Prof. Dr. R. Dörner
Fachbereich Design Informatik Medien
HS RheinMain

Praktikum zur Computergraphik

OpenGL-Praktikum

OpenGL ist eine sehr grundlegende API für Computergraphik, die gegenüber anderen APIs wie DirectX den Vorteil hat, auf sehr vielen Plattformen verfügbar zu sein. Im Praktikum soll ein einfaches OpenGL-Programm schrittweise aufgebaut werden. Damit wird auch ein Vergleich zwischen VRML und OpenGL möglich. Dabei stehen VRML und OpenGL für typische Ansätze in der Computergraphik: low-level API, Programmier-Ansatz (OpenGL), Szenengraph-Ansatz, deskriptiver Ansatz (VRML).Um sich auf die Computergraphik-Aspekte konzentrieren zu können, wird WebGL verwendet – hier können die OpenGL-Programme plattformübergreifend im Browser dargestellt werden, auch entfällt ein Kompilierungsvorgang. Schließlich ist es aufgrund der Dynamik auf dem Gebiet der Computergraphik wichtig, sich selbst zusätzliches Wissen anzueignen und sich selbständig in ein Programm, eine Sprache oder eine API einzuarbeiten. Auch dies soll hier als wesentliches Ziel geübt werden: der Stoff der Vorlesung allein genügt nicht, um die Aufgaben zu bearbeiten - Eigeninitiative ist also nötig.

Das OpenGL Praktikum darf nicht in Gruppen durchgeführt werden

GL.1 (zu G: OpenGL und WebGL)

Ziel der Aufgabe ist es, überhaupt eine Art "Hello World" in OpenGL zur Ausführung zu bringen, die Ausführungsumgebung für WebGL kennen zu lernen und sich die Grundstruktur eines einfachen OpenGL Programms zu vergegenwärtigen.

(a) Gegeben ist ein OpenGL Programm in Form von WebGL mit einer Webseite *praktikum.html* und einigen zugehörigen Javascript-Dateien. Öffnen Sie die HTML-Seite in einem Webbrowser, z.B. Firefox. Es sollte ein Würfel vor einem hellblauen Hintergrund zu sehen sein. Schauen Sie sich den Quelltext von *praktikum.html* und *praktikum.js* an. Dies können Sie mit einem normalen Texteditor tun, aber auch speziellen Editoren mit Syntaxhighlighting, wie sie z.B. in der Entwicklungsumgebung von Firefox zu finden sind.

(b) Benennen Sie die beiden Dateien um, indem Sie den Namen „*praktikum*“ durch „*G19_Gruppenbuchstabe_IhreMatrikelnummer*“ ersetzen. Testen Sie, dass Sie nach wie vor den Würfel sehen.

(c) Finden Sie die Stelle im Javascript-Code, an der Sie durch entfernen der Kommentarzeichen dafür sorgen, dass in einer Endlosschleife ein Frame nach dem anderen gezeichnet wird. Der Würfel dreht sich. Durch Klicken auf die Buttons „Rotate X“, „Rotate Y“ und „Rotate Z“ können Sie die Drehachse jeweils auf die x-Achse, y-Achse oder z-Achse ändern.

(d) Ändern Sie das Programm so ab, dass beim Klicken auf den „Rotate On / Off“ Button die Rotation gestartet bzw gestoppt wird.

(d) (Zusatzaufgabe) Ergänzen Sie das Programm um Code, der die Worte „FPS“ auf der HTML Seite durch den aktuellen Wert an Frames pro Sekunde ersetzt. Hinweis: es wurde schon eine Variable *now* im Code vorbereitet und auch gezeigt, wie man in Javascript die

aktuelle Zeit (in Sekunden) ermitteln kann. Der Anzahl der Frames pro Sekunde soll nur alle 20 Frames ermittelt werden.

GL.2 (zu G, H.1: OpenGL Objektmodelle)

Ziel der Aufgabe ist es, Objekte in der Szene gezielt anzuordnen und auch neue Objekte zu definieren.

