Universität Leipzig
Institut für Informatik
Abteilung Bild- und Signalverarbeitung

Abgabe in der Woche des **20.Mai**

Computergrafik-Praktikum
Sommersemester 2019

Aufgabenblatt III

Aufgabe 1: 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 2: 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 steht Ihnen die Klasse Skybox, sowie Testbilder (gerne können Sie eigene Bilder verwenden) 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 (QOpenGLTexture texture) zu erstellen. (Tipp: Nutzen Sie „std::make_shared< Klasse >(parameter)“ zum Erzeugen eines shared-pointers). Bauen Sie die Textur als TargetCubeMap (siehe Qt5 Dokumentation). Die Textur muss dann erstellt und die Dimensionen sowie das Format gesetzt werden. Daraufhin müssen Sie den Speicher freigeben. Mit der setData Funktion müssen nun alle Bilder einzeln (abhängig von der Seitenfläche) der Textur übergeben werden. Das Bild selbst übergeben Sie als Parameter wie folgt: „(const void*)skyImages[index]→constBits()“. Daraufhin müssen die mipmaps generiert werden. Die einzelnen Schritte sollten Sie in der Dokumentation finden können.

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. Binden Sie die Cube Textur analog zu einer normalen Textur an das Shaderprogramm. Im fragment shader muss das uniform mit dem vom Typ samplerCube gelesen werden. Mit der glsl-Funktion texture(…,…) können Sie die Farben der Textur auslesen. Beachten Sie dabei, dass die Texturkoordinaten für samplerCubes der position entsprechen und in Weltkoordinaten vorliegen müssen.

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

WICHTIG: Damit das hintere Ende die Skybox beim heraus fahren nicht das Modell überdeckt oder Ecken sichtbar werden bedienen wir uns eines einfachen Tricks. Die Skybox muss dabei mit deaktiviertem 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.

Rendern Sie erst die Skybox und danach die Modelle mit den entsprechenden Shaderprogrammen und Parametern!

Aufgabe 3: Billboarding/Raycasting

Ziel: Billboarding/Raycasting für Pixelgenaue Kugeln mit wenigen Vertices und Konzept des Raycasting anwenden. Erzeugen Sie eine Klasse Sphere mit folgenden Eigenschaften. Sphere besitzt ein Zentrum und einen Radius. Erzeugen Sie aus diesem Zentrum und Radius zwei Dreiecke (Siehe Skybox.h), die zusammen ein Quadrat bilden. Dieses Quadrat soll in der render-Funktion von Sphere an den Shader übergeben werden. Hinzu kommen zwei Uniforms, welche Radius und Zentrum an die Shader übergeben. Im Vertex-Shader müssen die Vertices so gedreht werden, dass die Normale stets in Richtung der Kamera zeigt.
Tutorial: http://www.opengl-tutorial.org/intermediate-tutorials/billboards-particles/billboards/

