Universität Leipzig
Institut für Informatik
Abteilung Bild- und Signalverarbeitung

Abgabe in der Woche des **6.Mai**

Computergrafik-Praktikum
Sommersemester 2019

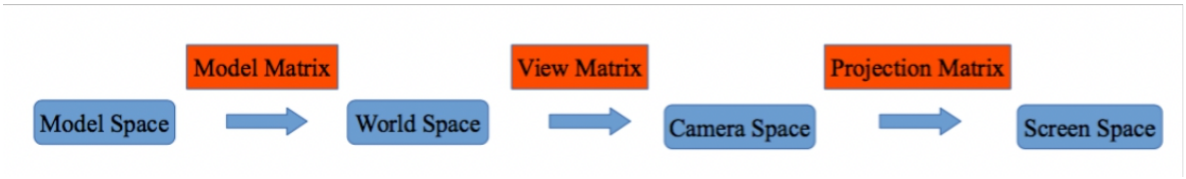
# Aufgabenblatt II – Transformationen, Selektion und Beleuchtung

## Aufgabe 0: Bereinigung

Entfernen Sie die rendering Funktionalität des Dreieckes aus dem Aufgabenblatt I. Kommentieren Sie in paintGL das gekennzeichnete Modellrendering wieder ein.

## Aufgabe 1: Transformationen

Die wichtigste Aufgabe des Vertexshader ist das Setzen der Position des Vertex. Diese Position muss im Screen Space gegeben sein, also komponentenweise Werte zwischen -1 und 1 annehmen. Ein Modell bzw. dessen Vertices sind aber immer im Model Space gegeben. Das sind in der Regel die Koordinaten, mit denen das Modell erstellt wurde. Wenn man nun ein Objekt in der Welt platziert bzw. bewegt, verändert man seine Koordinaten. Das sind die Weltkoordinaten (World Space). Wenn man sich in 3D eine Szene anschaut, geschieht dies immer durch eine Kamera. Dazu wird die gesamte Szene (also alle Modelle) vor die Kamera gedreht. Die Kamera im Camera Space im Koordinatenursprung. Schlussendlich müssen alle Modelle von der 3D-Welt auf unser 2D-Ausgabemedium (den Monitor) projiziert werden (es gibt verschiedene Projektionsarten, die in der Vorlesung erklärt werden). Jetzt haben wir die Modelle im Screen Space gegeben. Die Transformation zwischen diesen ganzen Koordinatensystemen erfolgt mittels 4-dimensionalen Matrizen:




Das Ziel dieser Aufgabe soll es sein, diese „Pipeline“ zu implementieren. Die Matrizen sind im CGViewer vorberechnet. Details zu ihrer Erstellung werden in der Vorlesung und Übung behandelt. Darüber hinaus kümmert sich der CGViewer um das komplette Setup der Shader und der Vertexattribute. Ihre Aufgabe ist es, die Matrizen an das Shaderprogramm zu übergeben und den gegebenen Vertexshader (unter *shader/vertex.glsl*) so zu ergänzen, dass die korrekte gl_Position ausgerechnet wird. Die Vertexpositionen werden über das Vertexattribut „position“ an den Vertexshader bergeben. Die anderen beiden Vertexattribute „normal“ und „texCoords“ werden noch nicht benötigt. Die Modellmatrix für jedes einzelne Modell bekommen Sie über die Funktion *getTransformations()* der Modellklasse. Die View Matrix ist in der Scene-Klasse durch „m_view“ gegeben. Sie wird in der Funktion *setTransformations()* bei jedem Renderdurchgang neu aufgebaut. Ebenso ist die Projektionsmatrix ein Attribut der Scene- Klasse („m_projection“). Sie wird in der Funktion resizeGL(int width, int height) immer dann neu aufgebaut, wenn sich die Auflösung des Ausgabefensters ändert. Der CGViewer verwendet eine perspektivische Projektion mit einem

Öffnungswinkel von 60°. Ergänzen Sie daraufhin den Fragmentshader so, dass eine konstante Farbe ausgegeben wird. Wenn Sie alles korrekt implementiert haben, sollten Sie die Silhouetten der geladenen Modelle sehen können, und mit der Maus die Kameraposition bzw. Rotation verändern können.

