

Praktikum Computergrafik, WiSe 21/22

Übungsblatt 3

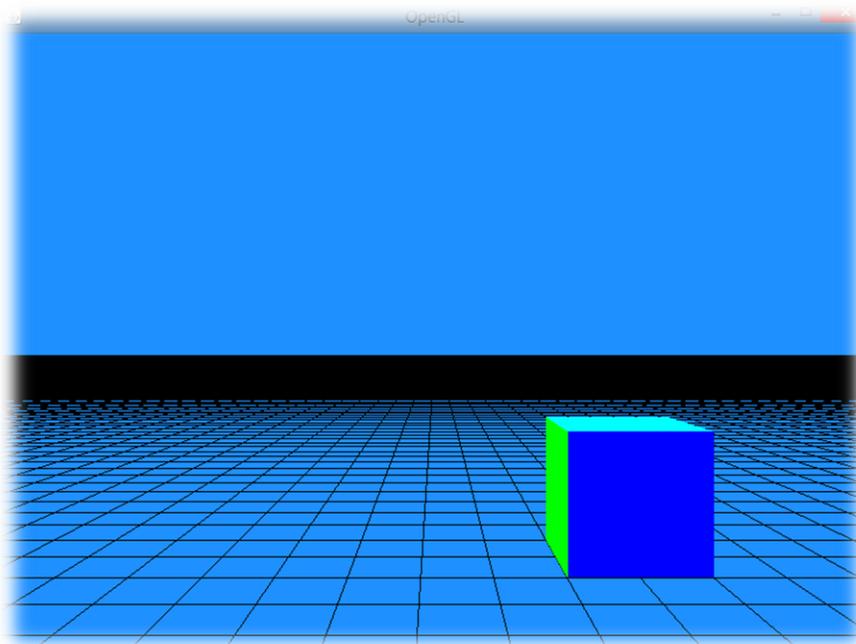
- **Abgabefrist:** 13.12.2021 11:59:59
- Abgabe erfolgt per E-Mail an jeos@mail.com
- **Betreff:** CG21WS ÜB3
- **Erste Zeilen der E-Mail:** Autorennamen (*höchstens 2 Kursteilnehmer*) und Matrikelnummern
- Der **lauffähige Code (Quelltext)** soll **als Anhang** in der E-Mail mitgeschickt werden
- Der Quelltext muss dabei als **ZIP-Archiv** exportiert worden sein (*siehe die Anleitung in den Folien zu #1*)

Quellen:

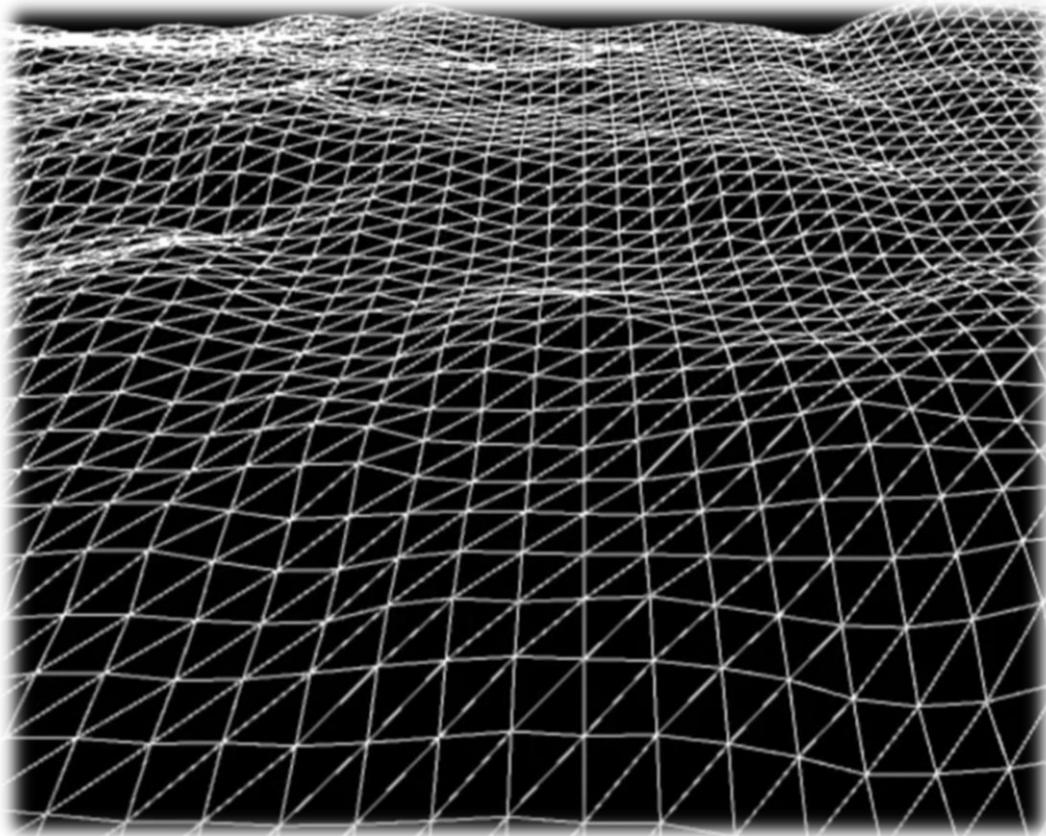
Aufgaben	http://www.uni-forst.gwdg.de/~wkurth/cg21_u03.pdf
Folien	http://www.uni-forst.gwdg.de/~wkurth/cg21_f03.pdf
Code-Frameworks	http://www.uni-forst.gwdg.de/~wkurth/cg21_c05.txt http://www.uni-forst.gwdg.de/~wkurth/cg21_c06.txt http://www.uni-forst.gwdg.de/~wkurth/cg21_c07.txt http://www.uni-forst.gwdg.de/~wkurth/cg21_c08.txt http://www.uni-forst.gwdg.de/~wkurth/cg21_c09.txt http://www.uni-forst.gwdg.de/~wkurth/cg21_c10.txt

Liste der Aufgaben:

1. Erstellen Sie den Code, der die folgendermaßen beschriebene Szene aufbaut:
 - [Quadratmaschenboden](#), der in der ganzen Szene verlegt wird (also die Szene soll mit dem Bodenhorizont rundum versehen sein);
Anregung: versucht den Boden auch durch `GL_TRIANGLE_STRIP` zu verlegen und passt dabei zweite Koordinaten (d.h. die Höhe) von Endknoten an, damit euer Boden wie ein realistisches Gelände aussieht (siehe [dieses Bild](#)).
 - Auf dem Boden platzierter Würfel, dessen Flächen ohne Farbverlauf (also einfarbig) aber unterschiedlich gefärbt sind;
 - Gleichmäßige Bewegung (d.h. mit einem konstanten Schritt) des Betrachters in 6 Richtungen durch die Szene mit den entsprechenden Tastaturtasten;
 - Gleichmäßige 360°-Rotation der Blickrichtung in 4 Richtungen – steuerbar durch die entsprechenden Tastaturtasten;
 - Mouse-gesteuerte gleichmäßige Rotation des Würfels mindestens in 4 unterschiedlichen Richtungen (also besser wenn man jeder Mausbewegung, anhand der `getX()` und `getY()` Befehlen, eine entsprechende Rotationsachse zuordnet), nur bei der gedrückten Umschalttaste (d.h. sonst muss der Würfel unbeweglich bleiben und auf Mouse-Ereignisse nicht reagieren). Dabei ist gleichgültig ob man `mouseMoved(...)` oder `mouseDragged(...)` Ereignis verwendet.



2. Wechseln Sie ihren Würfel zu einem gleichmäßig drehenden Würfel, der stoppt, sobald Sie die Taste **N** drücken und seine Rotation fortsetzt, falls Sie auf den Knopf **Y** drücken.
Dabei ist zu beachten, dass die Rotation dieses Würfels durch den entsprechenden Tastendruck **glattgehend** (*d.h. ohne sichtbaren Sprung in Rotation*) angehalten oder (wenn schon angehalten) auch **leichtgängig** weitergelaufen werden soll.
-



