Einführung in die Computergrafik

Wintersemester 2018/2019

Prof. Dr. Philipp Jenke

HAW HAMBURG | Department Informatik

Aufgabe 2

Texturen und Animation

Abbildung 1: Die animierte Spielerfigur (Eichhörnchen) steht auf einem texturierten Brick.

In diesem Aufgaben beschäftigen Sie sich mit zwei Aufgaben, um die visuelle Darstellung des Spiels ansprechender zu gestalten: Texturierung und Animation.

Texturen

Erweitern Sie die Darstellung der Bricks durch die Verwendung von Texturen. Dazu müssen Sie im generierten Dreiecksnetz Texturkoordinaten verwenden. Beim Erstellen der einzelnen Dreiecke müssen für jeden Eckpunkt Texturkoordinaten festgelegt werden. Als Textur verwenden Sie einen sogenannten Texturatlas (siehe Abbildung 2).

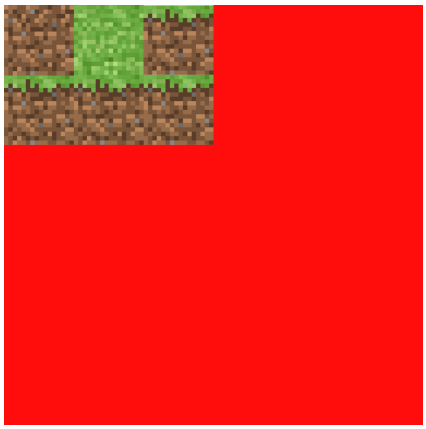

Abbildung 2: Ein Texturatlas beinhaltet mehrere einzelne Texturen in einem Bild, hier die sechs Seitenflächen für einen Brick. Jede der Seiten hat im Texturbild die Kantenlänge $\frac{1}{6}$.

Gehen Sie folgendermaßen vor:

- Weisen Sie dem TriangleMesh-Objekt ein Texturbild zu:
  textures/platformer_textures.png.
- Generieren Sie Texturkoordinaten für die vier Eckpunkte der sechs Seiten im Texturatlas
- Weisen Sie jedem Eckpunkt jedes Dreiecks die passenden Texturkoordinate zu (hier bietet sich ein ähnliches Vorgehen wie bei den Vertices an: Zunächst geht man von den Texturkoordinaten $(0,0), (1,0), (1,1), (0,1)$ für

die beiden Dreiecke einer Seite aus. Diese Texturkoordinaten werden wieder um die Seitenlänge ($\frac{1}{6}$) skaliert und dann um die linke untere Ecke verschoben (z.B. ($\frac{4}{6}, \frac{1}{6}$) für die Seite Z_{NEG}).

Abbildung 3 zeigt schematisch den Aufbau des Texturatlas' für die Texturkoordinaten (u, v).

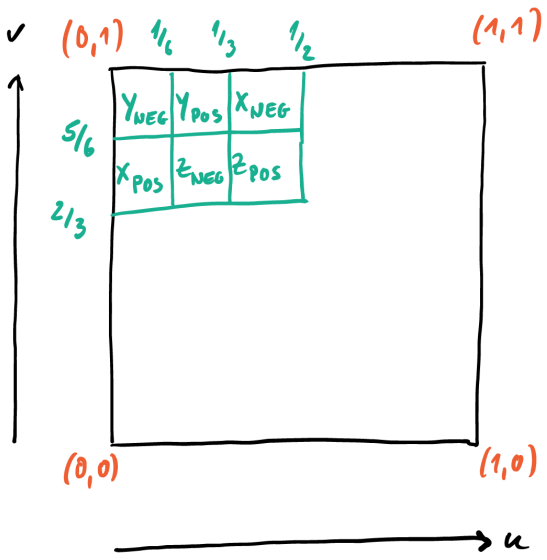

Abbildung 3: Zugriff in eine Textur durch die Texturkoordinaten (u, v). Teiltexturen für den Standard-Brick oben links.

Animation

Jetzt wollen wir die Spielfigur noch animieren - über die Zeit soll sie nicht starr dastehen, sondern sich bewegen. Die Animation soll hier durch Keyframes umgesetzt werden. Für die Spielfigur existiert also nicht nur ein Mesh sondern mehrere, die jeweils unterschiedliche Phasen der Animation zeigen. Eigentlich würden man hier einen komplex animierten Avatar benötigen. Die Generierung eines solchen Avatars ist aber sehr aufwändig. Wir machen es uns einfach und generieren die Keyframes durch Skalierung. Wir beginnen mit dem Original-Mesh (in `Player`) und nennen es `base`. Die Animation soll n Schritte haben. Den Skalierungsfaktor für den i'ten Keyframe berechnet man dann als:

$$scale = (1.0 - \delta) + (0.5 \cdot \cos(\alpha) + 0.5) \cdot \delta$$

mit

$$\alpha = i * 2.0 * \pi / n$$

Der Wert δ legt den maximalen Grad der Skalierung fest. Will man das Mesh um maximal 10% defomieren, dann ist `scaleFactor = 0.1`.
Schreiben Sie einen neuen Szenengraph-Knoten `AnimationNode`, der von `InnerNode` erbt. Er überschreibt die `traverse()`-Methode von `InnerNode`. Statt beim Traversieren des Knotens alle Kinder zu besuchen, wird immer nur einer der Knoten besucht (gerendert). Alle k Frames wird dabei der nächste Keyframe gewählt. Nach dem letzten Keyframe kommt wieder der erste Keyframe.

Im Rahmen des Praktikums wird über die Aufgaben hinweg ein Jump'n'Run Computerspiel (Platformer) entwickelt. In jedem Aufgabenblatt entwickelt sich das Spiel ein wenig weiter. Das Spiel ist in das Framework für die Lehrveranstaltung integriert. Eine Anleitung zum Einrichten des Frameworks finden Sie in der Dokumentation zum Framework auf der EMIL-Seite zur Veranstaltung. Innerhalb des Frameworks ist bereits Funktionalität zum Spiel vorgegeben. Eine Dokumentation dieser Funktionalität findet sich im doc-Ordner des Packages für den Platformer.
Das Spiel kann sowohl auf dem Smartphone unter Android verwendet werden als auch in einer Desktop-Anwendung. Ich empfehle, im Praktikum auf die Desktop-Entwicklung zu setzen, weil sich diese viel einfacher und schneller debuggen und kompilieren lässt. Informationen zur Einrichtung finden Sie im doc*-Ordner des Frameworks.*