Technische Hochschule Köln
Insitut für Informatik
Prof Dr.-Ing. Martin Eisemann

Technology
Arts Sciences
TH Köln

27.07.2018

Computergrafik und Animation
Klausur

Vor- und Nachname: _____

Matrikelnummer: _____

Studiengang, Semester: _____

Unterschrift: _____

Bitte lesen Sie die Anweisungen auf der nächsten Seite sorgfältig durch!
Dauer der Klausur: 120 Minuten
(Die folgende Tabelle wird bei der Korrektur ausgefüllt)

Thema	Punkte	Erreicht
Transformationen	25	
Rendering	21	
Rendering-Pipeline	19	
Programmierung mit GLSL	21	
Shadow Maps	14	
Summe	**100**	

Note:

Lesen sie die Anweisungen sorgfältig durch!

- Überprüfen Sie, ob Sie alle Seiten bekommen haben (13 Seiten, 5 Aufgaben)
- Füllen Sie das Deckblatt aus.
- Es sind keine Hilfsmittel außer Stiften (kein Rotstift) und Lineal/Geodreieck in der Klausur erlaubt (kein Taschenrechner, keine Bücher, keine Handys, keine Smartwatches).
- Besonders das Fotographieren der Klausurbögen ist verboten!
- Schreiben Sie die Lösung der Aufgabe auf die Aufgabenblätter oder fordern Sie ein separates Blatt an. Sie dürfen die Rückseiten mitverwenden. Wenn Sie ein separates Blatt benutzen, so darf *pro Blatt nur eine Aufgabe* abgegeben werden. Fangen Sie also für jede Aufgabe ein neues Blatt an (gilt nicht für Teilaufgaben). Vermerken Sie auf dem Aufgabenblatt, dass Sie ein separates Blatt verwendet haben.
- Schreiben Sie auf *jedes* Blatt, das Sie abgeben, Ihre Matrikelnummer!
- Ordentlich und leserlich schreiben. Was wir nicht entziffern können, können wir auch nicht bewerten.
- Sie können die Aufgaben in Deutsch oder Englisch beantworten, aber bleiben Sie bei Ihrer Wahl.

Viel Erfolg!

Matrikelnummer

Aufgabe 1 Transformationen (25 Punkte):

a) Stellen Sie eine 4×4-Transformationsmatrix auf, um ein Objekt im 3D-Raum um den Vektor $\begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix}$ zu verschieben. (2 Punkte)

Lösung:

$$\begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

b) Stellen Sie eine 4×4-Transformationsmatrix auf, mit welcher ein Objekt im 3D-Raum um $90°$ um die Y-Achse rotiert werden kann. (2 Punkte)

Lösung:

$$\begin{pmatrix} \cos(90°) & 0 & -\sin(90°) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(90°) & 0 & \cos(90°) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

c) Stellen Sie eine 4×4-Transformationsmatrix auf, mit welcher ein Objekt im 3D-Raum gleichförmig um 50% kleiner skaliert werden kann. (2 Punkte)

Lösung:

$$\begin{pmatrix} 0,5 & 0 & 0 & 0 \\ 0 & 0,5 & 0 & 0 \\ 0 & 0 & 0,5 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

d) Berechnen Sie ein orthonormales Koordinatensystem mit den Achsen U,V und W auf der Basis von den zwei Vektoren $\vec{a} = \begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix}$ und $\vec{b} = \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix}$ und zusätzlich unter den folgenden Bedingungen:

- U ist parallel/kollinear zu $\vec{a}$
- V liegt in der Ebene, welche durch $\vec{a}$ und $\vec{b}$ aufgespannt wird

Hinweis: Komplexe Rechenoperationen, wie Brüche, Wurzeln o.ä. können als solche stehen bleiben. (9 Punkte)

Lösung:

i) $U = a$ (2P) $\rightarrow \begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix}$

ii) $W = U \times b$ (3P $\times$-Produkt) $\rightarrow \begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix} \times \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \\ -1 \end{pmatrix}$

iii) $V = U \times W$ (3P $\times$-Produkt) $\rightarrow \begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix} \times \begin{pmatrix} 2 \\ 0 \\ -1 \end{pmatrix} = \begin{pmatrix} -3 \\ 10 \\ -6 \end{pmatrix}$

Final müssen alle Vektoren noch normalisiert werden. (1P für Normierung)

$\vec{U} = \frac{1}{\sqrt{2^2+3^2+4^2}} * \begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix} = \frac{1}{\sqrt{29}} * \begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix}$

