Karlsruher Institut für Technologie
Lehrstuhl für Computergrafik

Prof. Dr.-Ing. Carsten Dachsbacher
M. Sc. Alisa Jung
M. Sc. Reiner Dolp

5. Übungsblatt zur Vorlesung Computergrafik im WS 2021/2022

Abgabe bis **Montag, 23.01.2022**, 07:59 Uhr

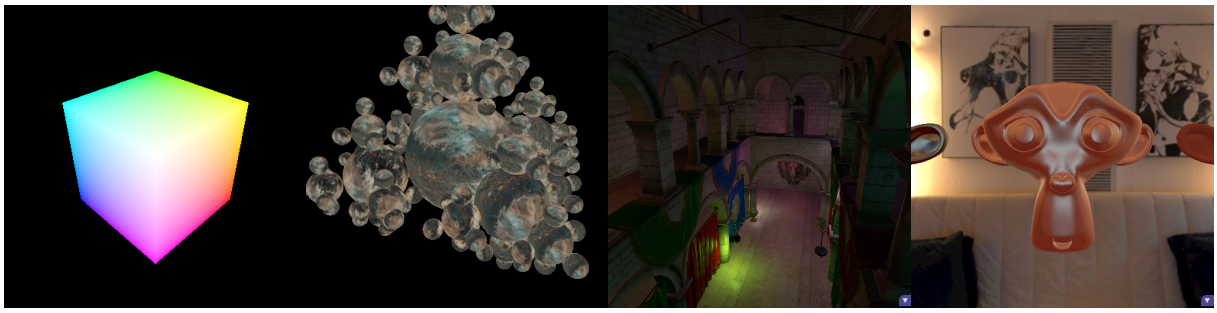


Abbildung 1: Ergebnisbilder des Übungsblattes.

Dieses Übungsblatt beschäftigt sich mit Szenengraphen und modernem OpenGL. Sie müssen in diesem Übungsblatt OpenGL-Shader in GLSL implementieren. Eine ausführliche Dokumentation von OpenGL und GLSL (3.x) finden Sie z.B. unter

`http://docs.gl`,
`https://www.opengl.org/registry/doc/GLSLangSpec.3.30.6.clean.pdf` und
`https://www.opengl.org/sdk/docs/man`.

Die Ergebnisbilder einer kompletten Implementierung sind in Abbildung 1 dargestellt. Lesen Sie bitte das Übungsblatt sorgfältig und machen Sie sich mit dem Code des Frameworks vertraut.

Hinweis: Während der Bearbeitung des Übungsblattes wird bei Ihnen wahrscheinlich die Warnung *Warning: uniform "xyz" not found in shader!* ausgegeben werden. Dies ist normal und tritt auf, wenn eine Uniform-Variable im Shader nicht benutzt wird oder vom GLSL-Compiler weg optimiert wurde. Wenn Sie mit der Implementierung fertig sind, sollten diese Warnungen nicht mehr auftreten.

Hinweis: Wenn Sie Shader implementieren, müssen Sie das Programm nicht nach jeder Änderung im Shader neu starten. Es genügt in der GUI den Button "Reload all shaders" zu drücken.

Hinweis: Sämtliche Winkel werden in der `glm`-Version des Frameworks in Bogenmaß angegeben.

Fehlerbehebung unter Windows: Fehler in der Originalversion von Microsoft Visual C++ 2013 führen zu fehlerhaften ausführbaren Dateien, welche das Bearbeiten einiger Aufgaben unmöglich machen. Glücklicherweise beheben die offiziellen Updates diese Fehler. Stellen Sie sicher, dass Sie mit der aktualisierten Version arbeiten (Update 4) und aktualisieren Sie ihr Visual Studio ggf. über `Tools` → `Extensions and Updates`, dort in der Sidebar `Updates` → `Product Updates`, wählen Sie `Visual Studio 2013 Update 4`.

1 *OpenGL-Shader*	**4 Punkte**

In dieser Aufgabe sollen Sie einen sehr einfachen OpenGL-Shader vervollständigen um einen RGB-Würfel zu zeichnen. Der Vertex-Shader `simple.vert` erhält als Eingabe `POSITION` die Position des jeweiligen Vertex des Würfels in Objektkoordinaten. Die Model-View-Projection Matrix ist durch die Eingabevariable `MVP` gegeben. Füllen Sie die OpenGL-Variable `gl_Position` mit der in den Clip-Space transformierten Position. Außerdem sollen Sie eine Vertexfarbe berechnen, die die Position $p \in [-1,1]^3$ (in Objekt-Koordinaten) auf eine Farbe $c \in [0,1]^3$ mit

$$c = 0.5 \cdot p + (0.5, 0.5, 0.5)$$

abbildet. Schreiben Sie die Vertexfarbe in die Ausgabevariable `color`.