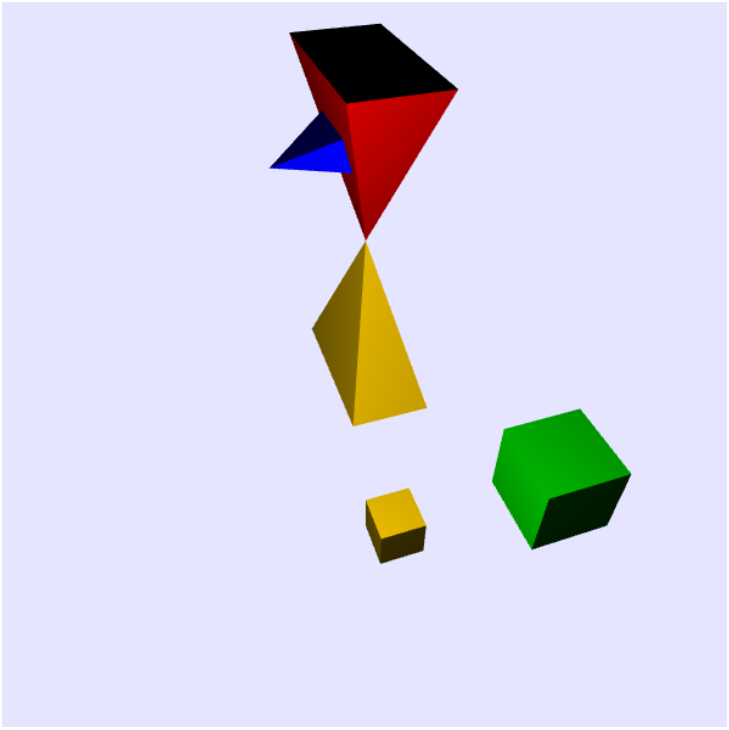

Abbildung 1: Die Szene nach Aufgabe GL.2

(a) Das Objektkoordinatensystem des Würfels in der Szene soll das Weltkoordinatensystem sein. Verschieben Sie den Würfel in diesem Weltkoordinatensystem aus dem Ursprung um 5 in x-Richtung und 1 in z-Richtung. Der Würfel soll sich mit den Buttons auf der HTML-Seite nach wie vor drehen lassen. Zusätzlich soll der Würfel aber auch sich um seine eigene z-Achse um seinen Mittelpunkt drehen (ca. 10 Sekunden für eine volle Umdrehung).

(b) Fügen Sie einen weiteren Würfel ein. Dieser soll sich in Weltkoordinaten an der Position (5, 0, -3) befinden und eine grüne Farbe haben. Dieser Würfel soll sich um seine eigene x-Achse um seinen Mittelpunkt drehen. Dabei soll sich der grüne Würfel doppelt so schnell drehen als der andere Würfel und auch doppelt so groß sein. Auch der grüne Würfel soll sich mit den Buttons auf der HTML-Seite drehen lassen.

(c) Zeichnen Sie eine Pyramide mit einer rechteckigen Grundfläche. Die Grundfläche liegt in der xz-Ebene, der Mittelpunkt des Rechtecks ist der Ursprung, die Länge des Rechtecks in x-Richtung beträgt 4, die Länge des Rechtecks in z-Richtung beträgt 2. Die Spitze der Pyramide liegt an der Stelle (0,4,0) im Objektkoordinatensystem der Pyramide. Die Pyramide soll die selbe Farbe wie der erste Würfel haben. Fügen Sie diese Pyramide in die Szene ohne weitere Transformation ein. Die Pyramide soll sich aber wie die Würfel mit den Buttons auf der HTML-Seite drehen lassen.

(d) Fügen Sie eine weitere Pyramide in die Szene ein. Diese soll die Farbe rot haben. Sie soll so positioniert werden, dass sie über die bestehende Pyramide an der Spitze berührt und die Grundflächen beider Pyramiden genau übereinander liegen (vgl. Abbildung 1).

(e) Fügen Sie eine dritte Pyramide in die Szene ein. Diese soll blau gefärbt sein und nur 40% der Größe der anderen Pyramiden haben. Diese Pyramide soll genau auf dem Dreieck der roten Pyramide stehen, das der Kamera anfangs zugewandt ist (vgl. Abbildung 1). Der Mittelpunkt der Grundfläche der blauen Pyramide befindet sich genau in der Mitte des Dreiecks.

(f) Alle drei Pyramiden sollen an der Rotation der gesamten Szene teilnehmen, die durch die entsprechenden Buttons auf der HTML-Seite gesteuert werden kann.

Hinweis: Lösen Sie die Aufgaben nicht durch herum probieren, sondern durch Berechnung. Hierzu kann auch das Konzept des Szenengraphs hilfreich sein.

GL.3 (zu H.3 „Kameramodell und Koordinatensysteme“)

Ziel der Aufgabe ist es mit den Kameraeinstellungen von OpenGL sich vertraut zu machen (sowohl innere als auch äußere Orientierung der Kamera) und auch eine Kamera gezielt platzieren zu können.

