Institut für Visualisierung
und Interaktive Systeme

Dr. Guido Reina Dr. Michael Krone

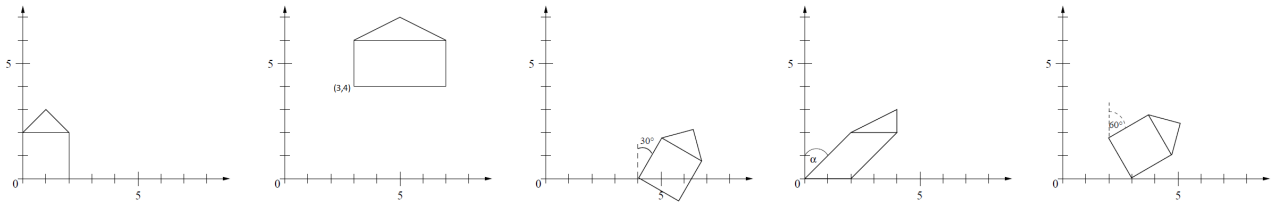
7. Übungsblatt zur Vorlesung Computergraphik im WS 2017/18

Abgabe am Mittwoch, 13.12.2017 (12:00)
Besprechung am Montag, 18.12.2017 17:30 - 19:00 Uhr

Aufgabe 1 *Affine Abbildungen, 2D-Transformationen [2 Punkte]*

Gegeben ist ein Haus in der Grundstellung, dessen linke untere Ecke im Ursprung $(0, 0)$ des Koordinatensystems steht. Stellen Sie die zugehörigen homogenen Transformationsmatrizen für die folgenden Transformationen auf. Orientieren Sie sich dabei an den vorhandenen Referenzbildern.

1. **Translation und Skalierung:** Verschieben Sie das Haus so, dass der linke untere Eckpunkt auf den Punkt (3,4) verschoben wird und skalieren Sie es um den Faktor zwei in x-Richtung.
2. **Translation und Rotation:** Verschieben Sie das Haus auf den Punkt (4,0) und führen Sie eine Rotation um 30° im Uhrzeigersinn durch.
3. **Scherung:** Führen Sie eine Scherung mit dem Scherwinkel $\alpha = 45°$ in x-Richtung durch.
4. **Rotation:** Gehen Sie für diese Aufgabe davon aus, dass sich die linke untere Ecke des Hauses an dem Punkt (2,2) befindet. Transformieren Sie das Haus, so dass die in der Abbildung *Rotation* dargestellte Endposition erreicht wird, die eine Rotation um 60° beinhaltet.



Grundstellung Translation und Skalierung Translation und Rotation Scherung Rotation

Hinweis: Die Teilaufgaben sind unabhängig voneinander. Teilaufgaben 1-3 gehen jeweils von der Grundstellung als Ausgangsposition aus. In Teilaufgabe 4 wird die eigens beschriebene Ausgangslage verwendet.

Aufgabe 2 *Rasterisierung [8 Punkte]*

In dieser Aufgabe werden Funktionen implementiert, die in einem 3D-Software-Rasterisierer Verwendung finden könnten.
Als Grundlage dient uns ein fast vollständig implementierter Rasterisierer, der zur Visualisierung den ImageViewer verwendet. Falls Sie Schwierigkeiten haben sollten das Programm zum Laufen zu bringen, halten Sie sich an die Informationen aus Aufgabenblatt 4. Das Programm sollte unter Windows mit Visual Studio 2015 direkt lauffähig sein und ein schwarzes Bild, sowie ein Menü anzeigen.

Hinweis: Sollten Sie Visual Studio 2017 verwenden, achten Sie darauf das Platform Toolset `v140` einzustellen, bzw. beim ersten Öffnen des Projektes die Version nicht! auf `v141` umzustellen.

