Swarm and Evolutionary Computation 26 (2016) 56–63

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

journal homepage: www.elsevier.com/locate/swevo

Regular Paper

A parallel Fruchterman–Reingold algorithm optimized for fast visualization of large graphs and swarms of data

Petr Gajdoš, Tomáš Ježowicz, Vojtěch Uher, Pavel Dohnálek *

Department of Computer Science, FEECS VŠB – Technical University of Ostrava, 17. listopadu 15, 708 33 Ostrava, Czech Republic

ARTICLE INFO

Article history:
Received 29 October 2014
Received in revised form
16 July 2015
Accepted 22 July 2015
Available online 14 August 2015

Keywords:
Graph layouts
Nearest neighbors
Fruchterman–Reingold
Fast graph visualization
GPU
CUDA

ABSTRACT

Graphs in computer science are widely used in social network analysis, computer networks, transportation networks, and many other areas. In general, they can visualize relationships between objects. However, fast drawing of graphs and other structures containing large numbers of data points with readable layouts is still a challenge. This paper describes a novel variant of the Fruchterman–Reingold graph layout algorithm which is adapted to GPU parallel architecture. A new approach based on space-filling curves and a new way of repulsive forces computation on GPU are described. The paper contains both performance and quality tests of the new algorithm.

© 2015 Elsevier B.V. All rights reserved.

1. Introduction

Graphs can bring a new view of the data structure and relationships between its elements. They can visualize some additional information, e.g. clusters, that are hidden by default. However, graphs become more complex and unreadable with a growing amount of data and it is evident that a naive visualization of complex graphs leads to loss of information, e.g. the user can see just a clutter of nodes and edges. Some layout algorithm must be used to provide a readable form of the graph structure. The main disadvantage of using layout algorithms lies in the additional computation time, which can be very long in the case of a large amount of data. Fast graph drawing with a readable layout is still a challenge [1]. For example, it is practically impossible to visualize large social networks [2] like Facebook and Twitter. and illustrate the dynamic of such networks. The same holds true for the visualization of WWW pages connected by links), computer networks [3], or protein similarity [4], or even visualizing large quantities of particles in Particle Swarm Optimization [5], ants during Ant Colony Optimization [6], genes in Genetic Algorithms [7], etc.

Several layout algorithms exist [8–10] that can be used to create an acceptable graph visualization. Usually, more layout algorithms can be applied on the same graph to provide variants for the readers. The layout should ideally help the reader to better understand the information contained in the graph. Several aesthetic techniques or metrics like edge crossing, line bends, symmetry, minimum angle or orthogonality are presented in [11]. Ref. [12] shows which of the aesthetic criteria have the greatest influence for human understanding. Note that for some graphs, even if one layout has no edge crossings, a different layout with more crossings may be considered as a better one (showed in [8]). The more aesthetic criteria are required, the more computation time is usually needed.

The paper is organized as follows. Section 2 presents the related work. In Section 3 the nearest neighbors (NN), space-filling curves (SFCs), graph layout problem and Fruchterman-Reingold (FR) algorithm are described. In Section 4, the proposed algorithm is discussed as well as the obtained speed-up in the final implementation. Finally, the last section summarizes the performance and quality of the results and experiments.

2. Related work

Some of the graph layout algorithms represent a class of the so-called force-directed layouts, e.g. Fruchterman–Reingold [9] or Kamada–Kawai [8]. These algorithms iteratively change the positions of vertices/nodes to reduce a defined energy function, also called the temperature *T*. These layouts are generally considered to be aesthetic. The problem related to the usage of the force-

* Corresponding author.
E-mail addresses: petr.gajdos@vsb.cz (P. Gajdoš), tomas.jezowicz.st@vsb.cz (T. Ježowicz), vojtech.uher@vsb.cz (V. Uher), pavel.dohnalek@vsb.cz (P. Dohnálek).

http://dx.doi.org/10.1016/j.swevo.2015.07.006
2210-6502/© 2015 Elsevier B.V. All rights reserved.

directed algorithms is that in general they are computationally expensive. The paper [2] describes the speed-up of the Fruchterman–Reingold by computing the most expensive part on GPU.

Our research focused on the method of finding the nearest neighbors for every vertex to achieve more effective computation. Thus the solution of the nearest neighbors problem described for example in [13,14] became the primary goal of our research. One of the fast methods for nearest neighbors searching is based on the space-filling curves. The space-filling curves used in the graph layout can be found in [15], where a new approach dealing with dense graphs was presented. Moreover, searching for new layout algorithm leads to the consequent challenge that consists of comparison techniques. When a new layout algorithm is found, the natural question is how to compare it with others. Ref. [11] shows several quality measurements for graph layouts. Next, in [12], quality measurements were studied from the point of view of human reading.

3. Background of the suggested visualization

In this section a selected space-filling curve is briefly described with primary focus on the problem of computation of graph vertices/nodes coordinates. Also the Fruchterman–Reingold algorithm is mentioned.