**Aufgabe 2: Selektion von Objekten**

Im CGViewer können die geladenen Modelle frei in der Szene bewegt werden. Dazu wird ein Modell mit einem Doppelklick ausgewählt (siehe mouseDoubleClickEvent-Funktion in der Scene-Klasse). Wenn das Modell ausgewählt wurde, können verschiedene Transformationen (Verschiebung, Skalierung, Rotation) auf das Objekt angewandt werden. Dass ein Objekt ausgewählt wurde, muss dem Benutzer aber auch visuell mitgeteilt werden.

Zurzeit besteht das gegebene Shaderprogram („m_program“ in der Scene-Klasse) aus dem in Aufgabe 1 erweiterten Vertexshader sowie einem simplen Fragmentshader, der einfach alle Pixel, welche ein Modell darstellen, konstant einfärbt. Das Ziel dieser Aufgabe ist es, einen zweiten Fragmentshader zu schreiben, der den Pixeln einfach eine andere (ebenfalls konstante) Farbe zuweist, wenn der Nutzer das entsprechende Modell ausgewählt hat.

Legen Sie dazu einen weiteren Pointer (std::shared_ptr) auf ein QOpenGLShaderProgram in der Szene-Klasse an (siehe „m_program“). Die Initialisierung des Programms erfolgt in der Funktion reloadShader(). Diese Funktion wird immer dann aufgerufen, sobald der Nutzer im Menü Edit → Reload Shader (oder kurz: CTRL+R) auswählt (sowie beim Starten des CGViewers). Sehen Sie sich an, wie das Shaderprogram „m_program“ geladen wird und implementieren Sie die entsprechende Initialisierung für Ihren neuen Shader. Hinweis: das neue Shaderprogramm unterscheidet sich von dem gegebenen Shaderprogramm nur durch den Fragmentshader. Der Vertexshader sollte der gleiche sein. Um zu überprüfen, ob ein Modell ausgewählt wurde, überprüfen Sie beim render-Vorgang das Attribut „selectedModel“ der Szene- Klasse und wählen dann das entsprechende Shaderprogramm aus. In „selectedModel“ ist die Position des selektierten Models im models-vector gespeichert. Ist kein Model selektiert, hat „selectedModel“ den Wert -1. Wenn Sie alles korrekt implementiert haben, sollten Sie durch einen Doppelklick auf ein Modell dessen Farbe ändern können. Ein weiterer Doppelklick sollte das Modell wieder deselektieren.

**Aufgabe 3: Schattierung und Beleuchtung**

Jetzt, wo der Vertexshader dafür sorgt, dass die Modell-Vertices korrekt angezeigt werden, geht es in dieser Aufgabe darum, die Modelle 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 zunächst eine Standartlichtquelle. Diese soll die Position der Kamera haben und die drei Lichtanteile sollen jeweils auf [1.0, 1.0, 1.0] gesetzt werden. Dazu führen Sie die Möglichkeit ein, bis zu 3 Lichtquellen in der Szene nutzen zu können. Hierzu eignet sich die Funktion *setUniformValueArray* der *QOpenGLShaderProgram*-Klasse. Berechnen Sie die Lichtquellenposition über die Methode getBoundingBox() (center) der Klasse Modell/Licht. 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 cos(Winkel(vektor1, vector2)) das Skalarprorodukt ( vector1, vector2 ) berechnen können, wenn die Vektoren normalisiert sind. Lichtquellen werden automatisch als Kugeln im Zentrum gerendert.

Beispiel Ergebnisse nach Aufgabe 1 (oben links) nach Aufg. 3 (rechts – Phong shading mit Materialinformationen)