Der Code gibt die Eingabevariable `TEXCOORD` unverändert an den Fragment-Shader weiter, indem deren Inhalt in die Ausgabevariable `texcoord` geschrieben wird. Verändern Sie `texcoord` nicht! Die Texturkoordinaten werden in der nächsten Aufgabe benötigt. Der Fragment-Shader `simple.frag` gibt die interpolierten Vertexfarben einfach als Ausgabefarbe weiter und muss nicht verändert werden.

2 *Szenengraph*	**6 Punkte**

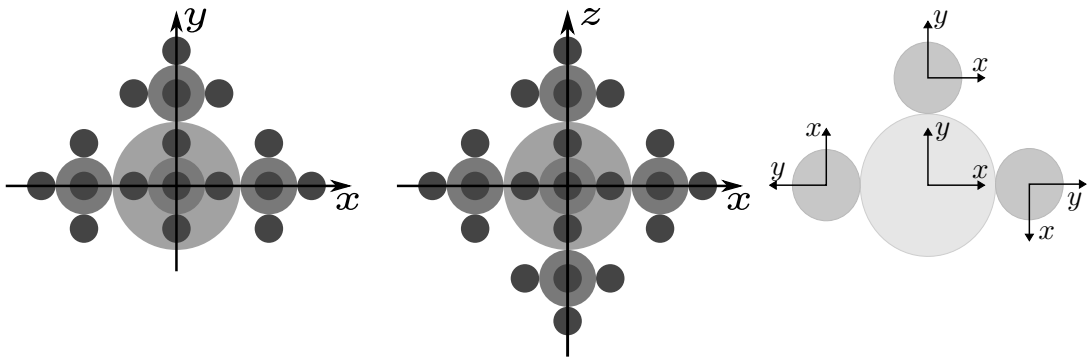


Abbildung 2: Illustration des rekursiven Aufbaus der Kugelflocke. Links: Frontsicht. Mitte: Draufsicht. Rechts: Illustration der lokalen Koordinatensysteme für 3 Kinder.

In dieser Aufgabe sollen Sie einen einfachen zeitabhängigen Szenengraphen implementieren und damit die Kugelflocke, die in Abbildung 1 zu sehen ist, modellieren.

1. Konstruieren Sie in `buildSphereFlakeSceneGraph` in der Datei `exercise_06.cpp` die in Abbildung 2 illustrierte rekursive Kugelflocke. Die Funktion soll einen Knoten vom Typ `SceneGraphNode` (definiert in `cglib/include/cglib/gl/scene_graph.h`) erstellen und dessen Variable `model` auf das als Parameter gegebene `GLObjModel` setzen.

   Zudem sollen Sie rekursiv fünf Kindknoten erzeugen. Berechnen Sie die für jeden Kindknoten die Transformationsmatrizen `node_to_parent` und `parent_to_node` und fügen Sie die Kindknoten zur Kindknotenliste `children` hinzu. Der Kugelradius soll, wie in der Abbildung gezeigt, in jeder Rekursion relativ um den Faktor `size_factor` kleiner werden. Die lokale y-Achse eines Kindknoten soll vom Parent-Knoten weg zeigen. Die Rekursion soll abgebrochen werden, falls `number_of_remaining_recursions` ≤ 0 ist. Sie können für die Berechnung der inversen Matrix und zur Erzeugung von Rotations-, Translations- und Skalierungsmatrizen GLM verwenden (`http://glm.g-truc.net/0.9.0/api/a00192.html`).

2. In der Funktion `collectTransformedModels` soll der Szenengraph rekursiv traversiert werden um die Objekt-zu-Welt Transformationen für jeden Knoten zu berechnen. Beim traversieren eines Knotens soll ein Objekt von Typ `TransformedModel` (definiert in der Datei `cglib/include/cglib/gl/scene_graph.h`) erstellt und der Liste `transformed_models` hinzugefügt werden.

3. Um eine zeitabhängige Animation zu realisieren soll in der Funktion `animateSphereFlake` rekursiv jeder Knoten des Szenengraph um seine lokale y-Achse rotiert werden. Als inkrementellen Rotationswinkel soll der Parameter `angle_increment` verwendet werden. Beachten Sie, dass sich die Inverse der Rotationsmatrix sehr effizient durch einfache Transposition (z.B. `glm::transpose`) berechnen lässt. Zur Laufzeit muss somit keine aufwendige Invertierung der Transformationsmatrix vorgenommen werden, um `parent_to_node` zu aktualisieren.

