Université Bordeaux 1
351, cours de la Libération F-33405 Talence cedex

PER PROJECT REPORT

Déformation de Grille pour la visualisation d'information

A.Lambert, R.Bourqui, D. Auber

January 13, 2012

Clients:
David Auber
Romain Bourqui

Students:
Barro Lissy Maxime
Manano Camille
Tombi A Mba James
Zenati Omar
Rozar Fabien

Engineering students
enseirb-matmeca 2011/2012

Abstract

The article we are currently working on deals with how to visualize graphs containing many nodes and edges. With huge amount of data generally comes visual clutter, in our case due to edges crossing. This solution is based on an edge bundling technique coupled with a grid built from the original graph. Authors also used a GPU-based rendering method to highlight bundles densities without losing edge color. We use in our solution the Tutte algorithm in order to quickly obtain a graph without crossing, based on a triangular-face grid, which helps to read relationship between nodes. This algorithm is available as a standalone program for Tulip.

CONTENTS

Introduction

The article talks about how to visualize graphs containing many nodes and edges. With improvements in data acquisition comes an increase of the size and the complexity of graphs and this huge amount of data generally causes visual clutter, in our case due to edges crossing. For example, it could be interesting to visualize data in fields like biology, social sciences, data mining or computer science and then emphasize their high-level pattern to help users perceive underlying models.

Nowadays, in the research world, the information is easily represented into graphs to visualize more and more data. However, this huge amount of information prevents the graph from being manually drawn: It explains the need of software able to generate an appropriate graph with all nodes and edges. Yet this graph may suffer from cluttering, which should be reduced for a better understanding.

(parler du travail réalisé et de son but)

The first part of this document presents review related work on reducing edge clutters and enhancing edge bundles visualization, with which the article is connected. The second will deal with the Tutte algorithm , followed by the implementation issues. A third part will show our results. Finally, we draw a conclusion and explain the limits of our work for further improvements.

CHAPTER 1

The context

Some classes of graph, such as trees or acyclic graphs, clearly facilitate user understanding by effective representation. However, most of graphs do not belong to these classes and algorithm giving nice results in terms of time and space complexity but also in terms of aesthetic criteria for any graph do not exist yet. For example, force-directed method produces pleasant and structurally significant results but do not help users comprehension due to data complexity. Authors of the paper specify two techniques for that reduction: the compound visualization and the edge bundling. But their interest goes to Edge Bundling which suggest to route edges into bundles in order to uncover high-level edge patterns and emphasize information. Their contribution was to set this edge bundling by discrediting the plane into a new specific domain where the graph is set so that boundaries of the new discretization.

Up to now, several techniques were used to reduce this clutter, based on compound visualization or edge bundling. In a compound visualization, nodes are gathered into metanodes and inter-cluster edges are merged into metaedges. To retrieve the information, metanodes could be collapsing or expanding. Edge bundling technique routes edges into bundles. This uncover high level edge pattern and emphasize relationships. Yet an important constraint is the impossibility for some nodes to move while avoiding edges crossing because node positions bring information: the compound visualization is consequently not suitable.

Some existing representations take into account this duty: some reducing edge clutters (Edge routing, Interactive techniques, Confluent Drawing, Node clustering, Edge clustering) and the other enhancing edge bundles visualization (Smoothing curves, Coloring edges). Edge routing use shortest-path edge routing to bound the number of edge crossings and use non-point-size to avoid node-edge overlaps. It do not highlight the underlying model. Interactive techniques remove clutter around the user's focii in a fisheye-like manner while preserving node position: this technique does not reduce the clutter of the entire representation. In Confluent Drawing, groups of crossing edges are drawn as curved overlapping lines. Node Clustering routed edge along the hierarchy tree branches. Both methods cannot be applied to any graph. Edge Clustering route edges either on the outer face of the circle or in its inner faces and bundle them to optimize area utilization.

The publication we are currently working on is based on a bundling edges method using quad-tree discretization, Voronoï diagrams and Bezier curves to visualize aesthetic graphs. By using the shortest path Dijskstra algorithm and optimizing the algorithm using efficiently the architecture of computer, the obtained graphs are also quickly computable. The Bezier curves are quite interesting because they reduce the "zigzag" effect which appears in edge weighting, However, the graph produced by this Bezier curves are very poor in information: because of

edges superposition over nodes, information is lost when edge weights are adjusted. Moreover, the graph is not as "aesthetical" as it should be.

CHAPTER 2

Tutte

2.1 Tutte's algorithm

This part of text comes from the article [1].

In this paper, we will use basic graph theory terminology, see for example [2]. Let $G = (V, E)$ be a planar graph. A mapping Γ of G into the plane is a function $\Gamma : V \cup E \rightarrow P(\mathbb{R}^2)$ which maps a vertex $v \in V$ to a point in $\mathbb{R}^2$ and an edge $e = uv \in E$ to the straight line segment joining $\Gamma(u)$ and $\Gamma(v)$. A mapping is an embedding if distinct vertices are mapped to distinct points, and the open segment of each edge does not intersect any other open segment of an edge or a vertex.

