BERGISCHE UNIVERSITÄT WUPPERTAL
FAKULTÄT FÜR MATHEMATIK
UND NATURWISSENSCHAFTEN

BERGISCHE
UNIVERSITÄT
WUPPERTAL

Angewandte Informatik
Dr. Martin Galgon
M.Sc. Jose Jimenez

Bildgenerierung

Wintersemester 2023 / 2024

Übungsblatt 7

Aufgabe 18 *(Färbung – Beleuchtungsmodell nach Phong)*

Das Rahmenprogramm `proj4.cc` im Verzeichnis `/home/bildgen/Aufgaben/projektion-4` implementiert die perspektivische Projektion samt Füllung der Dreiecksflächen. Ergänzen Sie das Programm um eine pro Gitterpunkt berechnete Beleuchtung. Implementieren Sie hierzu in der Funktion

```
VecRGB berechneBeleuchtung(const Vec3D& ecke, const Vec3D& normale,
                           const Vec3D& auge, const Vec3D& licht,
                           const DrawColour& farbe)
```

das Beleuchtungsmodell nach Phong für *eine* Lichtquelle. Die Funktion wird für jeden der drei Eckpunkte einer Dreiecksfläche aufgerufen und soll die Farbe an diesem Punkt berechnen. Die Interpolation der Farbe über die Fläche (Gouraud-Shading) übernimmt das Rahmenprogramm für Sie.

Die folgenden Größen sind vorgegeben:

`Vec3D ecke`	Position der Ecke in Weltkoordinaten
`Vec3D normale`	Normale zur Fläche in dieser Ecke (in Weltkoordinaten)
`Vec3D auge`	Koordinaten des Auges in Weltkoordinaten
`Vec3D licht`	Koordinaten der Lichtquelle in Weltkoordinaten
`VecRGB lightAmbient`	ambiente Lichtintensität
`VecRGB lightDiffuse`	diffuse Lichtintensität
`VecRGB lightSpecular`	Lichtintensität für winkelabhängige Reflexion
`VecRGB materialAmbient`	ambienter Reflexionskoeffizient des Materials
`VecRGB materialDiffuse`	diffuser Reflexionskoeffizient des Materials
`VecRGB materialSpecular`	winkelabhängiger Reflexionskoeffizient des Materials
`double materialSpecularity`	Exponent für winkelabhängige Reflexion
`double c0, c1, c2`	Konstanten für entfernungsabhängige Dämpfung

Die Rechnung mit Farbwerten innerhalb der Funktion erfolgt über dreidimensionale Vektoren, deren Einträge die drei Kanäle des RGB-Farbmodells darstellen. Die RGB-Intensitäten sind auf $[0, 1]$ normiert. Für Lichter stellen diese Werte die Lichtintensität des jeweiligen Farbkanals dar, für Materialien bzw. Objektoberflächen deren Reflexionskoeffizienten für den jeweiligen Farbkanal.

Um nicht jeden Farbkanal einzeln berechnen zu müssen sind neben `+` und `-` geeignete Operatoren vorgegeben, z. B. `VecRGB v1 * VecRGB v2` für elementweise Multiplikation.

Beachten Sie, dass negative Skalarprodukte auf dem Auge bzw. der Lichtquelle abgewandte Flächen hindeuten. Im Falle eines negativen Skalarproduktes ist der entsprechende Lichtanteil auf Null zu setzen, um keine negativen Lichtintensitäten zu erzeugen. Hierzu können sie die `max()`-Funktion benutzen.

Testen Sie Ihr Programm mit den `*.in`-Dateien.

Aufgabe 19 *(Rundflug um den Eiffelturm)*

Erzeugen Sie eine Animation, die einen spiralförmigen Rundflug um den Eiffelturm zeigt. Fahren Sie hierzu nicht nach Paris, sondern verwenden Sie die Daten aus der Datei `toureiffel.in`. Starten Sie Ihren Flug am Boden ($y = -662$) und fliegen Sie bis zur Spitze ($y = 550$), wobei Sie den Turm einmal umrunden. Halten Sie einen konstanten Abstand von 800 Längeneinheiten von der Mitte des Turms (y-Achse) und blicken Sie immer waagerecht zur Mitte. Dann passen Ihre Aufnahmen auf Bilder der Größe 250×400 Pixel. Der Abstand zur Projektionsebene spielt hier keine große Rolle und kann recht willkürlich gewählt werden. Setzen Sie Ihren Film aus 100 Einzelbildern zusammen.

Verwenden Sie die Eingabedatei `toureiffel.in` und ignorieren Sie die Werte der ersten beiden Zeilen (COP und TGT). Erzeugen Sie aus Ihrem Flug eine animierte `gif`- oder `mpg`-Datei. Ein Beispielprogramm zur Erzeugung von Animationen finden Sie im Verzeichnis `/home/bildgen/Aufgaben/rundflug`. Zum Betrachten der Dateien können Sie `gwenview` verwenden, für `gif`-Ausgaben zusätzlich auch `eog`.

