Prozedurale Generierung von Planeten

von Robin Gottschalk

Was bedeutet prozedurale Generierung?

* Methode zur Erzeugung von Programminhalten in Echtzeit
* Die Inhalte werden keineswegs Zufällig erzeugt
* Die Generierung folgt deterministischen Algorithmen



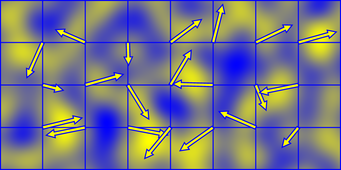
Anwendungsmöglichkeiten:

* Simulationen / Spiele
  + Terrain
  + Galaxien
  + Vegetation
* Texturen
* 3D Objekte
* Sounds / Musik

Perlin Noise

Die Perlin Noise erzeugt zufällig wirkende Strukturen in beliebig vielen Dimensionen. Eine weitere Dimension, die Zeit, kann immer zur Animation der jeweiligen Noise verwendet werden.

Funktionsweise der zweidimensionalen Noise:

* Die Noise besteht aus einer Art Gitter, bei dem jedem Eckpunkt ein Gradientenvektor zugeordnet wird.
* Um den Wert an der Position des gegebenen Punktes zu berechnen, betrachtet man nur den Gitterabschnitt und somit die vier Eckpunkte (acht in 3D) um den Punkt.
* Man bildet nun Vektoren von den Eckpunkten zum Punkt und man berechnet mit den Gradientenvektoren jeweils das Skalarprodukt, sodass jedem Eckpunkt ein Skalar zugeordnet ist.
* Schlussendlich kann man den Wert des Punktes durch Interpolation der Skalarprodukte erhalten. Zur Interpolation wird die von Ken Perlin definierte Fade Funktion genutzt: 6*t*5-15*t*4+10*t*3
* Die Frequenz der Noise bestimmt die Feinheit des Gitters.

Hohe Frequenz = feines Gitter.

* Nun können noch verschiedene Oktaven (Noise mit bestimmter Frequenz und Amplitude) addiert werden, um ein möglichst natürliches Resultat zu erhalten.

Implementierung

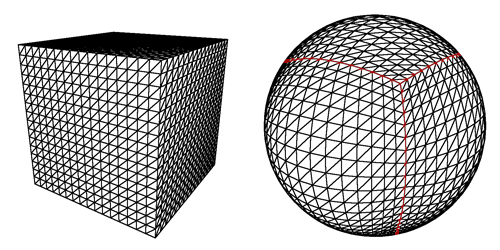
OpenGL Pipeline

* Die einzelnen Punkte (Vertex) werden in den Vertex Shader geladen und dort verarbeitet.
* In den nächsten beiden Schritten werden dann Geometrische Formen erstellt.
* Diese werden dann gerastert und im Fragment Shader je nach Belichtung, Textur, Farbe, etc. coloriert.
* Im letzten Schritt wird die Fertig gerasterte Form mithilfe des Depth Tests in das Gesamtbild geblendet.

Level of Detail

* Damit der Planet immer in einer detaillierten Auflösung mit guter Performance angezeigt werden kann, ist die Oberfläche mithilfe eines Quadtrees (Baum mit jeweils vier Ästen) in einzelne Chunks (Stücke des Terrains) aufgeteilt.
* Dieser Baum wird je nach Abstand der Kamera vom Chunk immer weiter verfeinert oder vereinfacht.
* Wenn ein Chunk zu groß ist, werden vier kleinere Chunks der gleichen Auflösung erzeugt und ersetzen diesen.
* Die Erzeugung eines Chunks wird von einem Threadpool (bestimmte Anzahl von Threads, welche bestimmte Aufgaben nach der Reihe abarbeiten) übernommen und beeinflusst somit nicht die Performance des Programms.

Projektion

* Der Planet besteht aus einem Würfel, dessen Seiten aus Quadtrees bestehen. Dieser Würfel wird dann auf eine Kugeloberfläche projiziert.
* Dafür werden alle Ortsvektoren zu den Punkten der Würfeloberfläche normalisiert. Später werden diese Vektoren noch auf den Radius des Planeten skaliert.

Implementierung der 3D Noise

* Um nun den projizierten Punkten eine bestimmte höhe zu geben, wird eine 3D Noise benötigt.
* Dazu wird die normalisierte Kugel bzw. das Mesh des Planeten mit der Noise überlagert und die Punkte auf der Kugeloberfläche in der Noise repräsentieren die Höhen des Terrains an diesen Stellen.

Modifikation der Noise

* Der Output der Noise wird durch eine Funktion so verändert, dass die Flachen stellen des Terrains auch möglichst Flach bleiben, aber die höheren Stellen umso höher sind um Gebirge zu generieren.
* Diese Funktion ist eine Modifikation der Sigmoid Funktion.
* Außerdem sollte man viele Oktaven mit zunehmender Frequenz benutzen um einen möglichst natürlichen Eindruck des Terrains zu vermitteln.

Animiertes Wasser

* Um nun Wellen im Wasser zu generieren habe ich eine vierdimensionale Noise genutzt, bei der die ersten drei Dimensionen die Position auf der Kugeloberfläche darstellen und die vierte Dimension die Zeit.
* Durch fortschreiten der Zeit wird nun der dreidimensionale Raum animiert.

Veränderung in Echtzeit

* Alle Parameter zur Generierung des Planeten können mithilfe einer GUI in Echtzeit verändert werden. Diese Parameter werden dann der GPU (Grafikkarte) übergeben um mithilfe der GLSL Shader von OpenGL das Terrain zu generieren.
* Diese Berechnungen werden jeden Frame (Renderdurchgang) neu ausgeführt. Das hat zur Folge, dass die Änderungen ohne Wartezeit sichtbar sind.
* Außerdem können die Parameter eines Planeten in einer Datenbank gespeichert werden. Und aus dieser auch wieder geladen werden.

Raytracing

Ein Bild, das Text enthält.

Mit hoher Zuverlässigkeit generierte BeschreibungDas Raytracing ist eine vollkommen andere Rendermethode als OpenGL und oft nicht so Effizient. Jedoch können mit dieser Methode sehr komplexe Szenen mit vielen Lichtern und reflektierenden Oberflächen sehr realistisch Dargestellt werden. Außerdem können sämtliche Physikalische Effekte mit einbezogen werden, wie zum Beispiel die Brechung von Licht in transparenten Körpern.

Funktionsweise

* In der Szene befindet sich ein Pixelraster, welches das zu rendernde Bild widerspiegelt. Die Kamera befindet sich zentral mit etwas Abstand davor.
* Jeder Pixel des Bildes wird einzeln berechnet, indem ein Ray (Strahl) von der Kamera durch den Pixel geschossen wird. Anschließend werden alle Kollisionen des Strahls mit den Objekten der Szene berechnet und je nach Material endet der Strahl entweder oder er wird reflektiert bzw. gebrochen. Diese Vorgänge müssen Rekursiv berechnet werden.
* Bei jedem Schnittpunkt wird Außerdem ein Ray vom Schnittpunkt zur Lichtquelle geschossen und auf Kollisionen überprüft. Ist ein Objekt dazwischen befindet sich der Schnittpunkt in einem Schatten.

Kontakt

Vielen Dank.

Weitere Fragen gerne per E-Mail an:

[robingottschalk1402@gmail.com](mailto:robingottschalk1402@gmail.com)

Projekt auf GitHub unter:

https://github.com/paramatedev/worldgen