Anders als auf dem letzten Blatt werden wir dieses Mal kein Raytracing anwenden, sondern die Objekte mittels Transformationen in das Bildschirmkoordinatensystem überführen. Die Hauptarbeit des Rasterisierers übernimmt dabei die Klasse *Rasterizer*. Im Ordner *Scene* finden Sie Objekte und Lichtquellen, die für die Generierung einer Szene verwendet werden können. Alle Objekte stellen ein Dreiecks-Mesh zur Verfügung.
Der fertig implementierte Rasterisierer soll drei Methoden zur Visualisierung der Dreiecke beinhalten:

- **POINTS:** Alle Eckpunkte der Dreiecke werden angezeigt.
- **WIREFRAME:** Die Kanten der Dreiecke werden gezeichnet.
- **FILLED:** Das Dreieck wird ausgefüllt gemalt.

Zudem werden die Farben und z-Werte des Objekts zwischen den Punkten interpoliert. Bei der Visualisierung der Kanten linear, bei ausgefüllten Dreiecken baryzentrisch.

Hinweis: Um das Zeichnen zu beschleunigen, wird OpenMP verwendet. Zum Debuggen des Programms empfiehlt es sich jedoch, dieses zu deaktivieren.

Hinweis: Verwenden Sie die automatische Update-Funktion am besten nur im Release-Modus, da das Zeichnen etwas Zeit in Anspruch nehmen kann.

1. **Modelltransformation** [1 Punkt]
   Implementieren Sie die Transformation der Punktpositionen und der Normalen in der Funktion *Rasterizer::drawObject*. An dieser Stelle wird auf einen einzelnen Punkt des Dreiecks zugegriffen und dieser von den Objektkoordinaten in Weltkoordinaten überführt. Verwenden Sie die Transformation *global_trafo* und weisen Sie die berechnete Position und Normale in Weltkoordinaten den Variablen *point_world* und *normal_world* zu. Ist die Transformation korrekt implementiert, können manche Elemente der Szene schon grob gerendert werden.

2. **Beleuchtungsberechnung und Interpolation** [1 Punkt]
   Als Beleuchtungsmodell wird das Lambertsche Gesetz verwendet. Dieses muss ebenfalls in der Funktion *Rasterizer::drawObject* implementiert werden. Die Farbe und die Intensität des Lichts werden dabei getrennt von der Lichtquelle abgefragt. Orientieren Sie sich dazu an der bereits vorhandenen Implementierung der ambienten Beleuchtung. Um auch Punkte innerhalb des Dreiecks korrekt zu beleuchten, muss zudem die Funktion *calculateBarycentricCoords* in der Datei *Math.cpp* implementiert werden. Zu diesem Zeitpunkt sollte das Rendern der Dreiecks- und Würfel-Szene mit ausgefüllten Dreiecken vollständig möglich sein.

3. **Sphärische Koordinaten** [0.5 Punkte]
   Zum Anzeigen der Kugeln in der Kugel- und in der komplexen Szene, wird noch die Umrechnung der sphärischen Koordinaten in kartesische benötigt. Implementieren Sie diese in der Datei *Math.cpp* und orientieren Sie sich dabei an der ISO-Konvention, mit $\theta \in [0, \pi)$ und $\phi \in [0, 2\pi)$. Nun können auch die Kugeln gerendert werden.

4. **Linienrasterisierung** [4 Punkte]
   Bisher können die Dreiecke auf zwei verschiedene Arten visualisiert werden: als ausgefüllte Dreiecke oder deren Eckpunkte. In dieser Teilaufgabe soll es nun auch möglich gemacht werden, dass die Kanten der Dreiecke gezeichnet werden. Diese Darstellung wird auch als *Wireframe* bezeichnet. Der wohl bekannteste Algorithmus zum Rasterisieren von Linien ist der von Bresenham. Implementieren Sie den Algorithmus in der Funktion *Rasterizer::rasterizeLine*. Die Funktion erhält die Endpunkte einer Linie in Bildschirmkoordinaten als Eingabe. Testen Sie Ihre Implementierung mit allen Szenen, da vor allem bei vertikalen Linien Artefakte auftreten können. Jetzt sollten alle Funktionen des Rasterisieres vollständig zur Verfügung stehen.

