GDV OpenGL-Praktikum

Bastian Kreuzer (734877), Adrian Müller (734922)

# Kurzbeschreibung Animation Skiroboter

In einer schneebedeckten Gebirgslandschaft fahren 12 humanoide Roboter einen Berg hinunter. Der in Ausgangssicht befindliche vorderste Roboter auf der Piste lässt sich mit den Tasten J und L nach rechts bzw. links bewegen. Die übrig restlichen Roboter fahren die Piste in Kurven hinab.

Ein Roboter besteht aus Kopf, Hals, Torso, Ober- und Unterarm, Ober- und Unterschenkel, Füßen und Skiern. Die Roboter sind in ihrer farblichen Gestaltung einerseits Crashdummys und andererseits typischen Förderbandfabrikrobotern nachempfunden: Sie sind überwiegend in orange, grau & schwarz gehalten und mit Symbolen versehen.

Beim Herabfahren sind alle Roboter leicht nach vorne (um die x-Achse) geneigt. Bei einer Rechts- oder Linkskurve ändert der Roboter seine Orientierung (y-Achse) und neigt sich zur Seite (z-Achse), entsprechend wird das jeweils innere Bein gefaltet, sodass die Skier auf der Piste bleiben. Die Arme sind leicht nach hinten geneigt und werden ebenfalls abhängig von der Seitenneigung bewegt.

# Implementierungsweg

(Die GIT-Historie ist ebenfalls auf der CD und unter <https://github.com/addilind/ModernGDV/> abrufbar)

Da unsere Implementierung auf Basis von Shadern stattfinden sollte, musste zunächst ein Grundgerüst zusammengebaut werden: Die Bibliotheken (GLFW zur Fenster/Kontextverwaltung, GLM für mit OpenGL kompatible Mathematikoperationen, anfangs GLEW, dann libepoxy und schließlich eine per glloadgen generierte Bibliothek zur Verwendung von OpenGL-Extensions) mussten in einem Projekt zusammengeschnürt, Programmabläufe wie die Initialisierung und der Renderloop geschrieben sowie minimale Shader programmiert werden.

Anschließend haben wir die geführten Aufgaben mithilfe des erstellten Frameworks umgesetzt.

Daraufhin haben wir mit der Erstellung des Roboters angefangen; dabei haben wir jedes Bauteil zunächst auf Papier skizziert (siehe Designdokumente.pdf im Doku-Ordner auf der CD), und dann in Form von Drei- und Rechtecken implementiert. Nach jedem fertiggestellten Teil haben wir den Roboter zusammengesetzt gezeichnet und (falls nötig) die Werte angepasst, falls Proportionen nicht wie gewünscht aussahen.

Während der Programmierung der Robotergeometrie haben wir allmählich Kamerasteuerung, Texturen, Beleuchtung per Phong-Shading und Skybox hinzugefügt.

Nach der Robotergeometrie haben wir ein Heightmap-basierendes Terrain erstellt und Funktionen geschrieben, die aus gegebenen Neigungen des Roboters die Beine entsprechend faltet.

Zuletzt haben wir die Animationen der Roboter implementiert.

# Übersicht

* Trennung in 2 Codeebenen: Aufgabenstellung & „Treiber“, der von der OpenGL-Schnittstelle abstrahiert
* Übergeordneter Programmablauf:   
  

# Features

## Texturen

Jede Geometrie wird mit einer Textur dargestellt.

### Mipmaps

Mipmaps sind erforderlich, um zu verhindern, dass Texturen bei Animationen rauschen, wenn sie kleiner dargestellt werden, als ihre Originalauflösung vorgibt. Daher speichert man neben der eigentlichen Textur auch verkleinerte Versionen des gleichen Bildes ab – üblicherweise jeweils halb so groß, wie die vorhergegangene Stufe, bis zur Auflösung 1x1 Pixel herunter. Beim Rendern sucht sich die Grafikkarte die geeignetste Auflösung aus.