(a) Ändern Sie das Programm so, dass bei der Auswahl von „x-Achse“ in der Auswahlliste der HTML-Seite die Kamera einen Abstand 10 vom Ursprung hat, sich auf der x-Achse befindet und in Richtung der negativen x-Achse schaut.

(b) Ändern Sie das Programm so, dass die Auswahlen „y-Achse“ und „z-Achse“ jeweils entsprechend wie bei Aufgabe (a) für die „x-Achse“ bereits umgesetzt zu einer Änderung der Kameraeinstellung führen.

(c) Ändern Sie das Programm so, dass für die Auswahl „Pyramidenspitze“ die ursprüngliche Kameraposition, die nach wie vor unter „Originalkamera“ auswählbar sein soll, wie folgt geändert wird: der Berührungspunkt der beiden Pyramidenspitzen ist genau im Mittelpunkt des Bildes zu sehen.

(d) Ändern Sie temporär den Öffnungswinkel der Kamera auf 30°. Wie ändert sich das Bild?

(e) Ändern Sie temporär den Abstand der Near-Clipping-Plane auf 15. Wie ändert sich das Bild?

(f) Ändern Sie temporär die Aspect Ratio auf 16:9 in den inneren Kameraparametern. Wie ändert sich das Bild? Warum hat das im Browser angezeigte Bild immer noch ein Seitenverhältnis von 1:1?

GL.4 (zu H.1, H.4 „Beleuchtungsrechnung im Vertex-Shader“)

Ziel der Aufgabe ist es, den prinzipiellen Unterschied zu erkennen, ob die Beleuchtungsrechnung auf „GPU-Seite“ durchgeführt wird oder die Farbwerte der Eckpunkte einfach von der „CPU-Seite“ festgelegt werden. Außerdem sollten Sie das am häufigsten in der Computergraphik verwendete Beleuchtungsmodell, das Phong-Beleuchtungsmodell, anhand der in der Vorlesung hergeleiteten Formeln selbst in einem Shader umsetzen können.

(a) Ändern Sie das Programm so, dass die Farbwerte des kleineren Würfels nicht mehr von der Beleuchtungsrechnung im Vertex-Shader ermittelt werden, sondern die Farbwerte, die von der CPU-Seite übergeben werden, zur Verwendung kommen. Ihr Würfel sollte dann auf jeder Würfelseite eine andere Farbe haben.

(b) Ändern Sie die Farbwerte des kleineren Würfels auf CPU-Seite so, dass zwei gegenüberliegende Würfelseiten schwarz und die restlichen Seiten rot dargestellt werden. Warum kann man den Übergang von einer roten Würfelseite zur benachbarten roten Würfelseite schlecht erkennen?

(c) Im Vertex-Shader ist bereits die Formel für die diffuse Beleuchtung aus dem Phong-Beleuchtungsmodell implementiert worden. Dies soll in den folgenden Teilaufgaben nun für die ambiente Beleuchtung und die spekulare Beleuchtung auch geschehen. Legen Sie dazu die benötigten zusätzlichen Material- und Lichteigenschaften auf „CPU-Seite“ fest und übergeben Sie diese Werte an den Vertex-Shader. Wählen Sie dabei 100.0 für den Wert der Shininess.

(d) Ergänzen Sie den Vertex-Shader um Code, der die ambiente und spekulare Beleuchtung gemäß den Formeln des Phong-Beleuchtungsmodells (vgl. Vorlesung) berechnet. Nutzen Sie bei der Berechnung des spekularen Anteils die Näherung von Blinn. Ermitteln Sie den endgültigen Farbewert eines Eckpunktes aus allen drei Beleuchtungsteilen. Wenn Sie die Szene aus Richtung der x-Achse betrachten, sollten Sie auf der gelben Pyramide nun ein deutliches Highlight sehen (vgl. Abb.2).

(e) Wie ändert sich das Bild, wenn Sie temporär den Wert der shininess ändern?

(f) Wie ändert sich das Bild, wenn Sie temporär die Intensität des ambienten Lichts ändern?