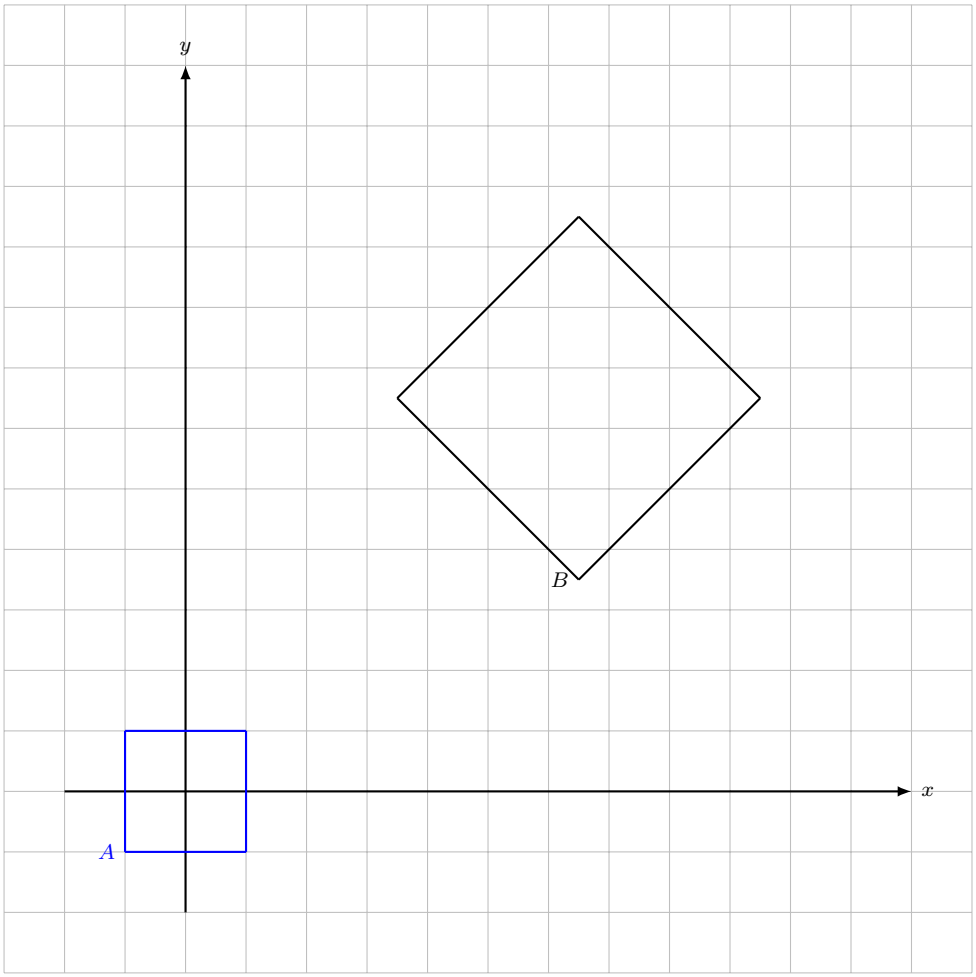
$\vec{V} = \frac{1}{\sqrt{(-3)^2+10^2+(-6)^2}} * \begin{pmatrix} -3 \\ 10 \\ -6 \end{pmatrix} = \frac{1}{\sqrt{145}} * \begin{pmatrix} -3 \\ 10 \\ -6 \end{pmatrix}$

$\vec{W} = \frac{1}{\sqrt{2^2+0^2+(-1)^2}} * \begin{pmatrix} 2 \\ 0 \\ -1 \end{pmatrix} = \frac{1}{\sqrt{5}} * \begin{pmatrix} 2 \\ 0 \\ -1 \end{pmatrix}$

e) Tranformationen sind selten eindeutig. Geben Sie zwei zusammengesetzte Transformationen inkl. Multiplikationsreihenfolge an, die das unten abgebildete Objekt aus dem Ursprung (Quadrat A) in die definierte Zielposition (Quadrat B) transformiert. Sie können hierfür die Transformationen wie folgt definieren:

- Translationen entlang der x-, y- oder z-Achse: T_x, T_y oder T_z
- Rotationen um die x-, y- oder z-Achse: R_x, R_y oder R_z und
- Skalierungen entlang der x-, y- oder z-Achse: S_x, S_y oder S_z

Wichtig ist die Art der Transformation und richtige Angabe der Reihenfolge. Die Angabe der genauen Werte ist nicht erforderlich. (10 Punkte)



