

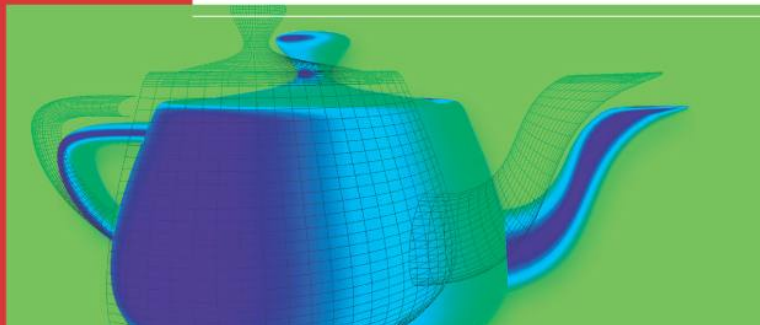
Michael Bender
Manfred Brill

Computer- grafik

Ein anwendungsorientiertes Lehrbuch

HANSER

2. Auflage



Aufgaben und Lösungen

©Michael Bender, Manfred Brill
Oktober 2005

Sie finden in diesem Dokument alle Aufgaben und die zugehörigen Lösungen aus

Michael Bender, Manfred Brill: *Computergrafik*

2. Auflage, Hanser Verlag, München, 2006



<http://www.vislab.de/cgbuch>

Diese Unterlagen wurden mit L^AT_EX erstellt.

Kapitel 5

Bildsynthese

5.7 (Anti-)Aliasing

1. Zum Rendern ohne Supersampling eines Bilds der Auflösung 768×576 benötigen Sie 20 Minuten. Sie entscheiden sich, das Bild produktionsreif zu berechnen, und wählen ein vierfaches Supersampling aus. Mit welcher Renderzeit müssen Sie jetzt für das Bild rechnen? Mit welcher Renderzeit müssen Sie rechnen, wenn Sie einen entsprechenden Film der Länge 5 Minuten rendern möchten (nehmen Sie eine zu berechnende Bildwiederholrate von 50 Hz und eine gleichaufwändige Bildberechnung an)?

Lösung:

Einzelbild:

20 Minuten $\cdot 4 \cdot 4 = 320$ Minuten = 5 Stunden 20 Minuten

5 Minuten Sequenz:

$$\begin{aligned} 320 \text{ Minuten} \cdot 5 \cdot 60 \cdot 50 &= 4.800.000 \text{ Minuten} \\ &= 80.000 \text{ Stunden} \\ &= 3.333 \text{ Tage } 8 \text{ Stunden} \\ &= 9 \text{ Jahre } 48 \text{ Tage } 8 \text{ Stunden} \end{aligned}$$

2. Betrachten Sie das obige Beispiel des sich drehenden Wagenrads. Was passiert, wenn wir das Rad genauso schnell drehen, dass eine achte Umdrehung 0.02 s dauert? Warum ist eine variierende Geschwindigkeit des Rads in einem Bereich um diese Geschwindigkeit besonders störend?

Lösung:

Bei dieser Geschwindigkeit des Rads scheint es still zu stehen. Ändert sich allerdings die Geschwindigkeit leicht zu- oder abnehmend, sehen wir das Rad abwechselnd vorwärts und rückwärts drehen.

3. Betrachten Sie das obige Beispiel des sich drehenden Wagenrads einer Kutsche. Die Wagenräder der Kutsche haben einen Durchmesser von 1.10 m. Bis zu welcher Geschwindigkeit darf die Kutsche gerade nicht mehr beschleunigen, damit keine Aliasing-Effekte bei der Wahrnehmung der Wagenräder auftreten?

Lösung:

Eine volle Periode (eine achte Umdrehung) muss mehr als 0.04 s dauern. Also muss eine

volle Umdrehung mehr als 0.32 s dauern. Eine volle Umdrehung entspricht dem zurückgelegten Weg von $2 \cdot \pi \cdot 1.1/2 \text{ m} = 3.46 \text{ m}$. Damit ist die maximale Geschwindigkeit kleiner als $10.8 \text{ m/s} = 38.9 \text{ km/h}$.

5.9 Fallstudien

5.9.1 Beleuchtung, Materialien und Schattierungen in OpenGL

1. Schreiben Sie ein kurzes OpenGL-Programm, das als Objekt eine einfache Kugel darstellt. Beleuchten Sie diese abwechselnd mit einem Richtungslicht und einer Punktlichtquelle. Verwenden Sie einmal ein in der Szene fest platziertes Licht und im Gegensatz dazu ein Headlight!

Lösung:

Ein entsprechendes Beispielprogramm finden Sie unter *KugelBeleuchtung* in den Quellcodes zu diesem Kapitel.

2. Definieren Sie sich per Hand mit `GL_TRIANGLES` Primitiven einen goldfarbenen Würfel. Entfernen Sie eine der Seitenflächen und experimentieren Sie mit `glFrontFace`, `glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, *)` und Backface-Culling!

