

Übungen zu Graphikprogrammierung in C++

Abgabetermin: 17. Januar 2005

Am 18. Januar wird von jedem Team ca. 5 Minuten die Lösung der Aufgabe 22 in dem wöchentlichen Treffen präsentiert. Jedes Team wählt hierfür einen Präsentator aus.

Was man wissen muss:

- *Texture Mapping*
- *Modelview Matrix*
- *Blending*
- *glut Interaktions- & Darstellungsfunktionalität*

Aufgabe 21 (H) Partikelsysteme - Schneeflocken und Regen

Mit Partikelsystemen kann man Bewegungen bzw. Krafteinwirkungen auf Objekte einer Szene modellieren. Beispielsweise könnte man den Ausbruch eines Vulkans oder Bewegungen von Schilf im Wind visualisieren. In dieser Übung sollen Schneeflocken und Regentropfen mittels Partikelsystemen modelliert werden. Eine Temperaturvariable soll mit geeigneten Tasten verändert werden können. Sobald die Temperatur über 2° Celsius steigt soll es regnen. Schneeregen soll im Übergang von 0° - 2° Celsius modelliert werden.

- Entwerfen Sie eine Klassenhierarchie mit Oberklasse `Particle` und zwei Unterklassen `Raindrop` und `Snowflake`. In `Particle` sollen alle Attribute und Methoden implementiert werden, die Partikel gemeinsam haben, d.h. Position, Geschwindigkeit und Beschleunigung (*engl.: acceleration*). Sie können auch noch Variablen für Lebenszeit eines Partikels, ob ein Partikel aktiv ist, und beispielsweise Rot-, Grün- und Blauwert der Partikel implementieren. Es soll zwei abstrakte Methoden geben, `update(float s)` und `render(void)`, die in den Unterklassen implementiert werden sollen. Implementieren Sie diese zunächst nur in der Klasse `Raindrop` (verwenden Sie eine passende 3D-Komponente¹, verzichten Sie auf Texture Mapping).
- Implementieren Sie eine Klasse `ParticleEmitter`, die mehrere Partikel verwalten kann. Diese soll möglichst einfach auf die Attribute der Klasse `Particle` zugreifen können. Sie können hierzu entweder strikt objektorientiert geeignete Getter- und Settermethoden verwenden, oder das *friend*-Konzept verwenden. Ist eine Klasse A in einer Klasse B als friend deklariert, so kann A auf alle `private` und `protected` Attribute der Klasse B zugreifen. Dies hat nebenbei auch noch den Vorteil, dass nun nur die Klasse A auf die privaten Attribute zugreifen kann, andere Klassen allerdings nicht (weil man keine Getter- und Settermethoden implementiert hat).

¹in glut sind schon vorgefertigte Strukturen implementiert, z.B. `glutSolidTeapot`, `cubes`, `spheres` oder ähnliches

Schreiben Sie eine main-Funktion und ein Fenstersystem mit glut in der Datei `mainA21.cpp`². Hierbei sollen v.a. display- und keyboard-Funktionen definiert werden. Instanzieren Sie nun mehrere Raindrops in der main-Funktion von `mainA21.cpp`. Mit der update-Funktion soll eine Animation simuliert werden. Diese soll immer stattfinden, nicht nur wenn man eine bestimmte Benutzereingabe macht. Implementieren Sie dazu eine geeignete Funktion `idle()` und übergeben Sie diese an glut mit `glutIdleFunc(idle);`.

- c) Regen fällt normalerweise nur in y-Richtung. Jetzt soll jedoch auch noch Windeinfluss modelliert werden. Verwenden Sie glut-keyboard-Abfragen

```
GLUT_KEY_LEFT      //Left function key
GLUT_KEY_RIGHT     //Up function key
```

um Wind von rechts oder links auf die Szene einwirken zu lassen.

