Graph Autolayout in Three Dimensional Space

Project Overview

This project is to implement an algorithm which automatically produces layout for graphs in three-dimensional space. The setting of this project are a three-dimensional graph editor, the OpenGL interface from the purely functional programming language Haskell, and the development version of the Haskell compiler GHC 6.3.

The layout algorithm used in this project is called force directed. This algorithm can produce nice layouts, often highly symmetric drawings, and tends to distribute vertices evenly.
**Force-Directed Algorithms**

Force-directed algorithm was introduced by Quinn, Breuer [QuBr79] and Eades [Eade64]. These algorithms use a physical analogy to draw graphs. A graph can be treated as a multi-spring system where we replace all edges by springs. Because of the nature of spring a multi-spring system always tends to reach the lowest energy configuration, with such configuration layout of the graph is reasonably nice. Therefore the algorithm is to seek a configuration of a system with minimal energy.

In general, there are two parts involving force-directed algorithm.

- The model: a force system defined by the vertices and edges, which provides a physical model for the graph.
- The Algorithm: a technique for finding lowest energy configuration of a physical model.

The aesthetic criteria depend on the model and efficiency depends on the algorithm. In this project we use the following equation to model the spring force between any two nodes.

\[ F = k \sum_{i=1}^{n} \frac{(p_i - p_j)^2}{-\text{len}^2} \]

K is the force constant of the spring. \( \|p_i - p_j\| \) is the distance in the space whereas len is shortest path between the two nodes.

With force model energy of whole system can be formulated as the following.

\[ E = \sum_{j=1}^{n} \sum_{i=1}^{k} K_{ij} (\|p_i - p_j\|^{-\text{len}^2})^2 \]

This equation has \( 3n \) variables, \( n \) is how many nodes. In order to find local minimum we have to solve \( 3n \) nonlinear equations and check all concavities, absolute minimum only can be found when all local minima have been found. Increasing the number of nodes significantly decreases efficiency. Alternatively, there is a more efficient algorithm which finds a node with the greatest gradient in current configuration, then fixes the remaining nodes and moves the found node to a position where there is a local minimum. This process continues until there is no node gradient is greater than a preset value. The technique to position a node is to find a zero gradient vector by applying the Newton- Raphson- Algorithm (NRA).

However, the NRA does not always terminate. To solve this problem, we use the simulated annealing method, which forces NRA to converge.

---

**Functional Programming — Haskell**

Both this project and the original graph editor are written in Haskell. Haskell is the most popular pure functional programming language (FPL). In particular, it is a polymorphically typed, lazy, purely functional language, quite different from most other programming languages. The language is named after Haskell Brooks Curry, whose work in mathematical logic serves as a foundation for functional languages.

FPL has many advantages, such as brevity, ease of understanding, higher-order functions and so on. The following are two functions, one is written in Haskell the other is C.

**Quicksort in Haskell**

```haskell
qsort []     = []
qsort (x:xs) = qsort elts_lt_x ++ [x] ++ qsort elts_greq_x

where
elts_lt_x = [y | y <- xs, y < x]
elts_greq_x = [y | y <- xs, y >= x]
```

**Quicksort in C**

```c
int qsort( a, lo, hi )
    int a[], hi, lo;
    if (lo < hi) {
        l = lo;
        h = hi;
        p = a[hi];
        do {
            while ((l < h) && (a[l] <= p))
                l = l+1;
            while ((h > l) && (a[h] >= p))
                h = h-1;
            if (l < h) {
                t = a[l];
                a[l] = a[h];
                a[h] = t;
            }
        } while (l < h);
        t = a[l];
        a[l] = a[hi];
        a[hi] = t;
    } else {
        qsort( a, lo, l-1 );
        qsort( a, l+1, hi );
    }
```

[Copied from www.haskell.org]