3 *Phong-Shading* — 6 Punkte

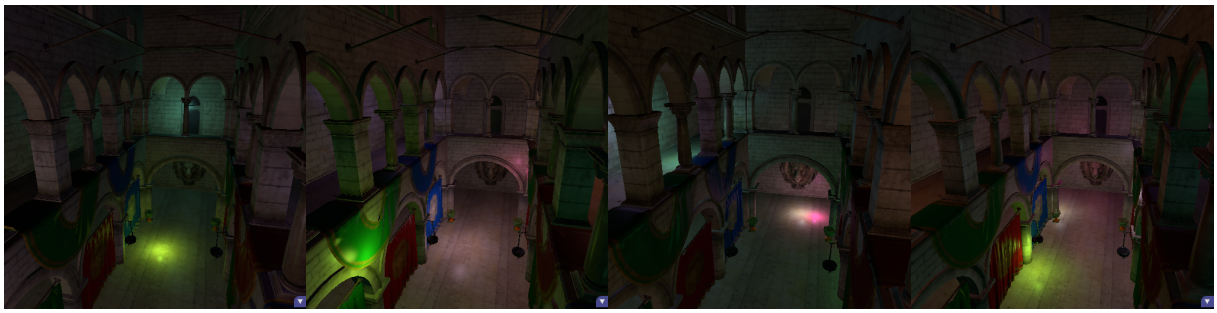


Abbildung 3: Die Sponza-Animation bei 0, 9, 20.5 und 28.5 Sekunden.

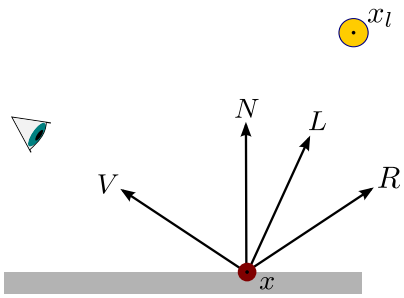


Abbildung 4: Lage der Vektoren für das Phong-Beleuchtungsmodell.

In dieser Aufgabe sollen Sie Phong-Shading, d.h. Beleuchtungsberechnung mit interpolierten Normalen im Fragment-Shader, mit dem Phong-Beleuchtungsmodell implementieren.

1. Implementieren Sie dafür zuerst den Vertex-Shader `phong.vert`. Hier sollen Sie, wie in der ersten Aufgabe, die OpenGL-Variable `gl_Position` füllen und zudem die Position und Normale des Vertex in Weltkoordinaten berechnen. Die Texturkoordinaten sollen einfach weitergereicht werden.

2. Implementieren Sie nun das Phong-Beleuchtungsmodell im Fragment-Shader `phong.frag`. Akkumulieren Sie die Beleuchtung aller Punktlichtquellen. Der Materialparameter `k_d` soll aus einer Textur ausgelesen werden. Verwenden Sie dazu die GLSL-Funktion `texture`, die als Parameter einen Textursampler und Texturkoordinaten erhält. Zur Berechnung des Reflexionsvektors können Sie die GLSL-Funktion `reflect` verwenden. Dokumentation zu `texture` und `reflect` finden Sie unter `https://www.opengl.org/sdk/docs/man/html/texture.xhtml` bzw. `https://www.opengl.org/sdk/docs/man/html/reflect.xhtml`. Die Beleuchtung durch eine Lichtquelle soll anhand der Formel

$$(k_d \max(0, L \cdot N) + k_s \max(0, R \cdot L)^n) \frac{I_l}{\|x - x_l\|^2}$$

erfolgen. Die Terme der Formel sind in Abbildung 4 illustriert. Beachten Sie, dass alle Richtungsvektoren bei der Berechnung normalisiert sein müssen. Skalarprodukte ($L \cdot N$, $R \cdot L$) können Sie mit der Funktion `dot` berechnen. Die Multiplikation der relativen Beleuchtung (linke Klammer) an I_l erfolgt komponentenweise.

4 *Environment Lighting* **4 Punkte**

Abbildung 5: Environment Lighting mit unterschiedlichen Materialparametern.