3.1. Nearest neighbors and space-filling curves

Space-filling curves (SFCs) were used in order to get the nearest neighbors, because they can be computed very fast in parallel way. Several variants exist such as Peano, Z-order or Hilbert space-filling curve [16–18] (Fig. 1).

Note that SFCs are very inaccurate in classifying nearest neighbors. Generally, SFCs connect the points that are close to each other and thus transform the n-dimensional problem into one dimensional (1D). Unfortunately, many points should be considered as nearest neighbors but ultimately they are far from each other on the SFC. On the other hand, some vertices lie very close together on the curve but in reality they are far away. The mentioned disadvantage is balanced with the fact that the SFC can be computed very easily and in parallel. It requires just computing a single index for each point ($\Theta(N)$) and then reordering such indices, e.g. with quick-sort or with parallel radix sort using CUDA. An example of Z-order SFC can be seen in Fig. 2.

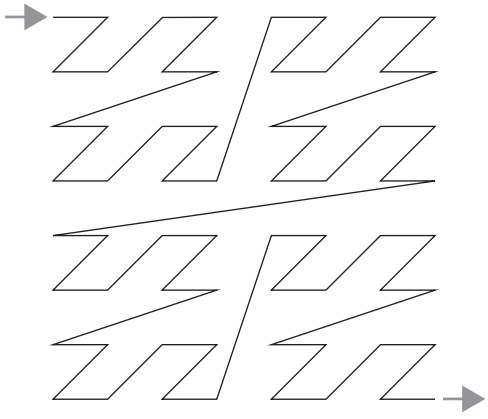

Fig. 1. Example of Z-order space-filling curve.

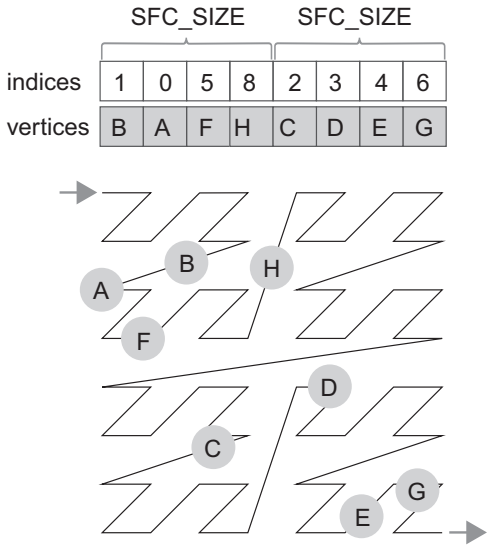

Fig. 2. 8 vertices (A, B, C, D, E, F, G, H) randomly placed and ordered by Z-order space-filling curve.

3.2. Graph layout and Fruchterman–Reingold

Let a graph $G \leftarrow (V, E)$ be a set of vertices V and edges E that connects those vertices. One of the well known algorithms for the graph layout is the Fruchterman–Reingold, which belongs to the family of force-directed graph layout algorithms. Vertices connected by an edge attract each other. It also defines an ideal distance for each vertex. The vertices should be drawn near each other, but not too close. To lay out a graph, the vertices are replaced by steel ring and each edge with a spring mechanical system [9]. The complete algorithm is shown below:

$area \leftarrow$ W * L {width and length of the frame}
$G \leftarrow (V, E)$ {random initial positions for the vertices}
$k \leftarrow \sqrt{area/|V|}$
function $F_a(z) \leftarrow$ {return x^2/k}
function $F_r(z) \leftarrow$ {return k^2/z}
for $i \leftarrow 1$ **to** *iterations* **do**
  {calculate repulsive forces}
  for all $v \in V$ **do**
    {each vertex has two vectors: .pos and .disp}
    $v.disp \leftarrow 0$
  for all $u \in V$ **do**
    if $u \neq v$ **then**
      {Δ is short hand for the difference}
      {vector between positions of the two vertices}
      $\Delta \leftarrow v.pos - u.pos$
      $v.disp \leftarrow v.disp - (\Delta/|\Delta|) * F_r(|\Delta|)$
    end if
  end for
  end for
  {calculate attractive forces}
  for all $e \in E$ **do**
    {each edge is an ordered pair of vertices .v and .u}
    $\Delta \leftarrow e.v.pos - e.u.pos$
    $e.v.disp \leftarrow e.v.disp - (\Delta/|\Delta|) * F_a(|\Delta|)$
    $e.u.disp \leftarrow e.u.disp - (\Delta/|\Delta|) * F_a(|\Delta|)$
  end for
  {limit the max displacement to the temperature T}
  {and then prevent from being displaced outside frame}
  for all $v \in V$ **do**

Download English Version:

https://daneshyari.com/en/article/493920

Download Persian Version:

https://daneshyari.com/article/493920

Daneshyari.com