In 1963, Tutte [3] gave a way to build embeddings of any planar, 3-connected graph $G = (V, E)$. Let C be a cycle whose vertices are the vertices of a face of G in some (not necessarily straight-line) embedding of G. Let Γ be a mapping of G into the plane, satisfying the conditions:

- the set Ve of the vertices of the cycle C is mapped to the vertices of a strictly convex polygon Q, in such a way that the order of the points is respected;
- each vertex in $V_i = V\ V_e$ is a barycenter with positive coefficients of its adjacent vertices (Tutte assumed all coefficients to be equal to 1, but the proof extends without changes to this case). In other words, the images v of the vertices v under Γ are obtained by solving a linear system (S): for each $u \in V_{i,v|uv\in E}\lambda_{uv}(u - v) = 0$, where the λ_{uv} are positive reals. It can be shown that the system (S) admits a unique solution.

Theroem 1 *(Tutte's Theorem)* Γ *is an embedding of G into the plane, with strictly convex interior faces.*

2.2 Tutte Sequential

2.3 Tutte Parallel Algorithms

Tutte' s algorithme is ...

2.3.1 Asynchronous parallel version

Distribution of nodes: Graph coloring

In order to implement a parallel asynchronous version of Tutte algorithm, it is necessary to separate graph nodes into different sets. The objectif is to extract an independance between

nodes. In fact, each node have to move while maintaining neighbors positions. Thus, the independance must be between node and his neighbors. This problem is similar to the famous problem of graph colorating.

The objectif of the modified Tutte algorithm is to handle graph of thousands nodes. To separate such a number of nodes, it is more performant to use a heuristic of the algorithm of graph coloring.
The greedy algorithm is a simple and good solution to separate nodes into sets fastly and effictively.

The algorithm used in this project is :

```
G={V,E}
Y = V
color = 0
While Y is not empty
   Z = Y
   While Z is not empty
      Choose a node v from Z
      Colorate v with color
      Y = Y - v
      Z = Z - v - {neighbors of v}
   End while
   color ++
End while
```

Applying Tutte algorithm to sets

Once a sets of nodes is obteined, it is possible to apply a parallel Tutte algorithm. The question is how to parallelize it on sets of nodes. The natural idea is to attribute one sets per thread : This distribution is far from being optimal. In fact, each thread have to lock neighbors node

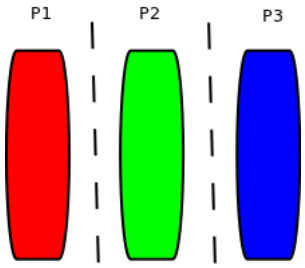

Figure 2.1: One set per thread

before moving it, wich introduce an importante critical section. In addition to being unfair, this distribution is limited by number of sets produced.

The best distribution for sets obteined by graph coloring is to execute n threads on one set. Each thread move an number of nodes of the set without any critical section. This is because each node of the set is not the neighbor of all others nodes of the same set. Once the thread finish moving all his nodes of the set, he must wait for others threads (implemented by a barrier). After, the same processus is applyed on the next set.

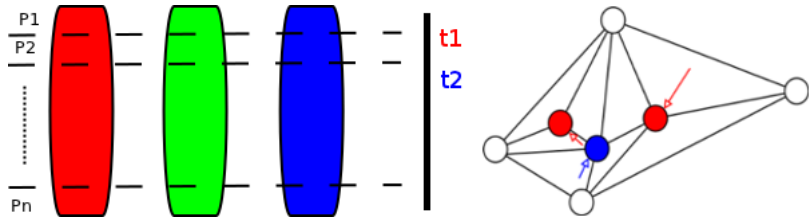

Figure 2.2: n threads per set

2.3.2 Synchronous parallel version

2.4 Tutte's algorithm on concave polygon

CHAPTER 3

Implementations

3.1 Data Structure

In the implementation of our sulution we have defined our own data structure on which we excute the Tutte's algorithm. We have implemented some mechanisms to convert a tulip format graph to our own graph structure and also to get informations from our structure inserted in a tulip format graph. In other words our structure is a temporary structure for storing informations about nodes in order to execute the Tutte's algorithm.

3.1.1 Issues

As a format tulip contains a lot of informations so it costs a lot to manipulate them and so far away we do not need all the informations about a given tulip graph. To point out we especially do not need all the properties about nodes to conduct the Tutte's algorithm.For intance, for a given node we just want to know if it is fixed for a fixed node's position nevers changes during the tutte's algorithme. In addition of that, as we are looking for performance, we need a light and adapted structure to the principe of Tutte's algorithm. Below are some of criterias which make us thought we need a new data structure.