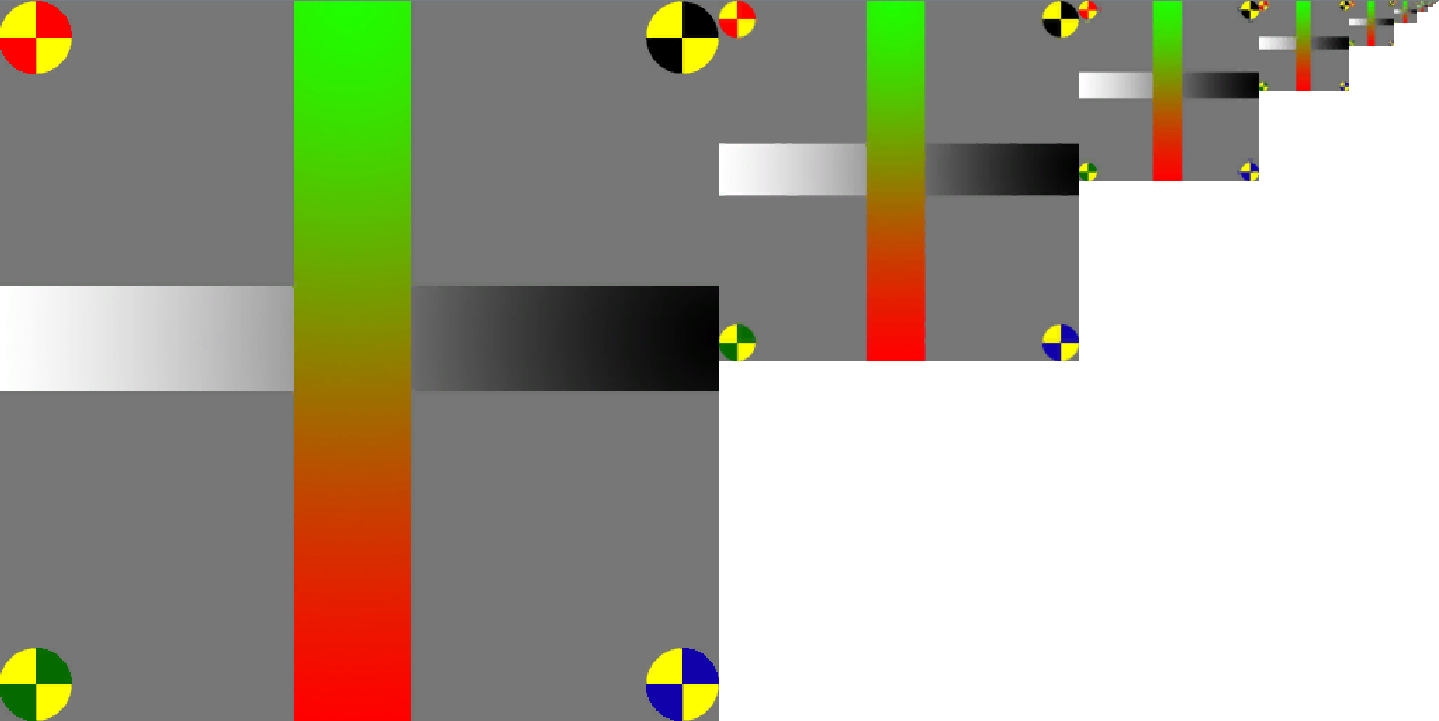


Abbildung : Die verwendete Testtextur und die dazu erzeugten MIPMAPs

### DirectDrawSurface (DDS)-Dateiformat, S3TC-Komprimierung

Das DDS-Dateiformat ist ein von Microsoft für seine Schnittstelle DirectX entworfenes Dateiformat zur Speicherung von Texturen, die für 3D-Rendering verwendet werden. Im Vergleich zu herkömmlichen Formaten, wie PNG oder JPEG, unterstützt es u.a. das Speichern von MIPMAPs, definiert aber kein eigenes Komprimierungsverfahren.

Zur Komprimierung verwenden wir daher die S3TC-Komprimierungen (auch bekannt als DXT1-5, wie Microsoft die Algorithmen in DDS nennt). Der Vorteil dieser Komprimierungsverfahren ist, dass Grafikkarten aller verbreiteten Hersteller das Format in Hardware ohne Zeitverlust dekomprimieren können. Neben geringeren Anforderungen an den Videospeicher und kürzeren Ladezeiten - da weniger Daten (i.d.R. via PCI-E) in den GRAM geladen werden müssen – macht es vor allem auch die eigene Implementierung des Dekomprimierens überflüssig, weil das Bild komprimiert an die Grafikkarte übertragen werden kann.