In dieser Aufgabe implementieren Sie Image-Based Lighting mit vorgefilterten Environment Maps. Das Ergebnis sehen Sie in Abbildung 5. Die Auflösung der Environment Map sei $[W, H]$. Die Ergebnisse der Vorfilterung werden bei Programmstart ins Hauptverzeichnis gespeichert.

1. Konstruieren Sie zuerst die vorgefilterte Environment Map für diffuse Beleuchtung in der Funktion `prefilter_environment_diffuse`. Die vorgefilterte Environment Map soll in jedem Texel die vorgefilterte diffuse Beleuchtung speichern für die Normale, deren Richtung durch dessen Texturkoordinaten repräsentiert wird. Berechnen Sie dazu für jede mögliche Normale $N \in \Omega = \{x \in \mathbb{R}^3 : \|x\| = 1\}$, die durch einen Texel der vorgefilterten Environment Map repräsentiert wird, das Beleuchtungsintegral

$$\begin{aligned} L_d(N) &= \int_\Omega L_{in}(\omega_i) \max(0, N \cdot \omega_i)\, d\omega_i \\ &\approx \sum_{x=0}^{\text{W-1}} \sum_{y=0}^{\text{H-1}} L_{in}(r(x,y))\ \max(0, N \cdot r(x,y))\, s(x,y). \end{aligned}$$

   Die Werte für $L_{in}(\omega)$ sollen aus der Environment Map ausgelesen werden. Die Abbildung r bildet Texturkoordinaten der Environment Map auf Richtungen ab. Die Funktion s gibt den Raumwinkel eines Texels an und ist gegeben durch `solid_angle_from_lonlat_coord` (`cglib/include/cglib/gl/prefilter_envmap.h`). Der Raumwinkel des Texels entspricht ungefähr der Größe von $d\omega_i$. Nutzen Sie die Funktion `direction_from_lonlat_coord` (`cglib/include/cglib/gl/prefilter_envmap.h`), um Texturkoordinaten (sowohl der originalen als auch der vorgefilterten Environment Map) in Richtungen zu transformieren. Die Skalarprodukte $N \cdot \omega_i$ und $R \cdot \omega_i$ können Sie beispielsweise mit der Funktion `glm::dot` berechnen.

2. In der Funktion `prefilter_environment_specular` sollen Sie nun für einen gegebenen Phong-Exponenten n eine Environment Map für den spekularen Anteil der Beleuchtung vorberechnen indem Sie für jeden möglichen Reflexionsvektor $R \in \Omega$ das Integral

$$L_s(R) = \int_\Omega L_{in}(\omega_i) \max(0, R \cdot \omega_i)^n d\omega_i$$

   analog zum vorherigen Aufgabenteil vorberechnen.

3. Implementieren Sie den Vertex-Shader `env_light.vert` analog zu `phong.vert`.

4. Implementieren Sie Environment Lighting im Fragment-Shader `env_light.frag` indem Sie die diffus und spekular vorgefilterten Environment Maps mit den entsprechenden

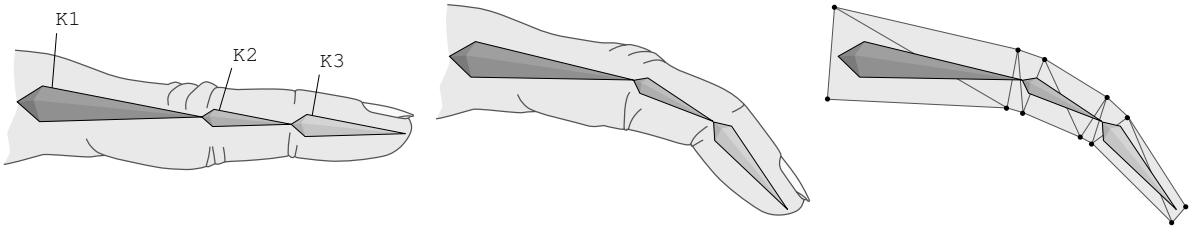
Richtungen auslesen. Die Environment Maps sind im Fragment-Shader als Cubemap-Sampler `EnvironmentTextureDiffuse` bzw. `EnvironmentTextureGlossy` gegeben und Sie können direkt mit den Richtungsvektoren auf die Texturen zugreifen. Werten Sie das Beleuchtungsmodell

$$k_d L_d(N) + k_s L_s(R)$$

aus, wobei N die Normale und R der reflektierte View-Vektor ist. Die Materialparameter k_d und k_s sind als Uniform-Variablen `diffuse_color` bzw. `specular_color` gegeben. Die Multiplikationen $k_d L_d(N)$ und $k_s L_s(R)$ sollen komponentenweise durchgeführt werden.