Lösung:

i) $R_z * T_x * S_x * S_y$ (5P)

ii) $T_x * T_y * R_z * S_x * S_y$ (5P)

iii) oder andere passende Transformationen

Matrikelnummer

Aufgabe 2 Rendering (21 Punkte):

a) Einige Bildverarbeitungsprogramme definieren den RGB-Farbraum im Wertebereich 0 (dunkelster Wert) bis 255 (hellster Wert) für jeden Farbkanal. Welche vec4-Repräsentation hat die Farbe RGB = (51, 0, 204) im OpenGL-Fragment Shader, wenn es sich um ein undurchsichtiges Objekt handelt? (2 Punkte)

Lösung:

`vec4 c = vec4(0.2, 0.0, 0.8, 1.0)` (0,5P pro Komponente RGBA)

b) Gegeben sei folgendes VertexBufferObject (VBO)

Ein VBO sieht hierbei wie folgt aus:

$$VBO = [\underbrace{\text{Position, Farbe, Normale, Textur, Shininess}}_{\text{Vertex1}}, \underbrace{\text{Position, Farbe, Normale, Textur, Shininess}}_{\text{Vertex2}}, ...]$$

mit den Vertex-Werten für Position, RGB-Farbe, Normale, Textur und Shininess. Die Werte sind im interleaved Modus abgelegt, d.h. es werden erst alle Werte des ersten Vertices beschrieben, bevor es zum zweiten Vertex weitergeht.

i) Aus wievielen Float-Werten besteht ein so definierter Vertex? Geben Sie für jedes Attribut die jeweilige Anzahl an. (5 Punkte)

Lösung:

- 3x Position (1P)
- 3x RGB-Farbe (1P)
- 3x Normale (1P)
- 2x Textur (1P)
- 1x Shininess (1P)

ii) Berechnen Sie Stride und Offset für die Attribute Farbe und Shininess in Byte an. Bei den einzelnen Daten werden Floats mit jeweils 4 Byte verwendet. (4 Punkte)

Lösung:

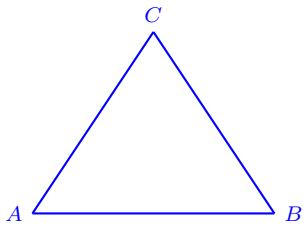
- Stride: (3(Pos) + 3(Farbe) + 3(Normale) + 2(Textur) + 1(Shininess)) * 4(Byte-Größe) = 48 (2P)
- Offset Farbe: 3 * 4 = 12 (1P)
- Offset Shininess: (3 + 3 + 3 + 2) * 4 = 44 (1P)

c) Worin liegt der Vorteil der Nutzung eines IndexBufferObject (IBO)? Begründen Sie Ihre Antwort in Bezugnahme des VBOs, welches in Aufgabe 2b) definiert wurde. (5 Punkte)

Lösung:

Man spart Speicherplatz (2P), da man Vertices mehrfach nutzen kann und 1 Vertex deutlich mehr Speicherplatz benötigt, als 1 Speicher im IBO.(3P)

d) Sei ein Dreieick in einem IBO definiert als D = (A, B, C). Sobald Sie die Definition in D = (A, C, B) ändern, fällt auf, dass das Dreieck nicht mehr gerendert wird. Bitte erklären Sie begründet, wodurch dies zustande kommt. (5 Punkte)



Lösung:

Die Orientierung des Dreiecks ändert sich, sobald die Definition statt counterclockwise clockwise stattfindet. Entsprechend dreht sich die Flächennormale um und zeigt in die entgegengesetzte Richtung, was zur Folge hat, dass das Backfaceculling dieses Dreieck als verdeckt erkennt und nicht mehr rendert.

Matrikelnummer

Aufgabe 3 Rendering Pipeline (19 Punkte):

HINWEIS
Für die Teilaufgaben a) - c) nutzen Sie bitte das Schaubild auf der nachfolgenden Seite. Die Grafik startet bei dem Eintritt in die Rendering Pipeline.

a) Ordnen Sie die folgenden Spaces entsprechend ihrer Reihenfolge in der Rendering Pipeline in den Rechtecken des Diagramms auf der nachfolgenden Seite ein. (5 Punkte)

A - Normalized Device Space
B - Object Space
C - Camera Space
D - World Space
E - Screen Space
F - Clipping Space

Lösung:

B - D - C - F - A - E (1P pro richtiger Positionsfolge)

b) Bitte markieren Sie in dem Schaubild den bzw. die Arbeitsbereiche des Vertex- und des Fragment-Shaders. (5 Punkte)

Lösung:

Vertex-Shader = {B, D, C, F}
Fragment-Shader = {E} (1P pro richtiger Einordnung)

c) Beschreiben Sie kurz und prägnant die Aufgaben der unten genannten Matrizen. Ordnen Sie diese Matrizen im Anschluss den Kreisen des Schaubildes zu, wo Sie Ihrer Meinung nach Verwendung finden. Streichen Sie die nicht verwendeten Kreise bitte durch. (9 Punkte)

