Karlsruher Institut für Technologie
Lehrstuhl für Computergrafik

Prof. Dr.-Ing. Carsten Dachsbacher
M. Sc. Lucas Alber
M. Sc. Nathan Lerzer
M. Sc. Vincent Schüßler

1. Übungsblatt zur Vorlesung Computergrafik im WS 2024/2025

Abgabe bis Montag, 11.11.2024, 8:00 Uhr

Abgabe Laden Sie die Datei `exercise_01.cpp` in Ilias hoch. Der von Ihnen geschriebene Code sollte sich ausschließlich in den dafür vorgesehenen Funktionen in der Datei `exercise_01.cpp` befinden.

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 `01_colors`, 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.

Alle Übungsleiter	`cg_ws2425@lists.kit.edu`
Lucas Alber	`lucas.alber@kit.edu`
Nathan Lerzer	`nathan.lerzer@kit.edu`
Vincent Schüßler	`vincent.schuessler@kit.edu`

[1] `https://cg.ivd.kit.edu/lehre/ws2023/cg/downloads/cgvm.zip`

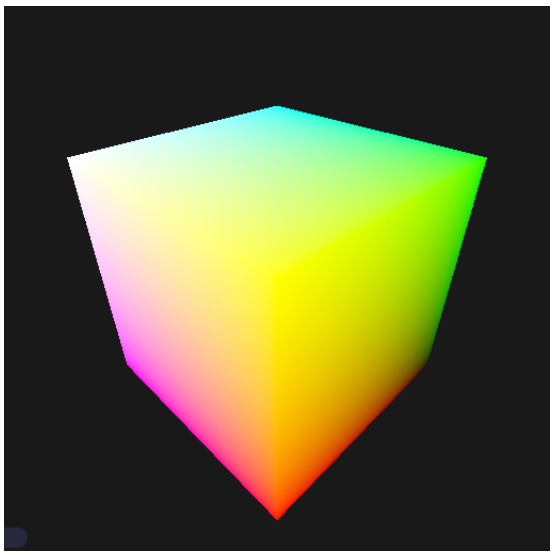
1 *OpenGL Immediate Mode und Color Matching* **20 Punkte**

In dieser Aufgabe sollen Sie Erfahrungen mit dem sogenannten *Immediate Mode* von OpenGL sammeln. Dieser Modus wird in modernem OpenGL nicht mehr eingesetzt, dennoch gibt es noch viel Anwendung in der Industrie. Außerdem eignet sich der Immediate Mode um ,,schnell mal etwas zu zeichnen".

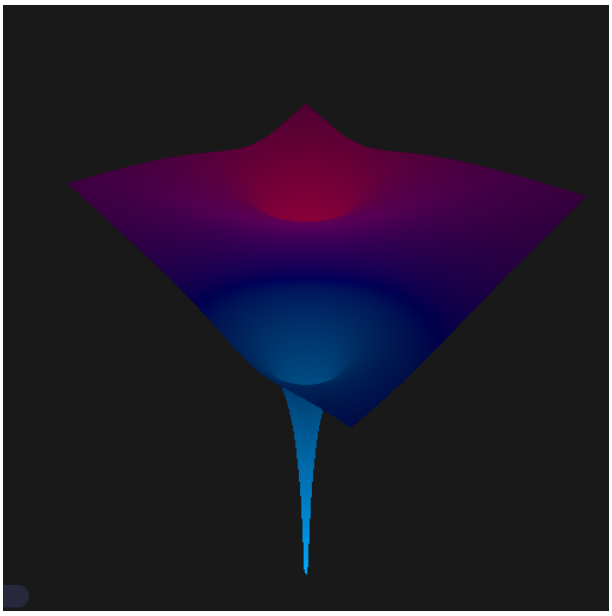
Desweiteren sollen Sie den Kernalgorithmus für Color Matching implementieren. Dazu wird ein Lichtspektrum gegen die Color Matching-Funktionen integriert, woraus man Farben im XYZ-Farbraum erhält. Diese Farben können dann in andere Farbräume, z.B. sRGB, konvertiert werden.

Ihre Implementierung beschränkt sich auf die Datei `exercise_01.cpp`. Sie dürfen sich gerne die anderen Dateien im Framework anschauen. Versuchen Sie, den Programmfluss nachzuvollziehen. Folgen Sie dafür den eingebundenen Headern und den Funktionen, die in `main.cpp` verwendet werden.

a) **(6 Punkte)** Der erste Teil dieser Aufgabe ist eine Visualisierung des RGB-Farbraumes. Dafür wird ein Würfel als Menge von Dreiecken gezeichnet, dessen Eckpunkten sRGB-Farbwerte zugeordnet werden. Ihre Aufgabe ist es, in der Funktion `draw_triangles()` in `exercise_01.cpp` die übergebenen Vertex- und Farbdaten als `GL_TRIANGLES` zu zeichnen. Achten Sie darauf, dass alle Dreiecke in der vorgegebenen Reihenfolge gezeichnet werden. Als zweite Testszene können Sie den *Current renderer* auf „CG Font" setzen.



b) **(6 Punkte)** Für den zweiten Teil stellen Sie den Render-Modus auf „Gravity Field" um. Hier sollen Sie eine Visualisierung des Gravitationspotentials zweier Punktmassen implementieren.