Lösung:

Ein entsprechendes Beispielprogramm finden Sie unter *GoldenerWürfel* in den Quellcodes zu diesem Kapitel.

Warum erkennt man bei eingeschalteter Beleuchtung und aktiviertem Flat-Shading die diagonale Kante auf einer Würfelseite?

3. Definieren Sie per Hand mit `GL_QUADS` Primitiven einen Würfel mit den sechs unterschiedlichen Materialien schwarzes Plastik, Messing, Bronze, Kupfer, Gold und Silber!

Lösung:

Ein entsprechendes Beispielprogramm finden Sie unter *MaterialienWürfel* in den Quellcodes zu diesem Kapitel.

4. Benutzen Sie die GLUT-Funktion `glutSolidSphere` zur Darstellung einer ausgefüllten Kugel. Diese Funktion setzt u. a. die Eckennormalen automatisch. Definieren Sie ein Material für die Kugel und experimentieren Sie mit Flat- und Gouraud-Shading in OpenGL! Polygonalisieren Sie nun selbst eine Kugel mit variabler Unterteilung in Richtung des Längen- und Breitengrads. Bestimmen Sie einmal Ihre Eckennormalen optimal für Flat-Shading und einmal optimal für Gouraud-Shading. Vergleichen Sie Ihre Ergebnisse!

Lösung:

Ein entsprechendes Beispielprogramm finden Sie unter *KugelExperiment* in den Quellcodes zu diesem Kapitel.

Offensichtlich sieht die von GLUT generierte Kugel sowohl bei Gouraud- als auch bei Flat-Shading „gut“ aus.

Wir polygonalisieren nun selbst eine Kugel, indem wir Sie von oben nach unten in Streifen/Lagen einteilen und jeden dieser Streifen mit Quads approximieren. Die Anzahl der Lagen bezeichnen wir mit `STACKS`, die der Quads pro Lage mit `SLICES`. Unter Zuhilfenahme von Kugelkoordinaten sollte Ihnen die Berechnung der einzelnen Eckpunkte gelingen. Wenn wir nun mittels der Eckpunkte Quads erzeugen, müssen wir uns über die Normalen, die wir dabei setzen müssen, im Klaren sein.

Für Gouraud-Shading setzen wir im Eckpunkt das Mittel der angrenzenden Facettennormalen. Dies ist für unsere benutzte Einheitskugel um den Ursprung genau identisch der Eckpunktcoordinate – wir müssen nichts ausrechnen. Beachten Sie, dass es zu jedem Eckpunkt (egal wie oft er physikalisch existiert) nur „eine“ Normale in eine Richtung gibt. Beachten Sie im Beispielprogramm die Darstellung mit dargestellten Normalenvektoren. Diese Modell sieht bei Gouraud- und bei Flat-Shading gut aus und gleicht dem GLUT-Modell sehr. Eine Schwierigkeit ist vorhanden: Bei Flat-Shading scheint die eine Polkappe flach zu sein? Woran liegt das? Verbessern Sie das Beispielprogramm!

Beim Flat-Modell wählen wir jeweils für jeden Quad in allen seinen Eckpunkten die Eckenormale gleich der Facettennormale. Damit kann eine Ecke „mehrere“ Normalen haben! Sie benötigen dazu das Kreuzprodukt. Vergessen Sie nicht, die Normalen zu normieren! Was passiert eigentlich bei aktiviertem Gouraud-Shading? Das Objekt bleibt anscheinend Flat beleuchtet. Warum? Aber halt – ändert sich da nicht die Farbe um eine Nuance? Können Sie sich das erklären?

Es ist übrigens nicht relevant, dass für das Gouraud-Modell Quad-Strips verwendet wurden – hier könnten ebenso einzelne Quads verwendet werden, es würde nur ineffizienter.

Programmieren Sie das gesamte Beispielprogramm nach! Dies ist eine gute Übung und Sie werden staunen, über wieviele Fallstricke (auf diese ist in unserem Quelltext hingewiesen) Sie stolpern, obwohl das Prinzip klar ist.

5.9.2 Cg Vertex-Shader in OpenGL

1. Erweitern Sie unseren Phong-Shader um eine zweite Lichtquelle!

Lösung:

Ein entsprechendes Beispielprogramm finden Sie unter *CgTwoLights* in den Quellcodes zu diesem Kapitel.

2. Das umgesetzte Phong-Modell ist nicht exakt identisch mit dem in Abschnitt 5.3.2 eingeführten Phong-Modell. Beheben Sie dies!

Lösung:

Wir haben im vorgeführten Shader-Beispiel von vielen Einzelheiten abstrahiert. Insbesondere hat eine Lichtquelle verschiedene Anteile (ambient, diffus, spiegelnd). Diese lassen sich auch in OpenGL als Parameter einer Lichtquelle einstellen. Auch die Materialparameter wurden einfach gewählt und fest im Shader verankert. Auch diese stellt man in OpenGL über Materialparameter ein.