## Heightmap

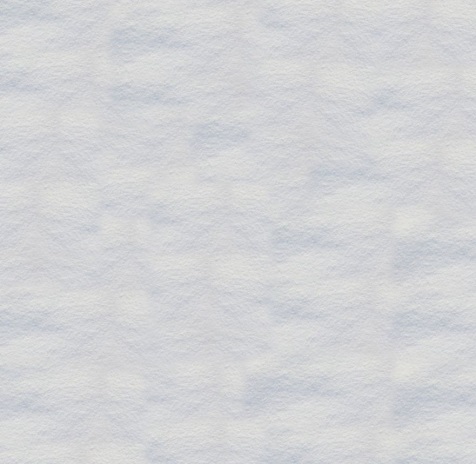
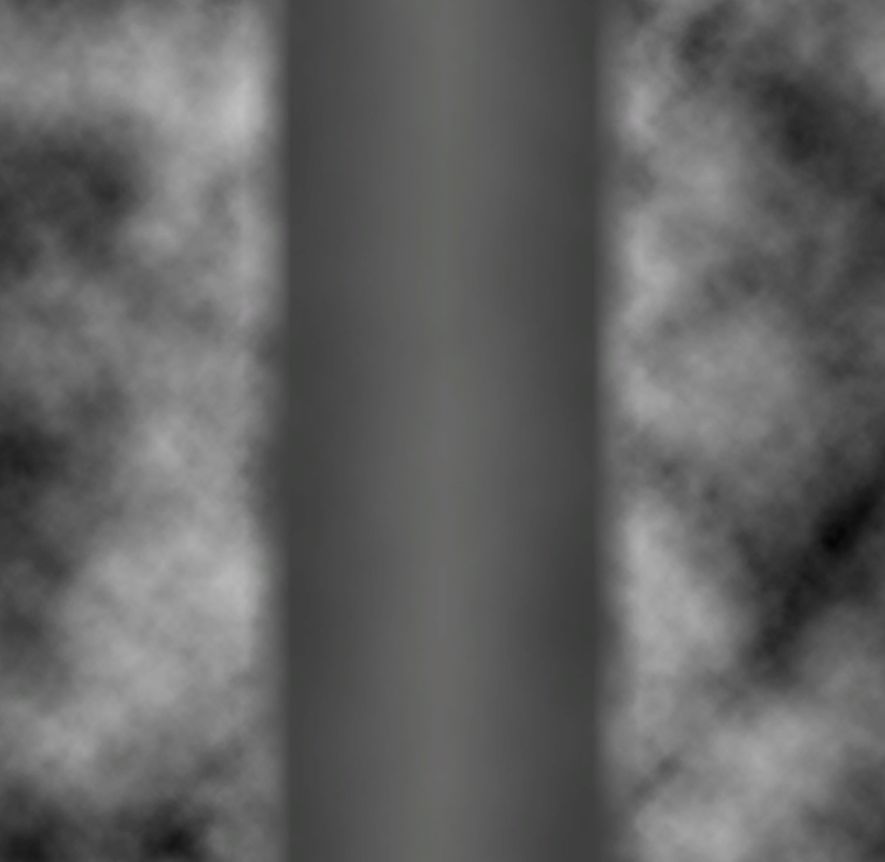


Abbildung : Heightmap Schneepiste Abbildung : Textur Schnee

Die Heightmap ist ein Bild, das ausschließlich Grauwerte enthält. Diese Grauwerte werden von einem entspechenden Shader als Höhe (y-Koordinate) eines Terrains interpretiert. Je heller der Grauwert, desto höher wird an dieser Stelle die verwendete Textur gezeichnet. Die Schneepiste ist nahezu eben bzw. leicht rund und in der Heightmap mittig zu erkennen.

# Tastensteuerung

W; S Kamera vorwärts, rückwärts bewegen  
A; D Kamera nach rechts, nach links bewegen  
Shift; Strg Kamera auf/ab bewegen

Pfeiltasten Kamera drehen  
Bild hoch Heranzoomen  
Bild runter Herauszoomen

1 Transformationen der Roboter nicht aktualisieren  
2 Piste nicht weiterschieben   
3 Wireframe-Ansicht umschalten  
4 Shader neu laden  
5 Lampe im LookAt-Punkt zeichnen  
6 Lampe schwächer machen (aktueller Wert wird in Konsole angezeigt)  
7 Lampe stärker machen

J; L Vordersten Roboter nach rechts bzw. nach links neigen