1. The fact that a given node is fixed or not is indicated firsly by a mobility property. However there is anover property incating nodes wich are part of graph contouring and these nodes need to be fixed too. So to deal with the fact that a given node is fixed or not we need to manipulate two properties that cost a lot.

2. In tulip format there is an hierarchy of graphs or we only need the parent of the graph, the one which is not subgraph of another one. we do not need the sub-graph relation between graphs.

3.1.2 First implementation

In order to avoid memory leak and implemente easily the Tutte's algorithme, we merely stock in our sttructure only the informations needed to run the algorithm. We implemented our structure so that one can easily access the neighbourhood of a given node for it is very crucial in am Tutte'algorithm implementation. To do this we define a class that contains diferent informations needed on a given node (the attibuts) and all the operations we need to run on a node (the methods).

```
class MyNode {
 private:
  node n;
  bool mobile;
  Coord coord;
  vector<MyNode *> voisin;

 public:
  MyNode();
  MyNode(const node n, const Coord coord);
  MyNode(const node n, const bool mob, const Coord coord);
  ~MyNode();

  const node getNode() const;
  bool getMobile() const;
  void setMobile(const bool b);
  const Coord getCoord() const;
  void setCoord(const Coord &);
  vector<MyNode *> * getVoisin();
  vector<MyNode *> getVoisin() const;
};
```

The vertex's attributs needed

n : this attribut has `node` type of tulip library and contains the `ID` of the node.

mobile : this attribut has a `boolean` type and used to know a given node is considered fixed.

coord : this attribut has a `Coord` type of tulip library and is used to stocked the node coordinate.

voisin : this attribut has a `vector` type of `C++` library and contain the neighbourhood.

The operations on a vertex

We used two types of operations or methods: setter and getter. A setter is a method used to set the value of an attribut and a getter is used to get the value of an attribut. For a given attribut `attribut` , the cooresponding setter and setter are respectively `setAttribut(args)` and `getAttribut()`. Below are the lists of the setters and getters of nodes in our structure:

Setters : `getMobile(), getCoord(), getVoisin()`

Getters : `setMobile(const bool b), setCoord(const Coord &), getVoisin()`.

3.1.3 Second implementation

3.1.4 Third implementation

3.2 Results

Thank to our teacher, three graphs haven been given in order to test our different implementation of the Tutte method.

These graph have the follow characteristic :

Graph	number of vertices	number of edges
aiir_traffic	14693	63403
imdb	9488	33942
migration	14318	49460

For the first implementation with a basic structure to store the graph, for performance aspect, we obtain these results :

Graph	number of iterations	mean time of execution of Tutte's algorithm	standard deviation
aiir_traffic	61	0.2526	0
imdb	473	1.2957	0
migration	5	0.006314	0

3.3 Complexity

3.4 Benchmark

Conclusion

With our best Tutte algorithm implementation , we obtain an algorithm convergence in 5 iterations and it spends x seconds to recalculate all the coordinates. **(à mettre à jour)** Finally, our optimizations allowed an improvement of the clutter reduction and the performance compared with existent methods.

After discussing with our teachers in charge, some issues concerning our standalone version remains unresolved and could be taken into account in the future: Rather than working with input graph nodes, it would be interesting to dynamically add nodes (and their edges to keep a triangular-face graph) and launch again the algorithm in order to refine given results. It will also be quite interesting to automatically join small triangles or divide big triangles into littles ones.

(à rajouter: plugin, prouver que ca marche: peut etre dû à la construction de la grille (triangulation de delaunay + contrainte: deux sommets fixes ne sont jamais reliés par une arête sauf sur le contour))

BIBLIOGRAPHY

[1] E. Colin de Verdière, M. Pocchiola, and G. Vegter. Tutte's Barycenter Method applied to Isotopies. *Computational Geometry: Theory and Applications, 26*, 81–97, 2003.

[2] B. Bollob's. Modern graph theory, *volume 184 of Graduate Texts in Mathematics*. Springer-Verlag, 1998.

[3] William T. Tutte. How to draw a graph. *Proceedings of the London Mathematical Society*, 13:743–768, 1963.

[4] A. Lambert, R. Bourqui, and D. Auber. Winding roads: Routing edges into bundles. *In 12th Eurographics/IEEE-VGTC Symposium on Visualization (Computer Graphics Forum; Proceedings of EuroVis 2009).* To appear., 2010.

[5] A. Lambert, R. Bourqui, and D. Auber. 3D edge bundling for geographical data visualization. *In IV '10: Proceedings of the 14 International Conference on Information Visualisation (IV'09)*, Washington, DC, USA, 2010. IEEE Computer Society.

(Reference : Introduction to Algorithms (Cormen, Leiserson, Rivest, and Stein) 2001, Chap- ter 16 "Greedy Algorithms".) (rajouter reférence d'omar) (refaire biblio en se centrant plus sur tutte)