WP Computergrafik

Abschlussprojekt Exposé

Simulation von Landschaftstopologien mithilfe einer fraktalen Rauschfunktion auf der Basis pseudozufälliger Gradientwerten

Exposé

Student: Adrian Helberg

MatrikelNr.: 2309051

Prüfer: Prof. Dr. Philipp Jenke

Abgabe: 2. Februar 2018

Inhaltsverzeichnis

1 Motivation

Die Generierung von Landschaftstopologien findet zahlreiche Anwendung in der Entwicklung von Computerspielen. Um eine der Natur ähnlichen Umgebung zu simulieren, können mithilfe von fraktalen Rauschfunktionen Muster berechnet werden, welche in Landschaften mit natürlichen Phänomenen wie Wolken, Berge und Wasser, übersetzt werden können.
Damit eine simulierte Spielwelt nicht allzu abstrakt wird und der Vorstellungskraft eines Spielenden entspricht, wird auf oben genannte Muster zurückgegriffen, um so eine Spielwelt zu erschaffen, welche der Natur ähnlich ist.

2 Ziele

Im Rahmen des WP Projektes „Computergrafik“ beschäftigt sich diese Arbeit mit einer Implementierung eines erweiterten „Perlin Noice“ -Algorithmus auf der Basis des im gleichnamigen Praktikums zur Verfügung gestellten Kernprojektes.

2.1 Zielsetzung

Ziel dieser Arbeit ist eine Generierung simulierter, randomisierter Spielwelten, welche mit verschiedenen Farben, die repräsentativ für die Natur funktionieren (Grün für Wiesen, Blau für Wasser, Grau für Gestein usw.), und festen Assets (Bäume, Steine usw.) „zusammengebaut“ wird.
Des weiteren werden verschiedene Gebäude-Assets dazu verwendet Dörfer in der generierten Spielwelt zu gestalten und sie in geeigneter Lage zu platzieren.

Dörfer sollen durch eine Straße miteinander verbunden werden.

2.2 Ergebnis

Eine oben genannte, generierte Spielwelt soll mit dem Vuforia® - Projekt auf dem Smartphone dargestellt werden können. Beispielsweise über einen Marker.
Wie eine so dargestellte Spielwelt in etwa aussehen könnte, kann der Praktikumsaufgabe 4 (Aufgabenblatt 4: Kurven und Splines) entnommen werden. Hier sieht man eine Szene, in der Berge, Wolen und Bäume dargestellt und farblich hervorgehoben sind. Diese Szene kann als Beispiel für eine generierte Spielwelt dieses Projekts betrachtet werden.
Wie ähnlich am Ende beide Szenen sein werden, kann man hier noch nicht genauer spezifizieren.

Die Position der Dörfer wird anhand einiger Kriterien berechnet:

- Neigung des Terrains
  - Das Terrain sollte möglichst horizontal sein
  - Keine Dörfer an Berghängen; Es sollen Täler entstehen
  - Ein Grenzwert soll ermittelt werden, ab wann sich die Positionierung eines Dorfes nicht mehr eignet
- Flussnähe
  - Nach Abbild der Wirklichkeit sollen Dörfer in der Nähe der Ressource „Wasser“ sein
- Straßenanbindung
  - Dörfer sollen miteinander verbunden werden (Straßenverbindung)
  - Die Dörfer sollen anhand der Möglichkeit einer Straßenanbindung positioniert werden.

3 Algorithmus

Der folgende Pseudocode beschreibt die Implamentation des „Perlin Noice“ Algorithmus.

3.1 Pseudocode

```
// Function to linearly interpolate between a0 and a1
// Weight w should be in the range [0.0, 1.0]
function lerp(float a0, float a1, float w) {
    return (1.0 - w)*a0 + w*a1;
}

// Computes the dot product of the distance and gradient vectors.
function dotGridGradient(int ix, int iy, float x, float y) {

    // Precomputed (or otherwise) gradient vectors at each grid node
    extern float Gradient[IYMAX][IXMAX][2];

    // Compute the distance vector
    float dx = x - (float)ix;
    float dy = y - (float)iy;

    // Compute the dot-product
    return (dx*Gradient[iy][ix][0] + dy*Gradient[iy][ix][1]);
}

// Compute Perlin noise at coordinates x, y
function perlin(float x, float y) {

    // Determine grid cell coordinates
    int x0 = floor(x);
    int x1 = x0 + 1;
    int y0 = floor(y);
    int y1 = y0 + 1;

    // Determine interpolation weights
    // Could also use higher order polynomial/s-curve here
    float sx = x - (float)x0;
    float sy = y - (float)y0;

    // Interpolate between grid point gradients
    float n0, n1, ix0, ix1, value;
    n0 = dotGridGradient(x0, y0, x, y);
    n1 = dotGridGradient(x1, y0, x, y);
    ix0 = lerp(n0, n1, sx);
    n0 = dotGridGradient(x0, y1, x, y);
    n1 = dotGridGradient(x1, y1, x, y);
    ix1 = lerp(n0, n1, sx);
    value = lerp(ix0, ix1, sy);

    return value;
}
```

Perlin Noise `https://en.wikipedia.org/wiki/Perlin_noise`

4 Vorgehen

Da es nur schwer vorherzusagen ist, wie lange eine Implementation der geplanten Ziele dauert und auf welche Probleme bei der Entwicklung gestoßen wird, wird dieses Projekt mit dem sogenannten „Prototyping “, eine Methode der Softwareentwicklung, erarbeitet.
Genauer handelt es sich um ein exploratives Prototyping, welches zur Bestimmung der Anforderungen und zur Beurteilung bestimmter Problemlösungen verwendet wird und sich dabei auf die Funktionalitäten des Systems konzentriert.

4.1 Herausforderung

Die größte Herausforderung besteht hier bei dem Zusammenspiel zwischen den mathematischen Funktionen und dem Aufbauen der Spielwelt nach dessen Ergebnissen.

4.2 Zeitplan

Das Abschlussprojekt erstreckt sich über folgende Meilensteine:

Datum	Beschreibung
bis 04.02.2018	Planung / Exposè
bis 12.03.2017	Präsentation + Abgabe

5 Quellen

Perlin Noise	`https://en.wikipedia.org/wiki/Perlin_noise`
Understanding Perlin Noise	`http://flafla2.github.io/2014/08/09/perlinnoise.html`
The Perlin noise math FAQ	`https://mzucker.github.io/html/perlin-noise-math-faq.html`