

Übungen zu Bildverarbeitung und Graphikprogrammierung in C++

In den folgenden Aufgaben werden Sie die grundlegenden Funktionen und Mechanismen zur Shader Programmierung mit GLSL (OpenGL Shading Language) kennen lernen. Es ist ein objektorientiertes Framework zur Verwaltung von GLSL Shader und GLSL Programmen gegeben. Code für das Laden, Initialisieren von Objekten und OpenGL ist bereits im Framework vorhanden und implementiert. In den folgenden Aufgaben werden Sie sich darauf konzentrieren OpenGL Vertex und Fragment Shader hinzufügen um die Szene zu animieren und realistisch zu beleuchten. Text Dateien für die zu implementierenden Shader sind bereits vorhanden und sind im Laufe der Aufgaben mit GLSL Code zu ergänzen.

Aufgabe 11 (H) GLSL Hello World!

- a) Implementieren Sie die Methode **renderGeometry**. In der Methode sollen fünf farbige Objekte auf einer blauen Fläche gezeichnet werden. Die Fläche ist bereits integriert. Fügen Sie OpenGL Code zum Zeichnen von folgenden Objekten hinzu
- Roter Einheitswürfel mit Kantenlänge 48.0 und Zentrum (0,0,0)
 - Grüne Kugel mit Radius 25.0 und Zentrum (-60.0,0,0)
 - Torus in Magenta (InnerRadius 8.0, OuterRadius 16.0 , RGB COLOR = 255,0,255) und Zentrum (0,0,60)
 - Gelber Kegel (BaseRadius 25.0 und Höhe 50.0) mit von der Fläche wegweisender Spitze und Zentrum (60.0,0,-24.0)
 - Teekessel (Relative Grösse 25.0) in Cyan mit Zentrum (0,0,-60.0)
- b) Implementieren Sie einen Vertex Shader (Datei **fixedfunc_purevertex.vs**) der den Vertex Processing Teil der OpenGL Fixed Functionality Pipeline nachbildet. Jeder Vertex soll durch den Shader von Object Space nach Clip Space transformiert werden, zusätzlich soll die primäre Farbe (FrontColor) durchgereicht werden.
- c) Implementieren Sie je einen Vertex Shader (Datei **fixedfunc.vs**) und Fragment Shader (Datei **fixedfunc.fs**) die die Funktion der OpenGL Fixed Functionality Pipeline nachbilden. Der Vertex Shader soll jeden Vertex von Object Space nach Clip Space transformieren und die primäre Farbe (FrontColor) durchreichen. Zusätzlich soll nun auch noch die Eye-Space Normale berechnet werden. Im Fragment Shader soll für jedes erzeugte Fragment die korrekte Farbe in den Framebuffer geschrieben werden.

Wichtige builtin Vertex Shader Variablen für das erfolgreiche Lösen der Aufgabe sind u.a. `gl_NormalMatrix`, `gl_ModelViewProjectionMatrix`, `gl_ModelViewMatrix`.

Aufgabe 12 (H) GLSL Vertex and Fragment Shaders

In dieser Aufgabe sind einige GLSL Shader zu implementieren. Die Textdateien in die der Shader Code einzufügen ist, sind angegeben und zu vervollständigen.

- a) Implementieren Sie einen Vertex Shader (**ptsize_purevertex.vs**) der Vertices von Object Space nach Clip Space transformiert, die primäre Farbe (FrontColor) durchreicht und die PunktGröße (PointSize) abhängig von der Distanz Kamera zu Vertex berechnet. PointSize soll in dem Shader nach folgender Formel (PseudoCode, kein GLSL) berechnet werden.

$$PointSize = \frac{50000.0}{distance(vertex, eye)^2}$$

Wichtige builtin Vertex Shader Variablen für das erfolgreiche Lösen der Aufgabe sind u.a. `gl_PointSize`, `gl_ModelViewMatrix`. Die Position der Camera kann auch mittels der built-in Variable `gl_ModelViewMatrixInverse[3]` ermittelt werden.

- b) Implementieren Sie einen Vertex Shader **diffuse_purevertex.vs** für diffuse Beleuchtung. Der Shader soll Vertices von Object nach Clip Space transformieren und nun nicht mehr nur die primäre Farbe durchreichen sondern diese über die Formel für diffuse Beleuchtung für eine Lichtquelle berechnen.

$$FrontColor = Color * vec4(max(0.0, NdotL))$$

wobei $NdotL$ das Skalarprodukt der normalisierten Vektoren N (Normalenvektor in Eye-Space) und L (Vektor von Vertex zur Lichtquelle (uniform vec3 Variable `lightPos0`) in Eye Space)

Wichtige builtin Vertex Shader Variablen für das erfolgreiche Lösen der Aufgabe sind u.a. `gl_NormalMatrix`, `gl_ModelViewMatrix`.