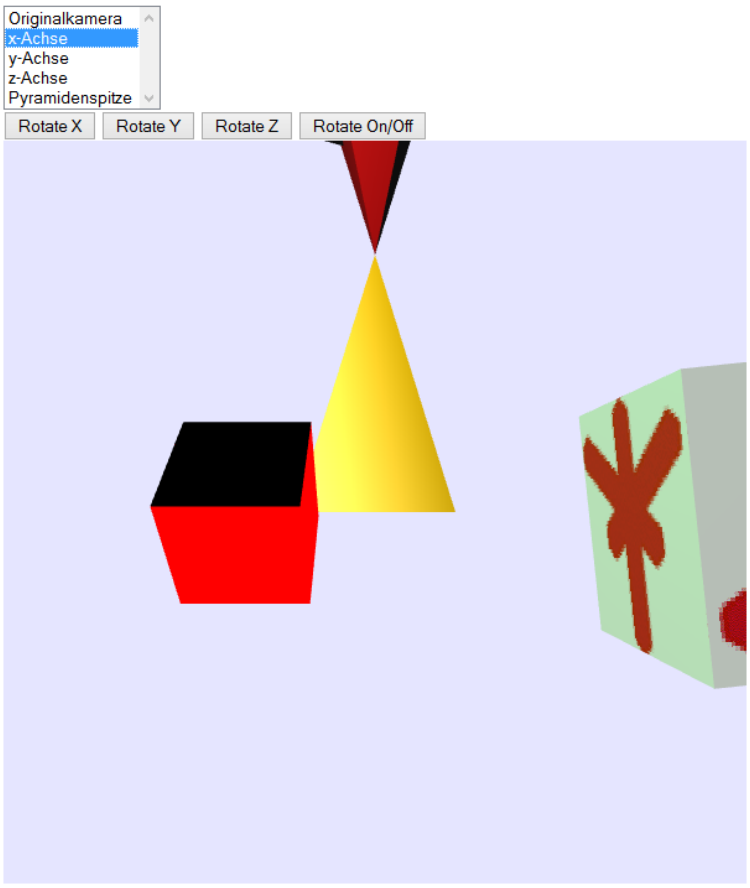

Abbildung 2: Die Szene nach Aufgabe GL.5

GL.5 (zu I: Rasterisierung und Texturierung)

Texturierung ist ein wichtiges Instrument, um die visuelle Qualität von computergenerierten Bildern zu erhöhen. Ziel der Aufgabe ist es, den grundlegenden Mechanismus von Texturen in OpenGL zu verstehen.

Hinweis: Im Folgenden werden lokale Dateien eingelesen (z.B. hsrm.gif). Die Sicherheitseinstellungen vieler Browser verhindern dies. Daher ist hier eine Freigabe durchzuführen, z.B. bei Firefox „about:config“ eingeben, suchen nach „security.fileuri.strict_origin_policy“ und den Wert auf “false“ setzen.

(a) Für die Texturierung soll das Bild „hsrm.gif“ genutzt werden. Es wird zur Verfügung gestellt, indem in den Body der HTML Seite **`<img id="texImage" src="hsrm.gif" hidden </img>`** einfügen. Um gefunden zu werden, muss das Bild in dasselbe Verzeichnis wie die HTML-Seite kopiert werden. Das „hidden“ bedeutet, dass das Bild in die HTML-Seite nur eingebunden, aber nicht angezeigt wird. Im Script kann auf das Bild zugegriffen werden durch: **`var image = document.getElementById("texImage");`**

(b) Fügen Sie in Ihre Javascript-Datei entsprechende OpenGL-Befehle ein, welche das Bild einlesen und als Textur für OpenGL aufbereiten. Erstellen Sie dabei auch eine MipMap aus der Textur. Das Resultat sollten Sie so an die „GPU-Seite“ übergeben, dass dort unter einer Variablen vom Typ **`sampler2D`** auf die Textur zugegriffen werden kann.

(c) Für die Texturierung müssen Sie noch jedem Eckpunkt eine Texturkoordinate geben. Legen Sie auf „CPU-Seite“ die entsprechenden Daten an und übergeben Sie sie an die „GPU-Seite“.

(d) Ändern Sie die Shader-Programme so, dass die Textur durchgeführt wird.

(e) Ändern Sie die Shader Programme so, dass die Farbe des Würfels zu 70% aus der Farbe der Textur und zu 30% aus der durch die Beleuchtungsrechnung ermittelten Farbe durch Farbmischung ergibt (vgl. Abb. 2). Verwenden Sie die eingebaute GLSL-Funktion **`mix`**.

GL.6 (zu J.2 Fragment-Shader)

Das Programmieren der Renderpipeline mittels GLS erlaubt hohe Flexibilität im Shading. Um ein Beispiel kennen zu lernen soll ein Cartoonshading realisiert werden.

(a) Um die Auswirkungen des Shadings besser erkennen zu können, soll ein komplexeres Objekt mit einer größeren Anzahl von Eckpunkten definiert werden. Die Eckpunktkoordinaten, zugehörige Normale und Indices sind in einer externen Datei teapot.JSON im JSON-Format gespeichert. Nutzen die Codefragmente und Hinweise in der Datei teapot.js, um eine mit dem Faktor 0.3 skalierte Teekanne an Position (-5, 0, 6) zu zeichnen. Die Teekanne soll dabei ständig um ihre eigene y-Achse rotieren.