Ihre Aufgabe ist nun, alle OpenGL-Parameter über geeignete Parameterübergaben in den Shader weiter zu reichen (Beispiele zur Parameterübergabe birgt das gegebene Beispiel) und die Phong-Beleuchtungsformel detaillierter umzusetzen.

3. Implementieren Sie die Variante des Phong-Modells, die den Reflexionsvektor statt des Halfway-Vektors benutzt! Vergleichen Sie die Ergebnisse!

Lösung:

Ein entsprechendes Beispielprogramm finden Sie unter *CgPhongBeleuchtungR* in den Quellcodes zu diesem Kapitel.

4. Fügen Sie dem Cartoon-Shading Farbe hinzu!

Lösung:

Ein entsprechendes Beispielprogramm finden Sie unter *CgColorCartoon* in den Quellcodes zu diesem Kapitel.

5.9.3 GLSL Vertex- und Fragment-Shader

1. Implementieren Sie das Phong-Shading für die Original-Phong-Beleuchtung mit dem Reflexionsvektor \mathbf{r} ! Vergleichen Sie!

Lösung:

Ein entsprechendes Beispielprogramm finden Sie unter *PhongShadingR* in den Quellcodes zu diesem Kapitel.

2. Implementieren Sie die Phong-Beleuchtung mittels Per Vertex-Lighting!

Lösung:

Ein entsprechendes Beispielprogramm finden Sie unter *PhongBeleuchtungR* in den Quellcodes zu diesem Kapitel.

3. Implementieren Sie die Blinn-Beleuchtung mittels Per Vertex-Lighting!

Lösung:

Ein entsprechendes Beispielprogramm finden Sie unter *PhongBeleuchtungH* in den Quellcodes zu diesem Kapitel.

4. Implementieren Sie ein farbiges Cartoon-Shading mittels Per Pixel-Lighting!

Lösung:

Ein entsprechendes Beispielprogramm finden Sie unter *GLSLColorCartoon* in den Quellcodes zu diesem Kapitel.

5.9.4 Mapping in OpenGL

1. Benutzen Sie eine beliebige Textur und konstruieren Sie per Hand mit Primitiven und den zugehörigen Textur-Koordinaten die linken Bilder aus Abbildung 5.32 nach!

Lösung:

Ein entsprechendes Beispielprogramm finden Sie unter *AbbildungTexturing* in den Quellcodes zu diesem Kapitel.

Die Schwierigkeit ist das nicht planare Viereck - korrekter das bilineare Flächenstück! Dies können wir mittels eines Bézier-Flächensegments vom Grad (1,1) darstellen und dann wie in der letzten Aufgabe dieses Kapitels mit uv-Mapping texturieren. Denken Sie insbesondere daran, dass OpenGL überhaupt nicht definiert was passiert, falls ein Primitiv nicht planar oder nicht konvex ist – es kann etwas beliebig falsches rauskommen.

2. Konstruieren Sie per Hand einen Spiel-Würfel mit entsprechenden Texturen!

Lösung:

Ein entsprechendes Beispielprogramm finden Sie unter *Spielwürfel* in den Quellcodes zu diesem Kapitel.

3. Erzeugen Sie mit einem prozeduralen Ansatz eine Schachbrett-Textur und texturieren Sie damit den Teapot!

Lösung:

Ein entsprechendes Beispielprogramm finden Sie unter *SchachbrettTeapot* in den Quellcodes zu diesem Kapitel.

4. Erzeugen Sie mit Hilfe eines prozeduralen Ansatzes eine „Yps“-Textur auf einem Teapot!

Lösung:

Ein entsprechendes Beispielprogramm finden Sie unter *YpsTeapot* in den Quellcodes zu diesem Kapitel.

5. Texturieren Sie ein Bézier-Flächensegment vom Grad $(3, 3)$ unter Ausnutzung des *uv*-Mappings! Lesen Sie dazu in [WND00] nach, wie die Evaluatoren für die Berechnung der Textur-Koordinaten benutzt werden können!

Lösung:

Idee: Die zu generierenden Textur-Koordinaten sind die Ausgaben der Auswertung eines zweiten (planaren) Bézier-Flächensegments über dem Einheitsquadrat. Wir wählen hier den Polynomgrad 1, was die wegen der linearen Präzession entstehenden Textur-Koordinaten linear ohne Verzerrung auf dem Parametergebiet $[0,1]$ entstehen lässt. Ein entsprechendes Beispielprogramm finden Sie unter *BezierTexturing* in den Quellcodes zu diesem Kapitel.

Interessant wäre auch ein höherer Polynomgrad auf dem Einheitsquadrat mit unregelmäßig verteilten inneren Kontrollpunkten; damit lässt sich die Verformung der Textur modellieren. Probieren Sie dies aus!