Universität Leipzig
Institut für Informatik
Abteilung Bild- und Signalverarbeitung

Abgabe in der Woche des 22.**Mai**

Computergrafik-Praktikum
Sommersemester 2017
Betreuer: Baldwin Nsonga (nsonga'at'informatik.uni-leipzig.de)

Aufgabenblatt 2

Aufgabe 0: CGViewer

Windowsnutzer, die Probleme mit glActiveTexture haben können wie folgt vorgehen: Im Headerfile der getroffenen Klasse die Variable„QopenGLFunctions gl;“ hinzufügen und im Konstruktor „gl.initializeGLFunctions(QGLContext::currentContext()):“ aufrufen. Anstatt glActiveTexture kann nun gl.glActivetexture… verwendet werden.

Bitte laden Sie die aktuelle Version des CGViewer herunter. In dieser Version wurde die Klasse Light verändert und die (unvollständige) Klasse Skybox hinzugefügt. Hinzu kommt ein Ordner mit Bildern für die Skybox. Pflegen Sie die Änderungen ein.

Aufgabe 1: Schattierung und Beleuchtung

Jetzt, wo der Vertexshader dafür sorgt, dass die Modell-Vertices korrekt angezeigt werden, geht es in dieser Aufabe darum, die Modelle (bzw. die vom Modell berdeckten Pixel) mit den vom Modell vorgegebenen Farben im Pixelshader einzufärben. In dieser Aufgabe soll pixelgenaue Beleuchtung (per Pixel Lighting) mittels Phong Shading und Phong Beleuchtung umgesetzt werden. Dabei spielen die in der Szene beteiligten Lichter und die Modell- Farbattribute (Materialien) eine Rolle. Lichter und Materialien besitzen drei verschiedene Farbattribute: *ambient*, *diffus* und *specular*. Wir benutzen für das Praktikum eine vereinfachte Form des Beleuchtungsmodells von Phong, welches die Intensität für eine bestimmte Wellenlänge in einem Punkt *p* liefert. Setzen Sie für den Fall in welchem keine Lichtquelle in der Szene ist eine Standartlichtquelle. Diese soll die Position der Kamera haben und die drei Lichtanteile jeweils auf [1.0, 1.0, 1.0] gesetzt werden. Dazu führen Sie die Möglichkeit ein, bis zu 5 Lichtquellen in der Szene nutzen zu können. Hierzu eignet sich die Funktion *setUniformValueArray* d e r *QOpenGLShaderProgram*-Klasse. Materialinformationen sollen beim Shading aus den Modellen geladen werden. Hierzu müssen Sie in der *render*() Methode der Modellklasse arbeiten. CGViewer fasst bereits alle Vertices mit gleichen Materialeigenschaften als *Mesh* zusammen. Der Vektor *material* liefert für einen Index, den Sie unter *material in Mesh* finden, eine Struktur *Material.* Diese Struktur beinhaltet die notwendigen Materialeigenschaften, die Sie in der render-Methode an die Shader übergeben können. Die Gleichung für das Phong-shading können Sie in den Unterlagen finden. Beachten Sie, dass die Gleichung für jede Lichtquelle berechnet werden muss und die Summe mit der Farbe (Textur) multipliziert wird. Für den Reflektionsvektor gibt es eine eingebaute glsl-Funktion. Beachten Sie auch, dass Sie statt Kosinus(Winkel(vektor1, vector2)) das Skalarprorodukt(vector1, vector2) berechnen können - wenn die Vektoren normalisiert sind. Lichtquellen werden automatisch als Kugeln im Zentrum gerendert.

Aufgabe 2: Texturierung

In dieser Aufgabe sollen die Modelle durch eine Textur erweitert werden. Wir verwenden zweidimensionale Texturen, die an die Shader übergeben werden. Der CGViewer kann bereits Texturen laden und übergibt fertige Texturekoordinaten (*texCoords*) an den Vertex-Shader. Diese müssen nicht Transformiert werden! Für die Textur müssen Sie im Fragmentshader einen uniform sampler2D anlegen. Diesem wird die Textur beinhalten. In der *paintGL()* Methode können Sie an diese diesen den Wert 0 übergeben. (Die Textur wird so an die nullte Texturstufe gekoppelt). Mithilfe der Textur und der zugehörigen Texturkoordinaten können Sie den Farbwert an einem Pixel abrufen und mit Ihren berechneten Werten (Phong-Modell) multiplizieren. Vergessen Sie dabei nicht die Shinieness und den Alpha-Wert (4. Wert im Licht-Vektor) zu setzen.