Falls Sie Aufgabe 18 nicht bearbeitet haben, lassen Sie die Beleuchtungsfunktion einfach die übergebene Farbe zurückgeben und setzen Sie den `outline` Parameter von `maleDreiecke` auf `true`.

Aufgabe 20 *(Perspektivische Projektion mit OpenGL)*

Im Verzeichnis `/home/bildgen/Aufgaben/opengl-2` finden Sie eine OpenGL-Implementierung der perspektivischen Projektion aus Aufgabe 13. Compilieren lässt sich das Programm mittels `make`. Weiterführende Informationen zu OpenGL finden Sie auf der Webseite
`http://www.opengl.org/sdk/docs/man/`.

OpenGL nutzt im Wesentlichen zwei Matrizen, um Objekte, welche in Objektkoordinaten gegeben sind, in normalisierte Koordinaten zu transformieren und anschließend zu clippen und zu zeichnen. Zum ersten handelt es sich dabei um die sogenannte ModelView-Matrix, welche die Objektkoordinaten in Augenkoordinaten überführt. Diese Matrix übernimmt also die Transformation des Augenpunktes (gegeben durch `COP`) in den Koordinatenursprung mit Blickrichtung (gegeben durch `-VPN`) entlang der negativen z-Achse, so wie es auf Seite 4-12 des Skriptes dargestellt ist. Die zweite Matrix ist die Projektionsmatrix, die die Augenkoordinaten im Unterschied zur Vorlesungen in einen Bildraum

$$B = \left\{(x, y, z)^T \mid -1 \leq x \leq 1, -1 \leq y \leq 1, -1 \leq z \leq 1\right\}$$

transformiert, an dem geclippt werden kann und welcher sich einfach auf den Bildschirm (den sogenannten Viewport) projizieren lässt.

Die Projektionsmatrix ist im Rahmenprogramm bereits mittels `glFrustum` gegeben und Ihre Aufgabe besteht zunächst darin, die ModelView-Matrix korrekt zu setzen. Verwenden Sie hierfür den Befehl `gluLookAt` der OpenGL Utility Library, deren Dokumentation Sie auch auf
`http://www.opengl.org/sdk/docs/man/`

finden. Sobald dies implementiert ist, sollte Ihr Programm bereits—bis auf das Clipping—die gleichen Bilder erzeugen, wie das Programm zu Aufgabe 13. Das Clipping in z-Richtung funktioniert ebenfalls bereits.

Um nun identische Bilder zur Lösung von Aufgabe 13 zu erhalten, muss ein künstliches Clipping auf die durch `clip.uminf`, `clip.umaxf`, `clip.vminf` und `clip.vmaxf` gegebene Ausdehnung auf der Projektionsebene erfolgen.

Ist die Projektionsebene gegeben durch $[\texttt{umin}, \texttt{umax}] \times [\texttt{vmin}, \texttt{vmax}]$, ergibt sich der geclippte Bereich als $[\texttt{umin} \cdot \texttt{clip.uminf}, \texttt{umax} \cdot \texttt{clip.umaxf}] \times [\texttt{vmin} \cdot \texttt{clip.vminf}, \texttt{vmax} \cdot \texttt{clip.vmaxf}]$.

Korrektes Clipping kann auf zwei Arten erzielt werden:

1. Mittels `glClipPlane` können Sie vier Clipping-Ebenen definieren, die durch den Koordinatenursprung verlaufen und den Sichtbereich in x- und y-Richtung auf der Projektionsebene auf den oben erwähnten geclippted Bereich einschränken. Hierfür müssen die Normalenvektoren der Ebenen in Abhängigkeit von `znear` und der Größen des geclippten Bereiches ermittelt werden.
2. Das Sichtfenster (Viewport) kann entsprechend beschränkt werden. Anschließend muss die Berechnung der Projektionsmatrix (`glFrustum`) angepasst werden, so dass die Projektion nun für den geclippten Bereich stattfindet.

Testen Sie Ihr Programm mit den gleichen Eingabedateien wie auch Aufgabe 13.

Aufgabe 21 *(Hidden-Surface-Verfahren und Beleuchtung mit OpenGL)*

Im Verzeichnis `/home/bildgen/Aufgaben/opengl-3` finden Sie eine OpenGL-Implementierung der Programme aus den Aufgaben 17 und 18. Aktivieren Sie die Verwendung des z-Buffer-Verfahrens mittels der Befehle `glEnable` und `glDepthFunc` mit passenden Parametern.

Informationen dazu finden Sie auf der Webseite
`http://www.opengl.org/sdk/docs/man/`

Abgabe: Fr., 13.12.2023, 13:15 Uhr

Senden Sie Ihre Lösungen der Theorie-Aufgaben und Ihre Programme per E-Mail an
`bildgen@studs.math.uni-wuppertal.de` .