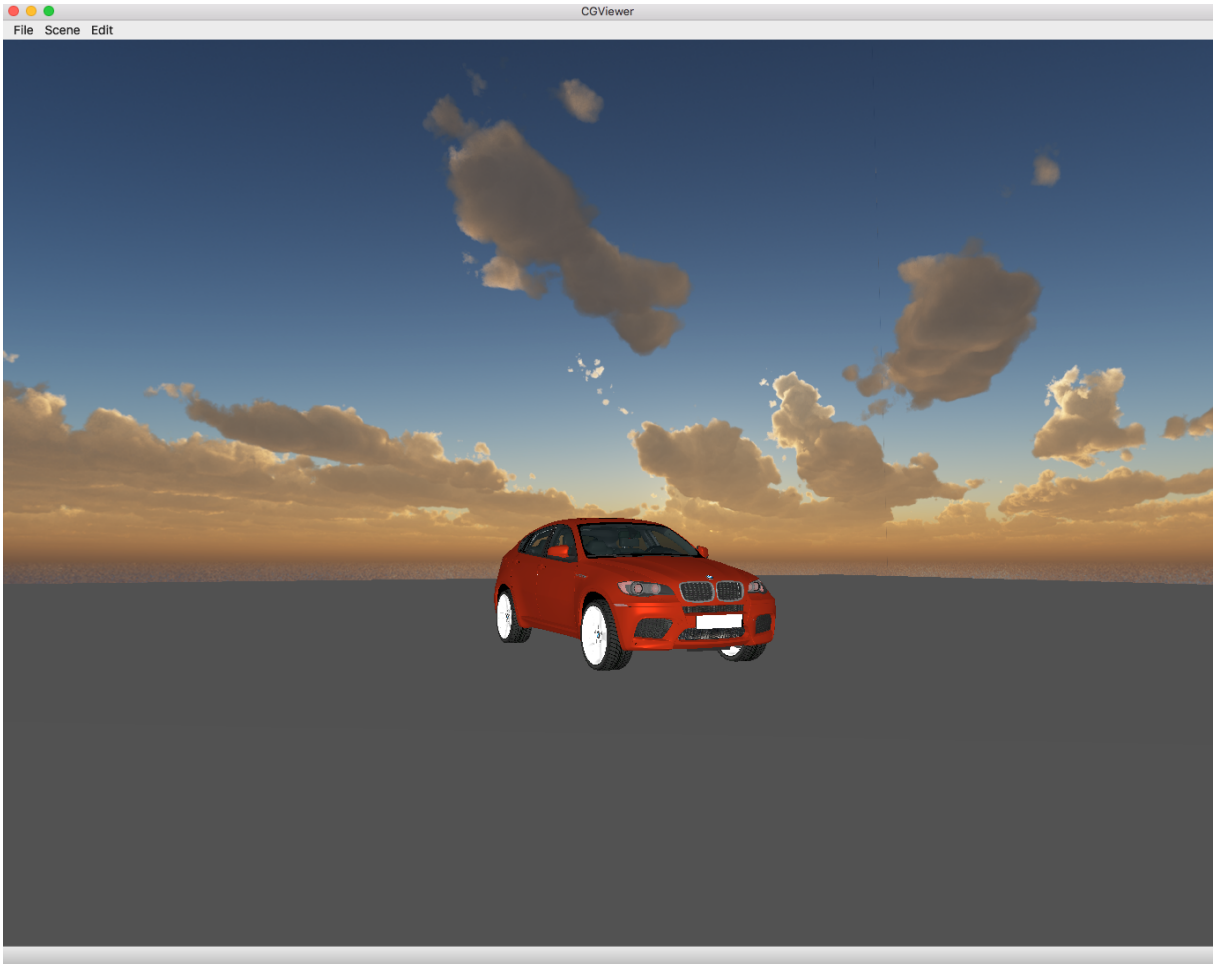
Rendern Sie eine Kugel beliebiger Größe an den world space Koordinaten (0,0,0).

Führen Sie alle Berechnungen im view space durch!

Für den Fall, dass der Strahl die Kugel verfehlt wird kein Farbwert gesetzt (GLSL hat hierfür das „discard"-keyword). Wird Sie getroffen, ist es möglich die genaue Position des Schnittpunktes zu erhalten und zusammen mit dem Zentrum eine Normale zu berechnen. Setzen Sie ein „einfaches" Phong-Shading für die Kugel mit einer konstanten Farbe um. Hierbei soll nur der diffusive Lichtanteil mit einer einzigen Lichtquelle (Kamerakoordinaten) gesetzt werden.

Zusatz: Setzen Sie mithilfe der getroffenen Position den korrekten gl_FragDepth (Tiefeninformation).

Hinweis: https://www.cs.princeton.edu/courses/archive/fall00/cs426/lectures/raycast/raycast.pdf (Folie 6 – Algebraische Methode)



Ergebnis nach den Aufgaben 1 und 2