(Ein Tutorial finden Sie hier: http://www.opengl-tutorial.org/beginners-tutorials/tutorial-5-a- textured-cube/)

Aufgabe 3: Skybox

In dieser Aufgabe soll die Scene um eine Skybox erweitert werden. Im Prinzip ist das ein Quader der um die Kameraposition aufgebaut wird. Dieser wird dann so texturiert, dass ein Horizont mit wenigen Kosten simuliert werden kann. Hierzu stelle ich Ihnen die Klasse Skybox, sowie Testbilder zur Verfügung. Die Skybox.h ist vollständig. Die Skybox.cpp soll von ihnen erweitert werden. Im Konstruktor werden die Buffer (Positionsdaten) schon gesetzt und die Bilder geladen. Ihre Aufgabe ist es hier die Textur (texture) zu erstellen (Tipp: std::make_shared< Klasse >(parameter) zum bauen eine unique-pointers). Bauen Sie die Textur als TargetCubeMap (siehe Qt Dokumentation). Die textur muss dann erstellt werden. Die grüße muss gesetzt werden und das Format festgelegt werden. Daraufhin muss der Speicher allokiert. Mit der setData Funktion müssen nun alle Bilder einzeln (abhänging von der Seitenfläche) der Textur übergeben werden. Das Bild selbst übergeben Sie als Parameter bitte wie folgt: „(const void*)skyImages[index]→constBits()“. Daraufhin wird MigMagFilters gesetzt und dann die MipMaps generiert. Die einzelnen Schritte sollten Sie in der Dokumentation finden können.

In der render(…)-Funktion muss das Zeichnen durchgeführt werden. Hierzu mussen positionsBuffer gebunden und mit glDrawArrays gezeichnet werden. Schauen Sie sich hierzu die render Methode der Klasse Light an und führen Sie die Schritte analog dazu aus (wir brauchen und haben aber nur positionBuffer).

Die vollständige Klasse sollte dann in der Scene initialisiert werden. Ein Weiteres Shaderprogramm sowie Vertex und Fragment Shader müssen hinzugefügt werden. Die Skybox besitzt eine render()-Funktion, die das zugehörige Shaderprogram sowie eine Kamera im world space bekommt. Kameraposition ist [0.0,0.0,0.0,1.0] im viewSpace → in den wordSpace transformieren.

Binden Sie die Cube Textur analog zu einer normalen textur an das Shaderprogramm. Im shadercode lesen sie als uniform im Fragment Shader ein „uniform samplerCube ihr_sky_sampler „ aus. Mit der glsl-Funktion texture(sampler ihr_sky_sampler, vec3 sky_vektor) können Sie die Farbe auslesen. Beachten Sie dass der sky_vektor in weltkoordinaten vorliegt und interpoliert wird. Er wird also aus position ermittelt. Die gl_Position muss in den screen space transformiert werden.

Damit die Skybox bei der Kamera bleibt muss diese verschoben werden. Übergeben Sie also beim Rendern ein uniform für die Kameraposition im Worldspace und verschieben Sie die Vertices dementsprechend.

WICHTIG: Damit das hintere Ende die Skybox beim heraus zoomen nicht das Modell überdeckt oder Ecken sichtbar werden bedienen wir uns eines einfachen Tricks. Die Skybox muss dabei zuerst mit Abgeschaltetem GL_DEPTH_TEST gerendert werden. Somit bekommen die Texturen keinen Tiefenwert und simuliert eine „unendliche“ Distanz. Nach dem Rendern der Skybox muss GL_DEPTH_TEST wieder aktiviert werden. Die Skybox sollte immer als erstes aufgerufen werden.