3. Rüsten Sie die Szene mit folgenden zusätzlichen Funktionalitäten aus:

- Die Tastaturtaste **P** soll die Rolle eines Schalters übernehmen, der den für die Szene eingesetzte Projektionsmodus von einem perspektivischen (`glFrustum(...)` oder `gluPerspective(...)` Befehl) zu einem orthogonalen (`glOrtho(...)` Befehl) wechselt;
 - nach einer bestimmten Zeit (*also nicht sofort beim Ausführen des Codes - `System.currentTimeMillis()` kann eingesetzt werden*) ein anderer, dabei deutlich kleinerer Würfel für eine Weile erscheint, dann für eine kurze Weile verschwindet und dieses Verhalten immer wieder wiederholt – dieser kleine Würfel soll um den anderen Würfel ständig drehen solange sich die Maus (*also der Mauszeiger*) über dem Fenster (*auf dem ihre Szene gerendert wird*) befindet, - sonst bleibt der Würfel ohne Rotation aber mit anderen oben beschriebenen Effekten;
 - Die Taste **A** soll die Animation anhalten (`pause()` Methode Ihrer Animationsklasse) und beim nochmaligen Drücken wieder fortsetzen (`resume()` Methode Ihrer Animationsklasse);
 - Die Taste **Esc** soll das Rendern beenden (`System.exit(0)` kann dafür verwendet werden).
-

4. Fügen Sie einfache 3D-Schießspielemente ihrer Szene hinzu und zwar:
- beim Drücken der **Maustaste** wird ein Ball vom Betrachter aus geworfen (mit irgendeiner konstanten Geschwindigkeit und irgendeinem konstanten Gewicht – setzt irgendwelche fixierten Werte nach eurer Wahl) - der Wurfwinkel und Wurfrichtung soll den Mauszeigerkoordinaten entnommen werden (*also wenn ihr beim Mausdrücken die entsprechenden X,Y-Koordinaten bekommt, berechnet daraus einen bestimmten Vektor/dessen Betrag und setzt diese als Wurfrichtung/Wurfwinkelgröße ein*).
 - Der geworfene Ball soll sich entlang einer Wurfparabel sichtbar bewegen und falls der euren großen Würfel trifft, soll das Fenster seine Farbe zu Rot (für eine kurze Weile – quasi wie beim Blitz im Himmel) ändern.
 - Der Ball muss vom Boden oder sonstigen Oberflächen nicht abprallen (diese Komplexität darf also ignoriert werden) – es reicht, wenn der nach einer bestimmten Zeit einfach verschwindet.
 - Zeichnen Sie (irgendwo auf dem Zeichenfenster) die aktuell gesammelten Punkte (muss mit jedem Treffer inkrementiert werden) sowie die Anzahl geworfener Bälle (mittels **TextRenderer**).
 - Fügen Sie die entsprechenden Sound-Effekte hinzu (zB. zu Ballwurf-, Treffer-Ereignissen).

Anregung:

- 1.** Versuchen Sie einen einfachen Roboter zu basteln, der aus den folgenden **sechs** 3D-Objekten besteht (beachtet dabei die Hierarchie von zusammengesetzten Teilen): Kopf, Körper, Arme, Beine.
- 2.** Der Roboter muss im Fokus des Betrachters bleiben und beim Drücken der **Maustaste** seinen ballförmigen Kopf wegschießen ☺ (der in wenigen Sekunden erneut heranwachsen muss [bis ursprünglicher Größe] – während des Wachstums darf nicht geschossen werden).
- 3.** Realistische Bewegung des Roboters darf stark vereinfacht umgesetzt werden.
- 4.** Implementieren Sie eine einfache Kollisionserkennung zwischen dem Ball und dem Zielwürfel.
- 5.** Die Wurfgeschwindigkeit könnte man auch folgendermaßen berechnen: je länger man die Maustaste gedrückt hält, umso höher (bis auf einem Grenzwert) wird die Wurfgeschwindigkeit.

Hinweise:

1. Für trigonometrische Berechnungen (zB. für die Wurfparabel) verwendet ihr **Java.lang.Math** Bibliothek und ihre Methoden (für Randeffekte bei Geländegestaltung usw. stehen beispielsweise Methoden der Klasse [java.util.Random](#) zur Verfügung).
 2. Für die „echte“ Zufälligkeit könnt ihr auch [diese Bibliothek](#) herunterladen und einsetzen (Zufallszahlen kommen vom RANDOM.ORG-Server – die generierten Zahlen basieren auf atmosphärischen Rauschmessungen).
-