Simulation von Emissionen in 2 und 3 dimensionalen Raum

Jonas Tietz

30. Oktober 2016

Content

Idee

- Simulation von Emissionen anhand verschiedener Faktoren
  - z.b. Luftdruck, Temperatur, Wind, Gravitation, Dichte
- Für ein interessanteres Strömungsverhalten etwas wie Hindernisse
- Möglichkeit von außen Einfluss auf die Simulation zu nehmen.
  - Winde setzen, Teilchenmenge erhöhen etc.

Algorithmische Lösung

Um die Emissionen zu Simulieren teile ich den Raum in Zellen auf. Jeder dieser Zellen speichert die Parameter, die Simuliert werden oder zu Simulation benötigt werden.
Eine Zelle könnte z.b so aussehen:

```
1  typedef struct sim_cell
2  {
3      int ParticleCount;
4      vector Velocity;
5      float Temp;
6  }sim_cell
```

Algorithmische Lösung

Die Idee der Simulation ist recht einfach. Man hat einen bestimmten Zeitschritt, den man mit der Mainloop immer voranschreitet. In der Loop geht man dann durch jede Zelle und updatet sie. Aber erst nur in einer 2ten Version, da die anderen Zellen den alten Wert noch brauchen.

```
1  void main(int argc, char **argv)
2  {
3      sim_cell Cells[Dim][Dim];
4      sim_cell TempCells[Dim][Dim];
5
6      for(int Iteration = 0;
7          Interation < IterationCount;
8          Iteration++)
9      {
10         for(int y = 0; y < Dim; y++)
11         {
12             for(int x = 0; x < Dim; x++)
13             {
14                 UpdateCell(x, y, Cells, TempCells);
15             }
16         }
17         memcpy(Cells, TempCells, (sizeof(sim_cell)*Dim*Dim))
18     }
19 }
```

Algorithmische Lösung

Die ganze Magie passiert nun in der UpdateCell-Funktion. Hier soll nun die aktuelle Zellen mit den Umliegenden verglichen werden und die Kräfte, die auf die Partikel wirken mit der schon vorhandenen Geschwindigkeit verrechnet werden. Dann soll anhand des entstandenen Vektor ausgerechnet werden welche Zelle oder Zellen wieviele Partikel bekommen.
Eine Kraft wäre z.b der Druckgradient oder die Gravitation.

Zeitplanung

1. Auslegen der Mainloop
2. Iteratives testen verschiedener Faktoren in der Update-Funktion im sequentiellen Code, bis ein zufriedenstellendes Ergebnis zustande kommt
3. Erweitern auf 3d falls Zeit vorhanden ist
4. Parallelisieren

Die meiste Zeit soll zum Testen(Punkt 2) und zur vernünftigen Parallelisierung(Punkt 4) genutzt werden.

Ergebnis

Leider konnte ich nicht alles Implementieren wie geplant. Aber die wichtigsten Sachen haben es ins Programm geschafft. Ich konnte durch Parallelisierung einen SpeedUp erreichen fraglich ist nur ob dieser den zusätzlichen Overhead kompensiert.