Algorithmenbeschreibung

Autoren:

David Eitelsebner – 1556111

Christian Alexander Steinparz - 1555766

Inhaltsverzeichnis

2. Billboards....2
3. Animated Textures....2

1. Partikeleffekte

Die gesamte Berechnung der Partikeln erfolgt in der ParticleSGNode. Damit diese auch funktioniert, muss der Particle-Shader aktiviert sein.

Die PartikelSGNode kann über Parameter angepasst und so zu verschieden aussehenden Partikeleffekten modifiziert werden, allerdings liegt die Hauptanwendung in der Simulation von Feuer, weshalb z.B. auch zufällig auftretende Funken (Spark-Particles) integriert sind.

Die Shader benötigen für das Rendern einige Attribute bzw. Uniforms:

- „a_position“: Array mit den Positionen aller Partikel
- „a_color“: Array mit den Farben für alle Partikel
- „a_size“: Array mit der Größe der Quadrate für alle Partikel
- „u_modelView“: ModelView-Matrix
- „u_projection“: Projection-Matrix

Bei jedem Renderaufruf wird zu Beginn eine konfigurierbare Anzahl an neuen Partikeln erzeugt (newSpawns). Zusätzlich kann eine Varianz hinzugefügt werden.

Die Erzeugung eines Partikels entsteht zufällig: Es wird zufällig eine Farbe über einen ausgewählten Farbbereich festgelegt. Die Größe wird ebenfalls in einem einstellbaren Bereich zufällig gewählt.

Jedes Partikel bekommt außerdem einen zufällig erstellten Richtungsvektor, der jedoch zu einem definierbaren Anteil in eine Richtung zeigen muss. Außerdem werden eine Startposition sowie eine Partikel-Geschwindigkeit zufällig ausgewählt (jeweils in konfigurierbaren Bereichen).

Die Partikel werden in einem Array (fireParticles) gespeichert.

Das gleiche Verfahren wird mit den Funken-Partikel angewandt, jedoch wird hier pro Render-Aufruf maximal 1 Partikel erzeugt und ob dieses erzeugt wird, hängt von einer Zufallszahl ab.

Als nächstes wird das Zentrum (durchschnittliche X/Z-Position) aller Partikel berechnet. Entfernt sich ein Partikel weiter vom Zentrum, wird es schneller „schwächer“, das führt dazu, dass die Flamme im Zentrum

höher brennt.

Zusätzlich reagieren die Partikel auf die Bewegung ihrer Quelle. Dazu wird die Positionsveränderung seit dem letzten Render-Aufruf berechnet (actPosition-lastPostion=moveVec). Um die tatsächliche Bewegung auch in eine richtige Reaktion der Partikel entgegen der Bewegung zu realisieren, muss außerdem der Vector in die View transformiert werden. Dies geschieht durch Multiplikation mit der inversen SceneMatrix.

Ein weiterer Einflussfaktor ist Wind. Dieser wird hier durch einen zufällig erzeugten Richtungs-Vektor erzeugt.

Anschließend wird für jedes Partikel, das sich im Array befindet folgendes durchgeführt:

Zuerst wird der Richtungsvektor für die Bewegung des Partikels ermittelt. Dies geschieht durch aufaddieren der zufälligen Startrichtung, des Move-Vektors und des Wind-Vektors. Anschließend wird die Partikel-Position mit dem Richtungsvektor aktualisiert.

Für jedes Partikel wird außerdem seine Distanz zum oben erwähnten Zentrum berechnet. Je weiter das Partikel von diesem Zentrum entfernt ist, desto schwächer wird es. Dies geschieht durch die Reduktion der Alpha-Values. Zusätzlich wird bei jedem Render-Durchlauf ein fixer Anteil vom Alpha-Value abgezogen sowie die Partikelgröße reduziert.

Hat ein Partikel einen Alpha-Wert <= 0 bzw. eine Größe <= 0, wird es aus dem Array entfernt.

Zusätzlich kann eine maximale Distanz die ein Partikel zurücklegen darf konfiguriert werden. Wird diese Überschritten, verschwindet es sofort.

Befindet sich das Partikel nach diesem Durchlauf noch immer im Array, wird Position, Farbe und Größe des Partikels in die zu Anfangs erwähnten Attribute-Buffer hinzugefügt.

Nachdem jedes Partikel aktualisiert wurde, werden die GL_ARRAY_BUFFER aktualisiert und es erfolgt ein gl.drawArrays Aufruf mit gl.POINTS-Setting, wobei die Partikel-Größe als Point-Size dient.

Da sich die Feuerpartikel alle überlagern sollen, gibt es noch eine Besonderheit.

Erstens wird mit gl.depthMask(false) der Schreibzugriff auf den Z-Buffer gesperrt, womit alle Partikel gerendert werden. Zusätzlich wird Blending mit SRC_ALPHA, ONE aktiviert, was dazu führt, dass sich

im Zentrum des Partikeleffekts alle Farben aufaddieren und somit eine Art heißeres Zentrum simuliert wird.

In den Shadern wird nur mehr die Transformation in die Projektion-World durchgeführt und die Farbe für jedes Partikel gesetzt. Die einzige Besonderheit ist im Fragment-Shader: die Größe der Partikel muss hier durch ihren Abstand zur Camera-Position dividiert werden, da das Zeichnen des gl.POINTS sonst Positionsunabhängig immer gleich große Partikel erzeugen würde.

2. Billboards

Billboards sind als eigene Scene-Graph-Node „BillboardSGNode“ implementiert und „extenden“ dementsprechend die TransformationSGNode. Die render-Methode ist so angepasst, dass über die mat4.lookat-Methode die lookAt-Matrix zwischen der Matrix der BillboardSGNode und der invertierten ViewMatrix des Context „context.invViewMatrix“ berechnet wird. Genauer werden dabei die X und Z-Koordinaten der Kamera berücksichtigt und die Y-Koordinate vernachlässigt, damit die Billboards immer aufrecht stehen. Die resultierenden Werte der lookAt-Matrix werden auf die Matrix der BillboardSGNode geschrieben (mit Ausnahme der Position). Anschließend wird super.render(context) aufgerufen um den Graphen weiter zu durchlaufen

3. Animated Textures

Die „AnimatedTextureSGNode“ wurde implementiert um das wiederholte Durchlaufen eines Arrays von Texturen darstellen zu können und damit Animationen auf beispielsweise Billboards zu erreichen. Man übergibt der „AnimatedTextureSGNode“ ein Array an Texturen, die zu verwendende Texture-Unit, eine Animationsgeschwindigkeit und möglicherweise die Kinder der Node. Das Anwenden der Texturen geschieht gleich wie bei anderen TextureSGNodes. Die Animation wird folgendermaßen erzielt: Im Konstruktor wird zuerst die Zeit in eine lastTime gespeichert. Bei jedem Aufruf der render-Methode wird die aktuelle zeit in „currentTime“ gespeichert. Bevor die Textur gebunden wird per „gl.bindTexture“ wird überprüft, ob sich „lastTime“ und „currentTime“ um die Animationsgeschwindigkeit unterscheiden. Ist dies der Fall, wird der Index, welcher das derzeitig Texturobjekt im Array kennzeichnet, inkrementiert. Wird das Ende des Arrays erreicht, wird der Index wieder zurückgesetzt, damit ein Zyklus im Darstellen der Animation entsteht.