- d) Im CVS ist unter `tools/bmpfiles` eine Datei `snowflakes.bmp` eingecheckt. Diese Datei ist schon auf eine Zweierpotenz skaliert (um sie für Texture Mapping verwenden zu können). Nun hat sie allerdings einen schwarzen Hintergrund, der vermieden werden soll (nur die Schneeflocke soll visualisiert werden in der Textur). Schreiben Sie eine Funktion `setAlpha(CAMP::Bitmap texImage)`, die aus den RGB-Daten des Bitmaps einen Datenarray mit RGB und Alpha-Wert erstellt. Mittels dieser Funktion soll nun eine Textur erstellt werden, die lediglich die Schneeflocke zeichnet, den Hintergrund der Textur allerdings nicht, also hier die schon vorhandenen Werte im Puffer verwendet. Verwenden Sie dazu die OpenGL-Funktionen `glBlendFunc(GLenum sfactor, GLenum dfactor)` bzw. `glEnable(GL_BLEND)`.
- e) Implementieren Sie nun die Methoden der Klasse `Snowflake` und verwenden Sie in der `render`-Methode die `snowflake`-Textur. Da diese 256×256 Pixel hat skalieren Sie auf geeignete Größen und verwenden Sie Mipmapping.
- f) Nun sollen Temperaturangaben implementiert werden. Mittels den Tasteneingaben `T` und `Shift+T` soll die Temperatur verringert, bzw. erhöht werden³. Zeigen Sie die Temperatur in dem glut-Fenster an. Dies kann folgendermaßen implementiert werden. Zuerst sollten alle Attribute (z.B. `GL_CURRENT_BIT`, usw.) auf den Stack gelegt werden (mit `glPushAttrib(GL_ALL_ATTRIB_BITS)`). Dann kann ein String erstellt werden und dieser mittels `glRasterPos2f(...)` an die richtige Position gesetzt werden. Mit der Funktion `glutBitmapCharacter(GLUT_BITMAP_8_BY_13, string[i]);` kann der String gezeichnet werden, wobei jeder einzelne Character durchlaufen werden muss.

Aufgabe 22 (H) Das Matterhorn

In dieser Aufgabe soll anhand der Höhendaten des Matterhorns das Matterhorn modelliert werden. Hierfür wird das Konzept von Shading eingeführt. Das Shading ist dafür zuständig, je nach Lage des Oberflächenpatches und unter Verwendung der zuvor definierten Lichtquelle, die Polygone mit authentischen Intensitätswerten zu rendern. Das Modell des Matterhorns besteht aus einem $81 \times 81 \times 3$ Array. Die Daten beschreiben je ein x , y und z Wert. Die Daten können Sie im CVS unter `prograkt/tools/data` finden. `data_big.h` enthält die höchste Auflösung, `data4.h` und `data11.h` enthalten weniger detaillierte Modelle.

- Als erste Teilaufgabe soll das Array der Höhendaten gerendert werden. Hierfür kann die Datei `mainA20.cpp` modifiziert werden, insbesondere die `draw()` Funktion durch das Zeichnen von Quadraten von jeweils vier angrenzenden Punkten. Da bisher noch keine Normalen dargestellt wurden, erscheint das Modell in einer einheitlichen Farbe.

²Sollten Sie Hilfe mit glut benötigen greifen Sie auf die main-Funktionen der letzten Aufgabenblätter zurück, oder lesen Sie im Internet unter <http://www.lighthouse3d.com/opengl/glut/> die Funktionalitäten nach

³Shift-Abfrage mit glut: `GLUT_ACTIVE_SHIFT`

- Nun sollen verschiedene Farben, bzw. Texturen für die Landschaft verwendet werden. Zwischen 3000m und 3300m gibt es noch Vegetation und daher sollten diese Quads grün, bzw. mit einer entsprechenden Textur versehen werden. Zwischen 3300m und 3800m ist felsiges Gelände. Hier sollten die Quads grau, bzw. mit einer entsprechenden Textur versehen werden. Auf allen Punkten bzw. Quads ab 3800m liegt dauerhaft Eis und Schnee. Dies soll dementsprechend durch Farbe, bzw. Textur realisiert werden.
- Die Lichtverhältnisse sind immer noch nicht realistisch. Hierfür werden die Normalen an den Eckpunkten benötigt. Diese berechnen sich aus dem Kreuzprodukt der Vektoren der Mittelwerte der angrenzenden Linien. Im Detail geschieht es folgendermaßen: Da es jeweils zwei unterschiedliche Vektoren in x und y Richtung gibt ($i-1,i$ und $i,i+1$) müssen diese Werte gemittelt werden. Danach muss das Ergebnis normiert werden ($\|\bar{x}\|_2$). Der Vektor senkrecht zu der x - und y -Richtung ist die Normale, die durch das Kreuzprodukt bestimmt werden kann. Durch das Drücken der Taste 's' soll nun zwischen den zwei Modi `glShadeModel (GL_SMOOTH)` und `glShadeModel (GL_FLAT)` gewechselt werden.
- Es soll nun durch Maus- und Keyboardeingaben eine "Fly-Trough" über das Matterhorn realisiert werden. Das Programm soll dahingehend geändert werden, dass man sowohl die Position der Kamera, als auch die Orientierung der Kamera in allen drei Freiheitsgraden steuern kann.
- Das Partikelsystem aus Aufgabe 21 soll nun verwendet werden. Durch Tastendruck soll es auf dem Matterhorn schneien. Unterhalb von 3500m regnen und nicht schneien. Windeinfluss soll ebenfalls implementiert werden.

Frohes Fest und einen guten Rutsch ins neue Jahr!