nach dem Cel Shading und der Kantenerkennung leicht verzerrt. Dadurch kann das Zittern der Hand beim Zeichnen simuliert werden. Um das Bild verzerren zu können, muss es zuerst wieder in eine Textur gezeichnet werden. Die Textur-Koordinaten müssen dann für den Verzerr-Effekt an jedem Pixel leicht verschoben werden. Eine Perlin Noise Textur kann verwendet werden um Richtung und Betrag der Verschiebung festzulegen.

Listing 3. Verzerrung des Bildes.

```
//noiseAmp bestimmt wie stark
//das Bild verzerrt wird
float noiseAmp=0.01;

vec2 noise;
noise=texture2D(noiseTex, TexCoords);
noise=normalize(noise*2.0-vec2(1.0));
noise*=noiseAmp;

//Verzerrung des Bildes durch
//Verzerrung der Textur-Koordinaten
vec4 finalColor=
texture2D(nprTex, TexCoords+noise);
```

4 Anwendungsbeispiele

Mit den vorgestellten Techniken können nicht-photorealistische Bilder mit programmierbarer Grafik-Hardware implementiert werden. Dies kann in vielen verschiedenen Variationen geschehen. Drei Varianten zum nicht-photorealistischen Rendering wurden mit den Techniken realisiert.

4.1 Cartoon Rendering

Durch die einfache Kombination der Techniken erhalten die erzeugten Bilder ein zeichentrickhaftes Aussehen (siehe Abb. 2). Für das Cel-Shading wurden drei Abstufungen in der Schattierung verwendet. Zusätzlich wurden noch Schatten mit dem Shadow Mapping Algorithmus hinzugefügt. Bei diesem Algorithmus können zwar Artefakte auftreten, diese passen aber zu dem „unordentlichen“ Stil der Zeichnung.

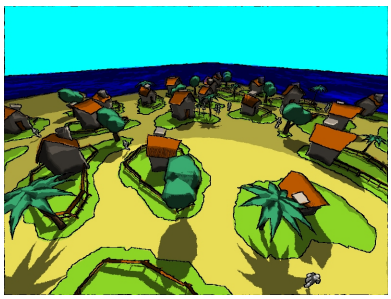

Abbildung 2. Cartoon Rendering mit programmierbarer Grafik-Hardware.

4.2 Bleistift und Tinte

Lässt man die Farben weg und zeichnet nur die Umrisse, entsteht der Eindruck einer groben Skizze (siehe Abb. 3). Damit die Umrisse so aussehen als hätte man sie mehrfach nachgezeichnet, werden mehrere Umrisse mit unterschiedlicher Verzerrung gezeichnet und überlagert. Um die unterschiedlichen Größen von Stiften zu simulieren kann die Größe des Kantenerkennungs-Filter verändert werden.

Abbildung 3 zeigt als Beispiel-Anwendung eine Zeichnung mit Bleistiften und eine Zeichnung mit Tinte.

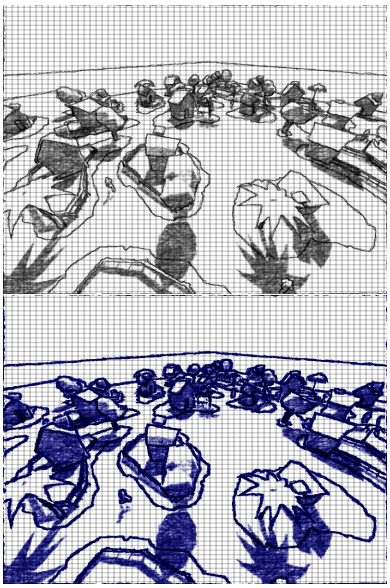

Abbildung 3. Oben: Bleistift-Zeichnung. Unten: Zeichnung mit Tinte.

5 Ergebnisse

Nicht-Photorealistisches Rendering auf programmierbarer Grafik-Hardware hat viele Vorteile gegenüber der Ausführung der benötigten Rechenoperationen mit der CPU oder nicht-programmierbarer Grafik-Hardware.
Der größte Vorteil ist natürlich, dass programmierbare Grafik-Hardware speziell für solche Berechnungen entworfen wurde. Somit ist die Ausführungsgeschwindigkeit deutlich höher als auf einer CPU oder nicht-programmierbarer Hardware.
Durch die kürzeren Rechenzeiten verkürzen sich auch die Produktionszeiten. Bilder, deren Berechnung minutenlang dauerte, können durch die Nutzung der Grafik-Hardware sofort angezeigt werden.
Ein weiterer Vorteil ist, dass nicht-photorealistische Bilder bei interaktiven Frameraten angezeigt werden können. So kann die Szene beliebig verändert werden, die Ergebnisse sind gleich sichtbar. Die Berechnung von Vorschaubiler ist nicht mehr nötig.

6 Literaturverzeichnis

1 Wikipedia-Online-Enzyklopädie: Laplace-Filter, http://de.wikipedia.org/wiki/Laplace-Filter, zuletzt besucht am: 08.05.07

2 Bui-Tuong Phong: Illumination for Computer Generated Pictures, Juni 1975

3 Lance Williams: Casting Curved Shadows on Curved Surfaces, Computer Graphics Vol.12 No.3 (SIGGRAPH'78 Proccedings): pp. 270-274, Atlanta(Georgia), 1978