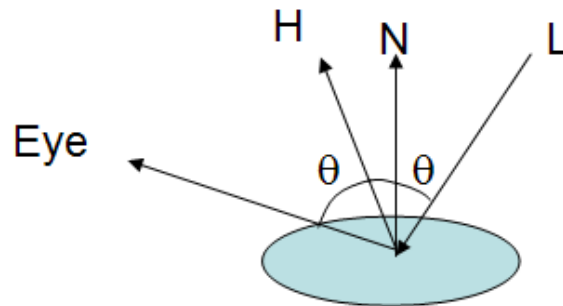
- c) Erweitern Sie den Code des Vertex Shader aus der vorigen Teilaufgabe so dass nun auch specular Lighting berechnet wird. Die resultierende Farbe ist nun die Summe aus den beiden Anteilen von diffuse und specular. Der Code soll in der Datei **specular_purevertex.vs** abgelegt werden. Berechnen Sie die Beleuchtung nun nicht mehr direkt in der Haupt Vertex Shader Funktion sondern in einer separaten Funktion `vec4 calcLighting(vec3 N, vec4 V, vec3 lightPos, float specularExp)` die sie in der Hauptfunktion aufrufen. In der Hauptfunktion steht dann $FrontColor = calcLighting(N, V, lightPos0, specularExp)$. Specular Lighting ist in `calcLighting` nach folgender Formel zu berechnen

$$specular = vec4(pow(max(0, NdotH), specularExp))$$

wobei $NdotH$ das Skalarprodukt der normalisierten Vektoren N (Normalenvektor in Eye-Space) und $H = L + Eye$ (Halbvektor zwischen Licht und Auge in Eye Space). In Abbildung 1 sind nochmal alle für die Berechnung wichtigen Vektoren zu sehen. Eye bezeichnet den Vektor vom Vertex zum Auge, L den Vektor von Lichtquelle zum Vertex, und H den Halbvektor zwischen den Eye und L. Setzen Sie `specularExp` auf einen Wert von 128.0.

- d) In dieser Teilaufgabe (Datei **stretch_purevertex.vs**) werden wir die Vertices im Vertex Shader im Object Space verändern bevor wir diese in den Clip Space transformieren. Die x,y und z Koordinate jedes Vertex und der Normalen soll mit einem Faktor (uniform vec3 stretch;) multipliziert werden. Danach soll die Beleuchtung über die Funktion `calcLighting` berechnet werden. Fügen Sie dafür den Code der Funktion in die Datei mit ein und rufen Sie Sie an der richtigen Stelle auf.
- e) In dieser Teilaufgabe (Datei **bulge_purevertex.vs**) werden wir nun eine zeitgesteuerte Animation der Vertices implementieren. Über die Formel

$$DisplacedVertex = Vertex + displacement * displacementDirection$$



ht

Abbildung 1: Vektoren zum Berechnen von specular Lighting.

soll die Vertices im Object Space bewegt werden. Der Faktor *displacement* soll abhängig von der Zeit (uniform float time;) mittels einer Sinus Schwingung berechnet werden.

$$displacement(t) = a \cdot \sin(frequency \cdot time + phase)$$

Fügen Sie dem Shader die nötigen uniform Variablen hinzu und implementieren Sie die Vertex Displacement Funktion indem Sie die Vertex Normale als Displacement Richtung verwenden. Die Beleuchtung soll wieder über die Funktion calcLighting berechnet werden. Fügen Sie dafür den Code der Funktion in die Datei mit ein und rufen Sie Sie an der richtigen Stelle auf.

Experimentieren Sie mit verschiedenen Parameter Settings oder implementieren Sie einen nicht nur von der Zeit aber auch vom Ort im Object Space abhängigen Displacement Faktor in dem Sie die x,y - und/oder z Koordinate der Vertices in die displacement Berechnung miteinbeziehen.

- f) In der letzten Teilaufgabe soll noch einmal diffuse und specular Lighting implementiert werden. Dieses Mal aber nicht pro Vertex sondern pro Fragment. Passen Sie dafür den Code aus den Vertex Shadern für diffuse und specular Lighting so an das in den Vertex Shadern die nötigen Vektoren berechnet und als varying Parameter an die Fragment Shader weitergegeben werden. Pro Fragment werden dann von OpenGL die Werte interpoliert. Der Code für diffuse Lighting soll in **diffuse.vs**, **diffuse.fs** abgelegt un der Code für specular Lighting in den Dateien **specular.vs**, **specular.fs** abgelegt werden.