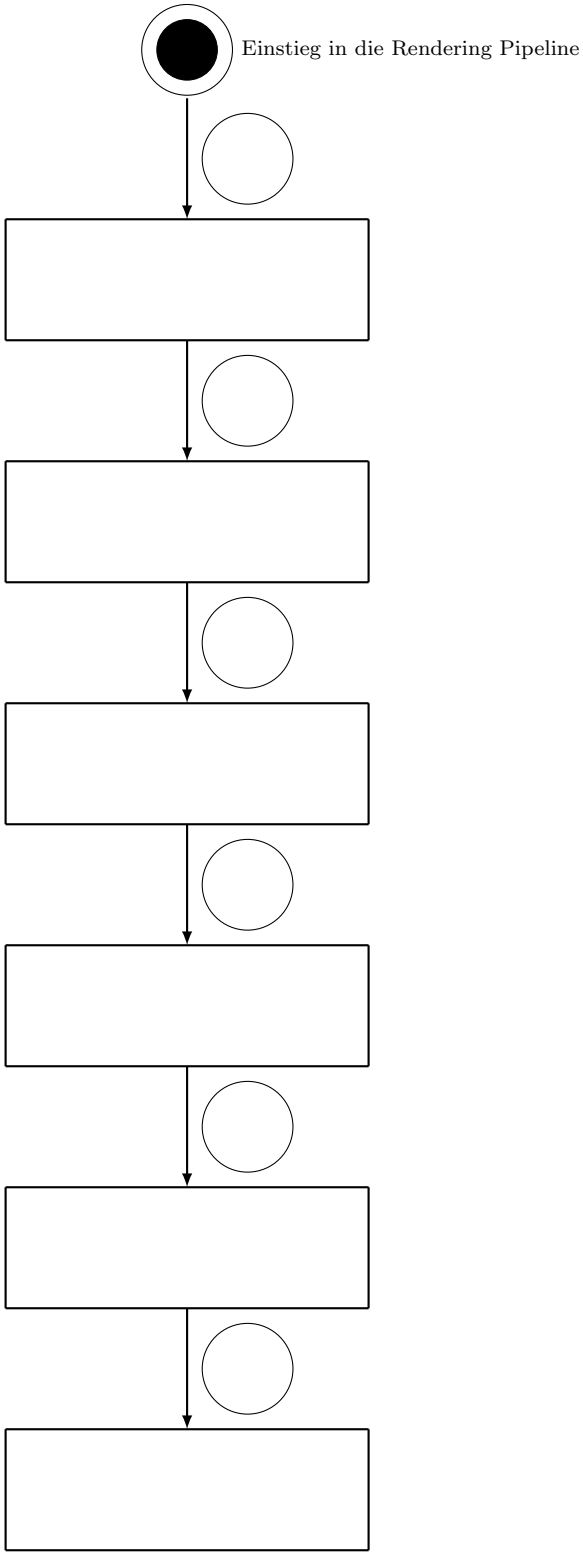
1 - View-Matrix

2 - Model-Matrix

3 - Projection-Matrix

Lösung:

Model-M: Object-to-World; View-M: World-to-Camera; Projection-M: Camera-to-NormalizedDevice
1P pro richtiger Zuordnung + 2P pro richtiger Beschreibung



Aufgabe 3) Begriffe zur Erinnerung:

a) **Spaces:**

A - Normalized Device Space
B - Object Space
C - Camera Space
D - World Space
E - Screen Space
F - Clipping Space

b) **Arbeitsbereiche:**
Vertex- und Fragment-Shader

c) **Matrizenzuordnung:**

1 - View-Matrix
2 - Model-Matrix
3 - Projection-Matrix

Aufgabe 4 Programmierung GLSL (21 Punkte):

a) Der folgende Programmcode gibt ein kurzes Beispiel für die Kommunikation zwischen Java-Programm, dem Vertex- und dem Fragmentshader. Leider sind uns jedoch vier Fehler unterlaufen. Bitte identifizieren Sie sämtliche Probleme und schreiben Sie den benötigten Quellcode daneben. Sollten Sie den Fehler detektiert haben aber nicht wissen, wie der korrigierte Code aussehen muss, dann umschreiben Sie den Fehler und dessen Lösung stattdessen. (8 Punkte)

Java-Code:

```
...
float mixture = 1.0f;

shader.setUniform("param", mixture);
...
```