(b) Realisieren nun einen Cartoon-Shader, der nur auf die Teekanne angewendet wird (vgl. Abb. 3): zunächst soll eine Beleuchtungsrechnung nach Phong durchgeführt werden, um für jedes Fragment einen Farbwert zu bestimmen. Von diesem Farbwert ist die Helligkeit H zu bestimmen. Falls H kleiner als 0.3 ist, soll die Resultatsfarbe ein Grauwert mit Helligkeit 0.2 sein. Falls H größer als 0.75 ist, soll die Resultatsfarbe (0.9, 0.7, 0, 1) betragen und sonst (0.6, 0.4, 0.1, 1).

(c) Fügen Sie einen Button auf der Webseite hinzu, um das Cartoon-Shading ein- und ausschalten zu können.

(d) (Zusatzaufgabe) Fügen Sie Slider ein, um die beiden Thresholds im Cartoon-Shading, sowie die Intensität des ambieten Lichts und die Shininess interaktiv verändern zu können.

(e) (Zusatzaufgabe) Auf dem grünen Würfel soll die Textur vierfach erscheinen.

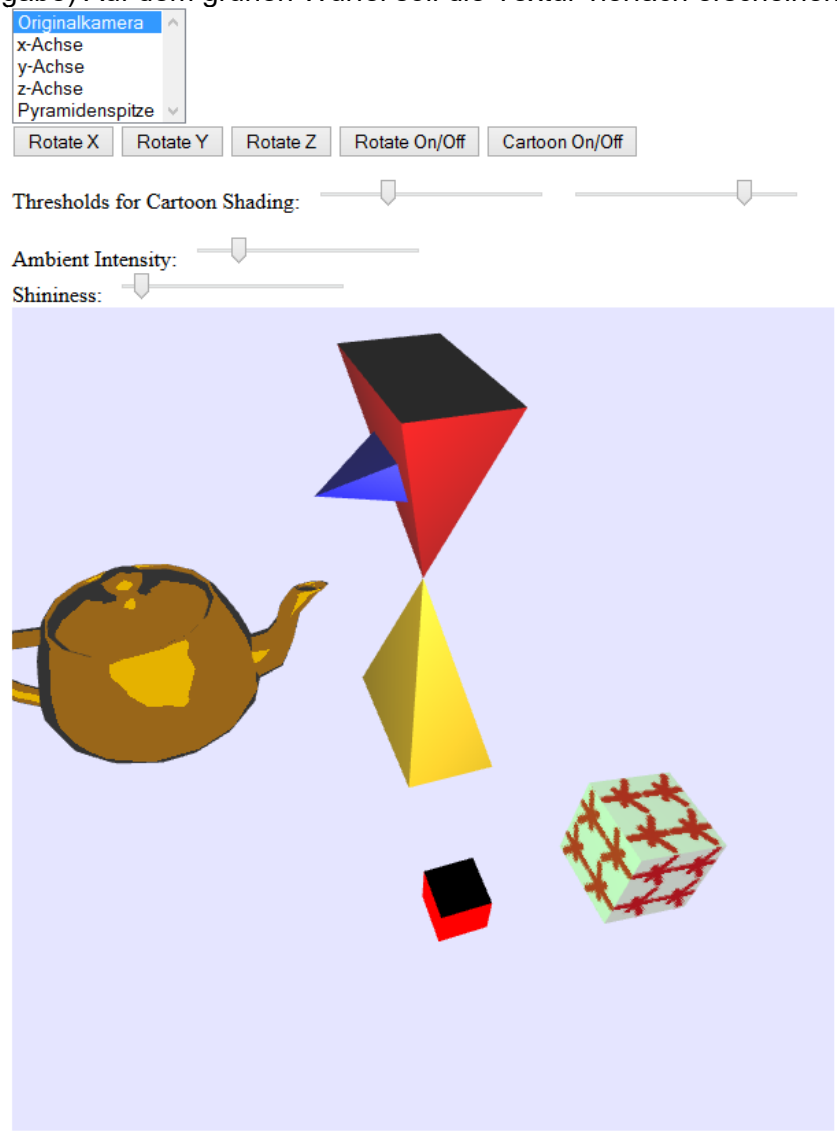

Abbildung 3: Die Szene nach Aufgabe GL.6

Nutzen Sie zur Bearbeitung des OpenGL Praktikums auch externe Informationsquellen (z.B. Bücher, Tutorials im Internet etc.) !!