   Hinweis: Verwenden Sie die Funktion *Rasterizer::setPixel* zum Setzen der einzelnen Pixel der Linie.

5. **Theorie** [1.5 Punkte]
   Folgende Fragen sollen abfragen, ob Sie die Transformationen, sowie die Schwierigkeiten beim Rasterisieren, verstanden haben.

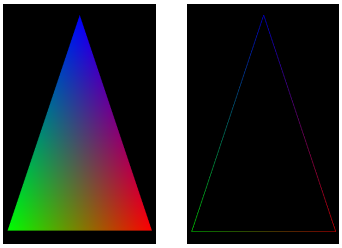
   (a) **Transformation:** Werfen Sie einen Blick in die Funktion *createCube* in der Datei *main.cpp*. In welcher Reihenfolge werden die Transformationen des Würfels ausgeführt? Begründen Sie Ihre Antwort!

   (b) **Z-Buffer:** Der gegebene Rasterisierer verwendet einen sogenannten Z-Buffer. Wofür wird dieser gebraucht und welche Problematik ergibt sich beim parallelen Rendern, wie es z.B. auf der GPU geschieht?

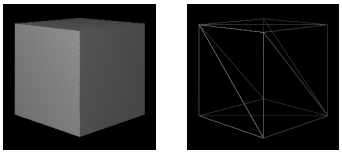
   (c) **Clipping:** Für die verschiedenen Visualisierungsmethoden des Rasterisierers werden unterschiedliche Ansätze für Clipping verwendet. Welche sind das und warum sind die Implementierungen im Falle der Wireframe-Methode und beim Darstellen der ausgefüllten Dreiecke eigentlich unzureichend?

6. **Eigene Szene (optional)**
   Als optionale Aufgabe haben Sie die Möglichkeit eine eigene Szene zu entwerfen. Dazu können Sie die bereits vorhandenen Objekte verwenden oder neue erstellen. Die schönsten und kreativsten Szenen werden in der Übungsgruppe vorgestellt.

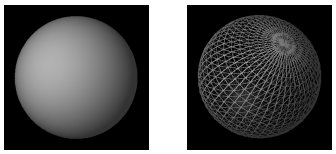
Die Ergebnisse für die verschiendenen Szenen sollten wie folgt aussehen:



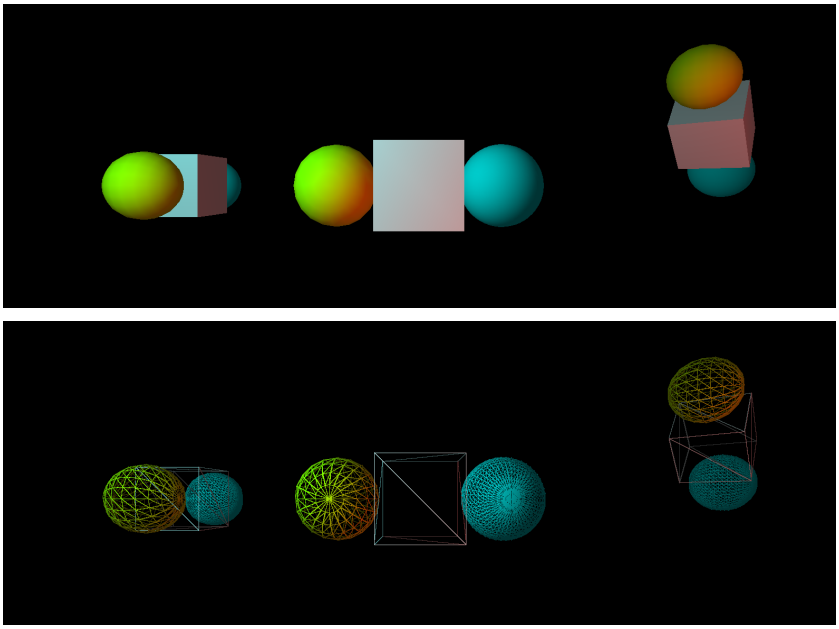
Triangle



Cube



Sphere



Complex Scene