Vertex-Shader:

```
//Objektgeometrie liegt zwischen (0, 0, 0) und (1, 1, 1)
layout(location = -1) in  vec3 pos;

out vec3 position; // soll die Vertex Position separat speichern

main(){

    gl_Position = pos;

}
```

Fragment-Shader:

```
in vec3 position;

uniform float mixture;

main(){

    vec3 c2 = vec3(position.x);

    color = vec4(mix(position, c2, mixture), 1.0f);

}
```

Lösung:

float mixture =1.0f; //simulierter Boolean, entweder 1 oder 0
shader.setUniform("**mixture**", mixture);

Vertex-Shader:

// 0 oder beliebiger anderer Wert layout(**location = 0**) in vec3 pos; //Objektgeometrie liegt zwischen (0/0/0) und (1/1/1)
out vec3 position;
main(){
gl_Position = pos;
position = pos;
}

Fragment-Shader:

in vec3 position;
uniform float bool;
out vec4 color;
main(){
vec3 c2 = vec3(position.x);
color = vec4(mix(position, c2, mixture), 1.0f);
}

(2P pro gefundenem Fehler)

b) Jetzt, wo Sie alle Fehler gefunden und erfolgreich behoben haben, sollte das Programm endlich wieder laufen. Beschreiben Sie kurz, was Sie als (graphische) Ausgabe des Programmes erwarten. Was wäre die Ausgabe, wenn mixture = 0.0f gelten würde? (2 Punkte)

Lösung:

Die Position der Vertices wird als Farbinformation verwendet. Durch die übergebene Uniform *mixture* kann zwischen zwei Farbverläufen hin- und her geschaltet werden. Bei mixture = 0.0f verläuft der Farbverlauf diagonal (x-/y-/z-Richtung), bei mixture = 1.0f ist der Farbverlauf horizontal, da er abhängig von der x-Position ist.

c) Wann und warum sollten berechnungsintensive Schritte möglichst im Vertex- und nicht im Fragment-Shader durchgeführt werden? In welchem Fall ist dies nicht sinnvoll? (3 Punkte)

Lösung:

In der Regel wird der Fragment-Shader sehr viel häufiger als der Vertex-Shader aufgerufen, weswegen die berechnungsintensiven Schritte möglichst im Vertex-Shader stattfinden sollten. Unsinnig, wenn mehr Vertices als Pixel/Fragments vorhanden sind.

d) Welche Aufgaben werden standardmäßig im Vertex- und/oder im Fagment-Shader ausgeführt? Kreuzen Sie die richtigen Antworten in der unten stehenden Tabelle an. (8 Punkte)

	Vertex-Shader	Fragment-Shader
Objektplatzierung		
Auslesung der Textur		
Farbberechnung		
Zugriff auf das VAO und VBO		
Entgegennahme von Uniform-Variablen		
Lichtberechnung (inkl. der nötigen Vektoren)		

Lösung:

- Objektplazierung: VS
- Auslesung der Textur: FS [+ VS (eher nicht standardmäßig)]
- Farbberechnung: FS
- Zugriff auf das VAO: VS
- Entgegennahme von Uniform-Variablen: VS + FS
- Lichtberechnung (inkl. der nötigen Vektoren): VS + FS

Matrikelnummer

Aufgabe 5 Shadow Maps (14 Punkte):

a) Bringen Sie diesen Pseudocode für die Schattenberechnung mit Shadow Maps in die richtige Reihenfolge (1-5). (10 Punkte)

___ Vergleiche Distanz zwischen Punkt und Lichtquelle mit den Werten aus der Shadow-Map

___ Speichere die Tiefenwerte in eine Textur

___ Render die Szene aus Sicht der Lichtquelle

___ Bei annähernd gleichen Tiefenwerten beleuchte den Punkt, bei niedrigerem Tiefenwerten nicht.

___ Render aus Sicht der Kamera

Lösung:

4. - 2. - 1. - 5. - 3.
Hierbei kann es zu Folgefehlern kommen. Evt. auch hier die korrekten Abfolgen bewerten

b) Die Auflösung einer Shadow-Map für eine gegebene Lichtquelle ist normalerweise konstant. Die gewünschte projizierte Fläche im Bild eines beleuchteten Objekts hingegen ist direkt abhängig von dessen Entfernung zur Kamera und zu seiner Größe. Welche visuellen Probleme sind beim Rendern zu erwarten, wenn sich die Kamera immer weiter in Richtung Objekt bewegt? (4 Punkte)

Lösung:

Je weiter die Kamera an die Objekte herankommt, desto grob-pixeliger werden die Schatten dargestellt. (Aufteilung der Punkte noch unklar)