5 Deformation mit Skelettsystemen (Theorie, keine Abgabe)

Die Animation von deformierbaren Körpern wird oft über sogenannte *Skelettsysteme* bewerkstelligt. Dabei werden einzelne Vertizes eines Modells mehreren *Knochen* zugewiesen. Skelettsysteme sind hierarchisch, sodass Kindknochen die Transformation von Elternknochen übernehmen. Die folgende Skizze zeigt einen Finger, der über drei Knochen deformiert wird.



Der Knochen `K1` ist die Wurzel der Skeletthierarchie. Sein Kindknochen ist `K2`. `K3` ist das Kind von `K2`.

Es sind drei Transformationsmatrizen gegeben. `M1` beschreibt die Lage des gesamten Fingers in Weltkoordinaten. `M2` beschreibt die Lage von `K2` *relativ zu* `K1`. `M3` beschreibt die Lage von `K3` *relativ zu* `K2`.

Für jeden Vertex sind Gewichte `w[0]`, `w[1]` und `w[2]` gegeben. Sie beschreiben, wie viel Einfluss die Transformation des entsprechende Knochens auf den Vertex hat.

Vervollständigen Sie den Vertex-Shader, sodass die Position des Vertex `P` wie angegeben gewichtet transformiert wird! In `gl_Position` muss dann letztendlich die Position in Clip-Koordinaten geschrieben werden.

```
in vec3 P;            // Position des Vertex in Objektkoordinaten.
in float w[3];        // Einfluss der Knochen.

uniform mat4 matVP;   // View-Projection-Matrix.
uniform mat4 M1;      // Transformation des Fingers (Objekt -> Welt).
uniform mat4 M2;      // Transformation von K2 relativ zu K1.
uniform mat4 M3;      // Transformation von K3 relativ zu K2.

void main()
{

    ...

    gl_Position = ...
}
```

Abgabe Laden Sie die Datei `solution.zip` in Ilias hoch. Achten Sie darauf, dass dieses Archiv alle von Ihnen bearbeiteten Dateien enthält: `exercise_06.cpp`, `simple.vert`, `phong.vert`, `phong.frag`, `env_light.vert`, `env_light.frag` und keine zusätzlichen Dateien.

Framework Für jedes Übungsblatt stellen wir ein Framework bereit, das Sie im Ilias-Kurs herunterladen können. Das Framework nutzt C++11 und wird unter Linux getestet. Es ist allerdings auch unter Windows mit Visual Studio 2013 lauffähig. Das Framework enthält das Unterverzeichnis `cglib`. Weiterhin gibt es das aufgabenspezifische Unterverzeichnis `06_shaders`, in dem Sie Ihre Lösung programmieren. Die Datei `Kompilieren.txt` enthält Informationen darüber, wie Sie das Framework kompilieren können.

Achtung: Abgegebene Lösungen müssen in der VM erfolgreich kompilieren und lauffähig sein, ansonsten vergeben wir 0 Punkte. Insbesondere darf Ihre Lösung nicht abstürzen.

Allgemeine Hinweise zur Übung

- Scheinkriterien: Sie benötigen jeweils 50% der Punkte aus den Praxisaufgaben von Block A (Blatt 1, 5, 6) und Block B (Blatt 2, 3, 4).
- Die theoretischen Aufgaben bedürfen *keiner* elektronischen Abgabe.
- Die Abgabe muss im Ordner `build` mit `cmake ../ && make` in der bereitgestellten VirtualBox VM[1] kompilieren, andernfalls wird die Aufgabe mit 0 Punkten bewertet.
- Da nur einzelne Dateien abgegeben werden, müssen diese kompatibel zu unserer Referenzimplementation bleiben. Verändern Sie daher wirklich nur die Dateien, die auch abgegeben werden müssen, insbesondere *nicht* die mitgelieferten Funktionsdeklarationen! Sie können allerdings in den abzugebenden Dateien Hilfsfunktionen definieren und benutzen.
- Sie dürfen sehr gerne untereinander die Aufgaben diskutieren, allerdings muss jede*r die Aufgaben *selbst* lösen, implementieren und abgeben. Plagiate bewerten wir mit 0 Punkten.
- Sie können sich bei Fragen an die Übungsleitung oder das Ilias-Forum wenden.

Alisa Jung `alisa.jung@kit.edu`
Reiner Dolp `reiner.dolp@kit.edu`

[1] `https://cg.ivd.kit.edu/lehre/ws2021/cg/downloads/cgvm.zip`