Ihre Aufgabe besteht darin, ein reguläres Gitter zu erzeugen und dieses dann auch zu zeichnen. Das Gitter wird intern in ein Höhenfeld umgewandelt, indem die Punkte proportional zum Gravitationspotential zweier Punktmassen in der Höhe verschoben werden. Außerdem findet intern eine Farbzuordnung statt.

Implementieren Sie das reguläre Gitter in der Funktion `generate_grid()` in `exercise_01.cpp`. Dies soll als Shared Vertex-Repräsentation geschehen. Erzeugen Sie dafür zunächst $(N+1) \times (N+1)$ Vertices in der xy-Ebene. Erzeugen Sie dann $2 \times N \times N$ Index-Vektoren, die das reguläre Gitter aus Dreiecken zusammensetzen.

Achten Sie darauf, dass Dreiecke nicht überlappen und dass die gesamte Fläche $[0,1]^2$ abgedeckt wird.

Implementieren Sie schließlich `draw_indexed_triangles()`, um das Gitter zu zeichnen.

c) **(8 Punkte)** Im dritten Teil sollen Sie Color Matching und eine Visualisierung des Schwarzkörperspektrums implementieren.

Implementieren Sie zunächst `generate_strip()` in `exercise_01.cpp`. Diese Funktion soll einen Triangle Strip mit $2N$ Dreiecken erzeugen, der in der xy-Ebene liegt. Der erste Vertex soll bei $(0, 1, 0)$ liegen, der letzte Vertex bei $(1, 0, 0)$.

Achten Sie besonders darauf, die richtige Reihenfolge einzuhalten. Nutzen Sie außerdem nicht mehr Vertices, als unbedingt notwendig.

Der Triangle Strip wird verwendet, um das Spektrum (im Bild oben) und den Farbgradienten (im Bild unten) zu zeichnen. Implementieren Sie `draw_triangle_strip()`, um beide Elemente darzustellen. Das Spektrum sollte jetzt schon gezeichnet werden, der Gradient ist noch schwarz. Um das zu ändern, müssen Sie zunächst `integrate_trapezoidal_student()` implementieren. Diese Funktion soll die durch `x` und `y` gegebene Funktion mit der Trapezregel[2] integrieren.

Benutzen Sie schließlich `integrate_trapezoidal()`, um `spectrum_to_rgb()` zu implementieren[3]. Hier sollen zunächst die Integrale

$$X = \int_\Lambda S(\lambda)x(\lambda)\mathrm{d}\lambda \tag{1}$$

$$Y = \int_\Lambda S(\lambda)y(\lambda)\mathrm{d}\lambda \tag{2}$$

$$Z = \int_\Lambda S(\lambda)z(\lambda)\mathrm{d}\lambda \tag{3}$$

berechnet werden (Color Matching). Die Funktion S ist das gegebene `spektrum`, die Funktionen x, y und z, sowie geeignete Funktionen zum Konvertieren finden Sie im Verzeichnis `cglib/include/cglib/colors` in den Dateien `cmf.h` und `convert.h`

[2] `http://de.wikipedia.org/wiki/Trapezregel#Sehnentrapezformel`

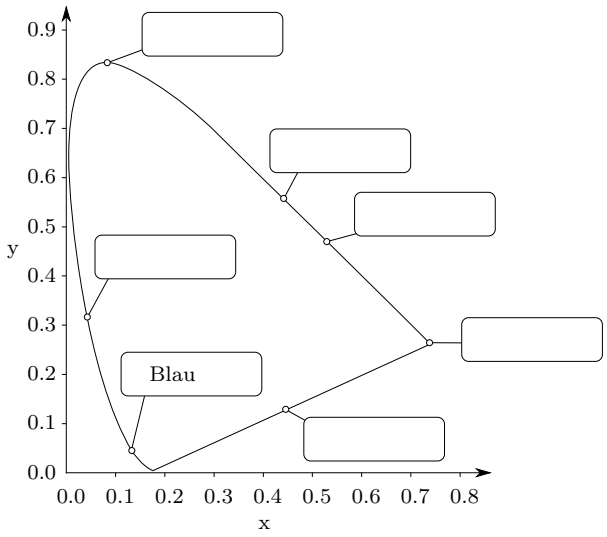
[3] Wird `integrate_trapezoidal_student` verwendet, so können Folgefehler auftreten. Das können Sie durch Nutzung von `integrate_trapezoidal` verhindern; dieses wird bei der Korrektur durch eine Referenzimplementierung ersetzt, und ruft in Ihrem Framework einfach `integrate_trapezoidal_student` auf.

2 *Farben und Farbwahrnehmung (Theorie, keine Abgabe)*

a) Tragen Sie die Farben

Grün, **Rot**, **Gelb**, **Orange**, **Cyan**, **Magenta**

in die entsprechenden Felder im Chromatizitätsdiagramm ein.



b) Welcher der Farbeindrücke aus Aufgabe a) lässt sich nicht durch monochromatisches Licht erzeugen?

c) Die Farbräume xyY und XYZ sind eng verwandt. Wie ist der mathematische Zusammenhang zwischen der Chromatizität (x, y) und der passenden Farbe (X, Y, Z)?

d) Gegeben sind

1) der Farbraum **XYZ**,

2) ein physikalisch realisierbarer **RGB**-Farbraum,

3) sowie der Raum aller Farben, die durch **100 monochromatische Leuchtdioden** mit den äquidistanten Wellenlängen {380nm, 384nm, ..., 776nm} darstellbar sind.

Ordnen Sie diese Räume aufsteigend nach der Größe ihres für den Menschen sichtbaren Gamut.